Demystifying Security Enhanced (SE) Linux

BY ABHISHEK SINGH, CISSP # 82044

Prologue

In this paper I will try to explain the philosophy behind the Security Enhanced Linux (SE Linux). I will however try to explain the concept with an example but to keep the length readable I will restrain myself to go into much of implementation details for e.g. commands and similar stuff.

What is SE Linux?

This flavor of linux has strong Mandatory Access control Built into the kernel where by the process and objects such as files are classified based on the confidentiality and integrity requirement, hence the affect of a security breach is reduced to minimal.

It is to be noted that this doesnot mean that SE Linux was designed to correct flaws which are present in the Linux rather it's an attempt to use MAC (in contrast to DAC used by traditional Linux Systems) to make a system which will mitigate the affects of security policy breaches to a minimum, by the help of policies which specify the security requirements of a system.

Researchers in the Information Assurance Research Group of the National Security Agency (NSA) worked with Secure Computing Corporation (SCC) to develop a strong, flexible mandatory access control architecture based on Type Enforcement, a mechanism first developed for the LOCK system. The NSA and SCC developed two Mach-based prototypes of the architecture: DTMach and DTOS. The NSA and SCC then worked with the University of Utah's Flux research group to transfer the architecture to the Fluke research operating system. During this transfer, the architecture was enhanced to provide better support for dynamic security policies. This enhanced architecture was named Flask. The NSA has now integrated the Flask architecture into the Linux operating system to transfer the technology to a larger developer and user community.

Why SE Linux does exist in the first place?

To understand this we have to first understand the flaws in the traditional architecture of Operating systems. Most of the systems are built on the premise that security can be treated as an add-on to the OS however as we will see that a secure application will remain secure only if there is a secure OS underneath.

Now that the distinction between the data and the executables is diminishing malicious code can take many forms for e.g. an applet or a harmless apperaring data which actually has malicious code in it an example can be postscript document which can be harmul as they have access to the local file system a malicious program taking the advantage of this can have a priviledge escalation and hence compromising the security of the system, another nusances is the ActiveX Obejct whose security is based on the digital signature the flaw here is that once we say ok to an ActiveX object we typically can't restrict its execution domain this is All Or Nothing Approach. However the digital signatures do give us an idea of the trustworthiness of the Applet or ActiveX but can't be solely reiled upon for security.

Lets consider another example IPSec and SSL are typically used to provide the security in the networking world now imagine a scenario in which there is a malicious program resident on one of the end point which can tamper with the keys or bypass the legitimate software with its own malicious code compromises the security of the end point and if the security of one end point is screwed and direct access to unprotected data is achieved protection provided by IPSec and SSL is a myth. So an underlying operating system can help mitigate this to a large extent.

Giving controls to users phew ……

Security of the system can't be left to the users as in case of the present day traditional (non SE Linux) DAC based systems do. The flaw with this approach is that a small mistake on the users part can compromise the security of the overall system consider an example where a user can compromise a vulnerability in an application then run a program with setuid of root and gaining the root priviledges. Also in this approach where there is no concept of Security Domains a program executed by a user will have all the priviledges of the user, another headache.

So we can say that there are only two supported categories of the users "Priviledge" – Can do anything on the system and "untrusted/restricted" – Have limited access to the system facilities. It is interesting to note that many times services and priviledge programs run with priviledges which far exceed there requirement.

DAC also paves way for major damage by a 0-Day attack by the similar logic.

Base of a Secure Operating System

Now that we have understood the need of a secure OS lets understand the general components which can be used to make it.

1. **Mandatory Security:** This is a security policy in which the security policy and the security attributes are contolled by an administrator rather than user. Roles are defined for different operations which are necessary to be performed on system and users are assigned to them. Mandatory Security can be used to implement the organization wide security policies.

   These can be futher subdivided into:

   a. **Access Control Policies** – How Subject will access the objects under OS.
   b. **Authenticaion Usage Policies** – what are the acceptable mechanisms by which a subject can authenticate to the system.
   c. **Cryptographic Policies** – This defines means used to protect the data.

There can be covert channels in this system but they can be traced by proper auditing.

Trusted Processes – Processes which will need to perform many security related tasks in system. They are assigned rights to so by ***principle of least priviledge*** which will prevent system from the abuse commited by a trusted process.

Flow of data in this system is controlled by a sub system which verifies its accordance with the accordance to policy.

Domains – Applications are confined in strict domains which are separated by other domains, by help of which a misuse by an application can be confined to a single domain. Domains also provide a sand box to the untrustworthy applications.

Even in the Personal Computer Environment this approach can help protect against the malicious software whereby the oridinary role of the user which is distinct from the administrator, the extent of damage by malicious software is greatly reduced.

2. **Trusted Path:** This is a means by which a user can interact directly with trusted software and this can't be imitated by other software. In absence of this, malicious software can impersonate as a truted process or a user. Commercial softwares implement this only in limited functionality for example Windows NT/2K have this for there login and password changing but lacks support for providing this to other trusted applications.

   Need for this truted path can be seen from this simple example where user has to enter his PIN or Credit Card number and a Trojan Horse is waiting for such event to happen.

SE Linux to the Rescue

SE Linux has concept of Subects, Objects and Operations. Objects can be grouped into security classes called Object classes on which polices will be defined.

DAC system of the traditional Linux still has a major role to play, is an operation is not permitted by this than SE Linux will not come into the picture for example If read if not allowed on a file object for a particular user, it will be denied in the first place by DAC.

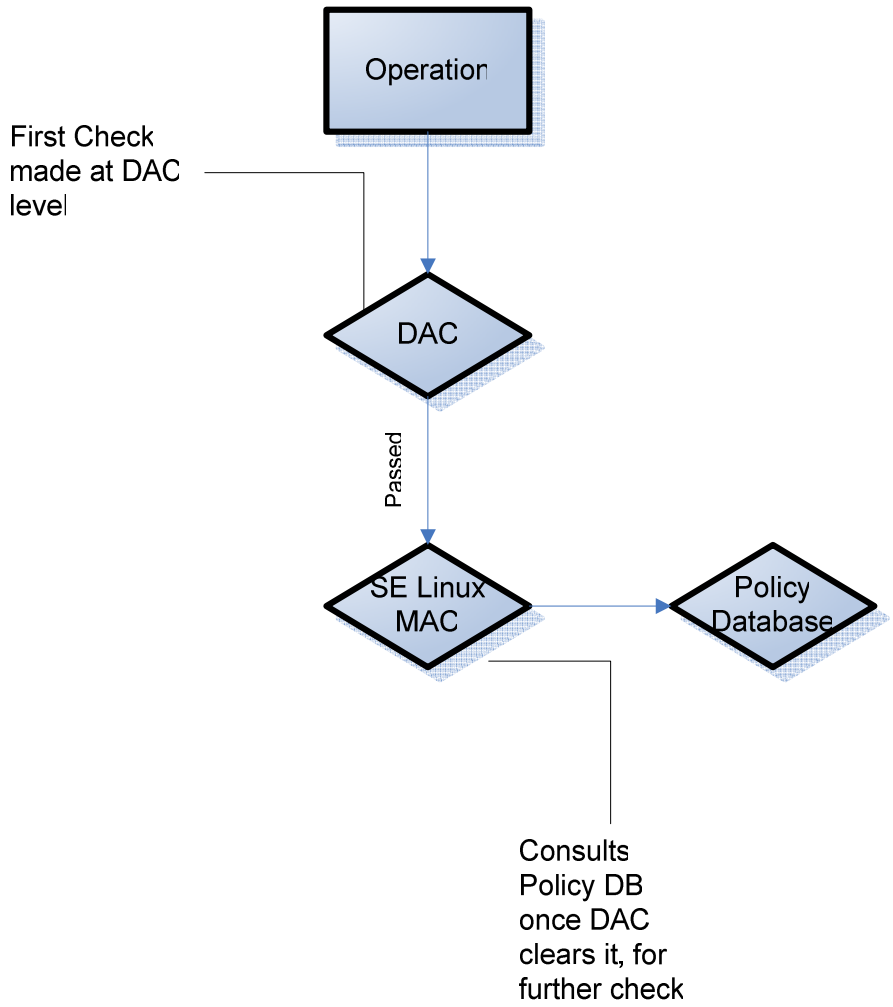

Fig 1. SE Linux Decision making process

Policy contains the definition of a domain and it also specifies the valid transitions from one domain to another.
(I will discuss about the policies later.)

SE Linux is implemented as a **Linux Security Module (LSM)** in which hooks are inserted in the kernel and when a process wants to perform an operation on an object this will be notified to SE Linux which can allow or disallow the process as per the policy.

To run a program securely in an SE Linux environement admin user needs to know every file the process opens and every subroutine the process calls.

Object and Subject Privildges Criteria

Priviledges of the objects and subjects (mainly processes) are decided on the base of a tuple consiting:

1. **Identity -** Each user and process has a unique Identity on the system. Changes to it are strictly controlled and policy only allows certain processes to change the identity like login and crond. For a user SE Linux has its own database and doesn't use the one reflected in */etc/passwd,* the obvious benefit of this approach is that the access control of one can be separated from another and so the changes in any will not affect the functioning of other. In the context of the objects like files, directories etc they have the identity of their owner.

   Actions of a user are strictly controlled by roles to which he is assigned as RBAC and transition from one role to another is supervised by the policy. Users can be identified by "**_u"** suffix for example in order to reduce the complexity of the system administrator can make a generic non-priviledge user **user_u** now the policies will be customized for the users who need more default priviledges than the user **"user_u"** . In terms of SE Linux the term user is actually used to represent the users who can interactively interact with the system, an exception to this is **"system_u"**

   One obvious result of this approach is that there is no need to have the pseudo-users as the processes can be run according the priviledges assigned to their respective type (Types are explained in detail later, for now consider it to be a mechanism to assign priviledges to an object.).

2. **Roles –** Used to specify acceptable actions from a user. Each role has a set of priviledges assigned to it; users are assigned to these roles. It is to be noted that at any given point of time user can only be in one role and transition between roles is strictly controlled by the policy. It is possible to have multiple processes and applications with in the context of a given role. Roles are typically made oon the basis of the task to be performed by it.

Following is a list of roles defined:

1. **staff_r –** Contains users who are permitted to enter **sysadm_r** role.
2. **sysadm_r –** Role defined for system administrator.
3. **system_r –** System processes run under the context of this role. Typically used for depicting objects and processes.
4. **user_r –** Role for normal users.

User can be assigned to a role by:

user abhi roles {staff_r sysadm_r}

Transition between roles can be allowed by:

allow staff_r sysadm_r

Roles can be assigned to types by:

role sysadm_r type ftpd_t

As can be seen roles can be identified the suffix "**_r".**

3. Type - This refers to the priviledges assigned to the object thereby assigning each object to a domain. Can be distinguished with a **"_t"** suffix.

SE Linux has a major dependence on the concept of domains and types. A mechanism called **Type Enforcement (TE)** assigns the processes with the domains and each object is assigned to a type. Permissions defines what can be done on a particular object and what is a process allowed to do, so domain can be said to be a set of processes having similar permissions. An **access matrix** is defined for these set of permissions. TE specifies what is permissible in a domain and observes that these domains are always followed.

This tuple is kept in a Database with the system and each entry is indexed by a **Security Index (SID)** which is used to make decisions, during the booting of the system only a handful of SID are loaded.

Type of Objects

1. **Transient Objects –** These objects will be destroyed after their use so they don't have a permanent SID and so a SID from memory is assigned.

2. **Persistent Objetcs –** These stay in the system till they are destroyed for e.g. files and directories so they need a Permanent SID which can be easily provided to them with the help of extended attribute feature of file systems like ext2, ext3 etc **PSID's** are stored in them.

Decisions made by Security Server

Two basic decisions are made by the Security Server, these are

I. Access Decisions:

Primarily these are based on the class of the object. Three **access vector** categories – **Allow, Audit Allow** and **Don't Audit,** are associated with each of the allowed action for e.g. if the object is a file then

File Class					
Allowed Actions	**Create**	**Execute**	**Link**	**Lock**	**Get Attribute**
Allow	-	x	-	-	x
Audit Allow	-	-	-	-	-
Don't Audit	-	-	X	-	-

Fig 2. Some attributes of the file class.

Access Vectors –

1. **Allow:** Specifies an operation permissible on the object, no log is made.
2. **Audit Allow:** Allows Auditing for permissible actions.
3. **Don't Audit:** By default an attempt to perform an action not permited explicitly, has a log entry, by turning this bit on we say that no log is to be created.

So in the above example we have permissions to execute the file and get the attribute, also we don't want an entry for Linking.

These three access vectors are set by the admin and when the policy match is performed these are returned.

II. Transition Decisions:

1. **Process Transition –** This can happen in two ways

   Same as Parent – lets say an **ls** is executed for **VI**

   New Domain in accordance with the policy – Example can be that during the startup of the system **init** process starts **ftpd.**

2. **File Transition –** This again can happen in the above two ways

   Same as Parent

   New Domain in accordance with the policy

:::::::::::::::::::::::::: Flask Architecture ::::::::::::::::::::::::::

SE Linux is based on the **Flask Architecture** in which the security policy is separated from the enforcement logic. The advantage of this approach is that a flexible MAC security model (typically a sort of RBAC is also integrated), as per the security need of the organization, can be implemeneted. An additional benefit of this approach is that enforcement of security policies can be transparent to the applications since it's possible to define the default security behavior. This architecture ensures the data integtrity and the trustworthiness or simply put it provides access controls.

Componets of Flask Architecture:

1. **Security Server:** The enforcement part of the SE Linux is accomplished by means of a **Security Server** which makes Security decisions based on the *Security Context*, which are a set of attributes associated with a particular object.

2. **Object Managers:** They manage security attributes, ensure appropriate security bindings for objects like files, directories, sockets and enforce the decisions made by **Security Server**.

::

How are decisions enforced in SE Linux?

Security attributes are combined to form a security context. The security context of a subject (a process) or an object (a file, socket, IPC object …) comprises a three-part colon-separated text string. Attributes are the user, the user's role, and the type, for example "system_u:object_r:inetd_exec_t".

The security server assigns a security context to each process. The security context comprises a set of rules, the parent process, and the user ID for the process. The rules must define what processes can spawn what child processes. This technique would create a "sandbox". These security contexts can be applied by the help of the Access Vectors as mentioned above, as in case of file and in case of processes it can be allow or deny fork, ptrace or signalling.

SID are now used to refer to these Security Contexts at runtime. Before the access occurs Object Manager sends the security contexts of the object and subject to the Security Server which will now deny or allow the access. This will happen for each access made, in contrast to the traditional Linux where the system checks only for the initial request and subsequent requests are based on the feedback of this result.

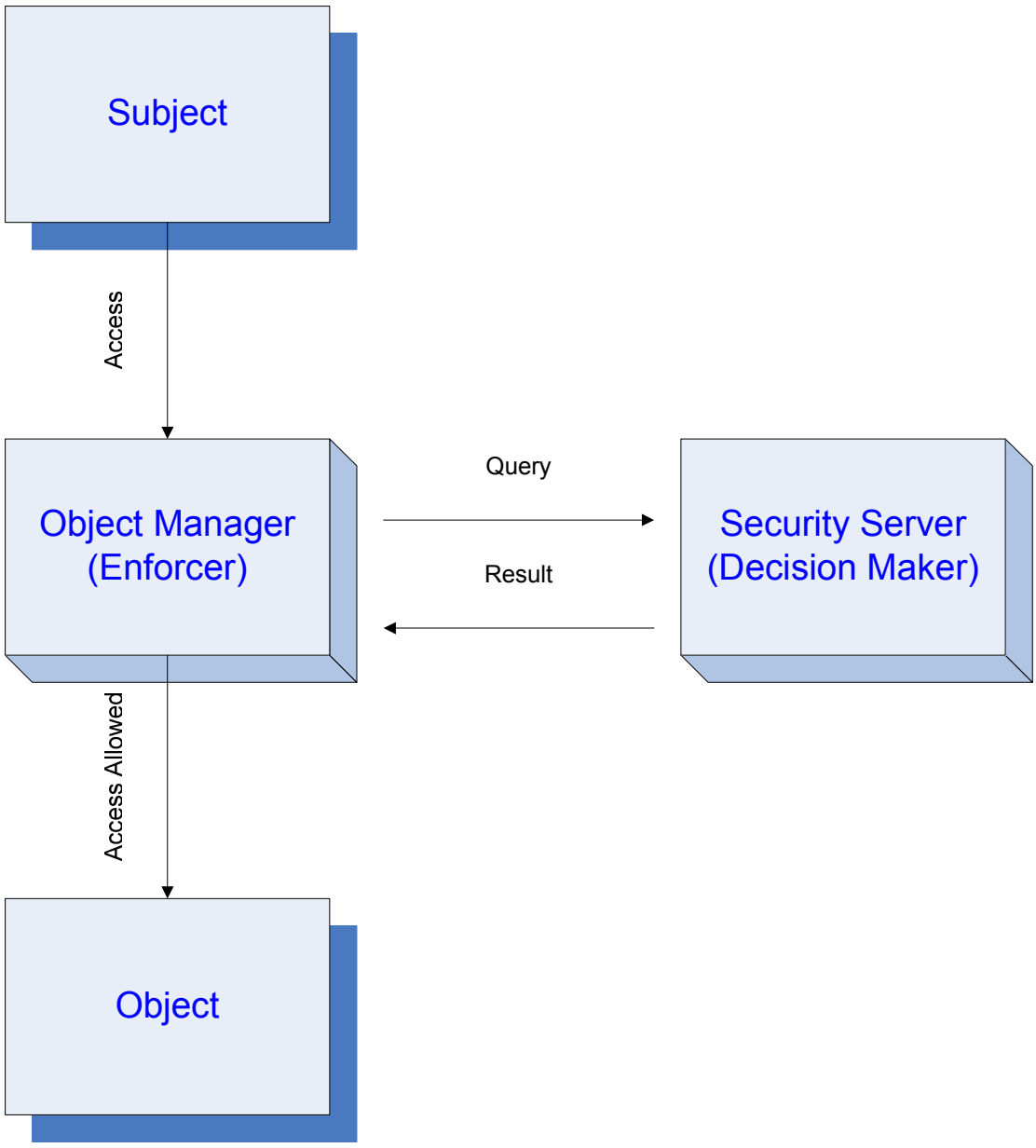

Fig 3. Decision process of SE Linux

Since the Security Server is consulted for each access, a mechanism called **Access Vector Cache (AVC)** is implemented to cache the decisions of the Security Server this will stop the performance of the system from degrading. Now if after sometime the security context of an object/ subject changes by which the access is denied Security Server will mark the entry in AVC as invalid.

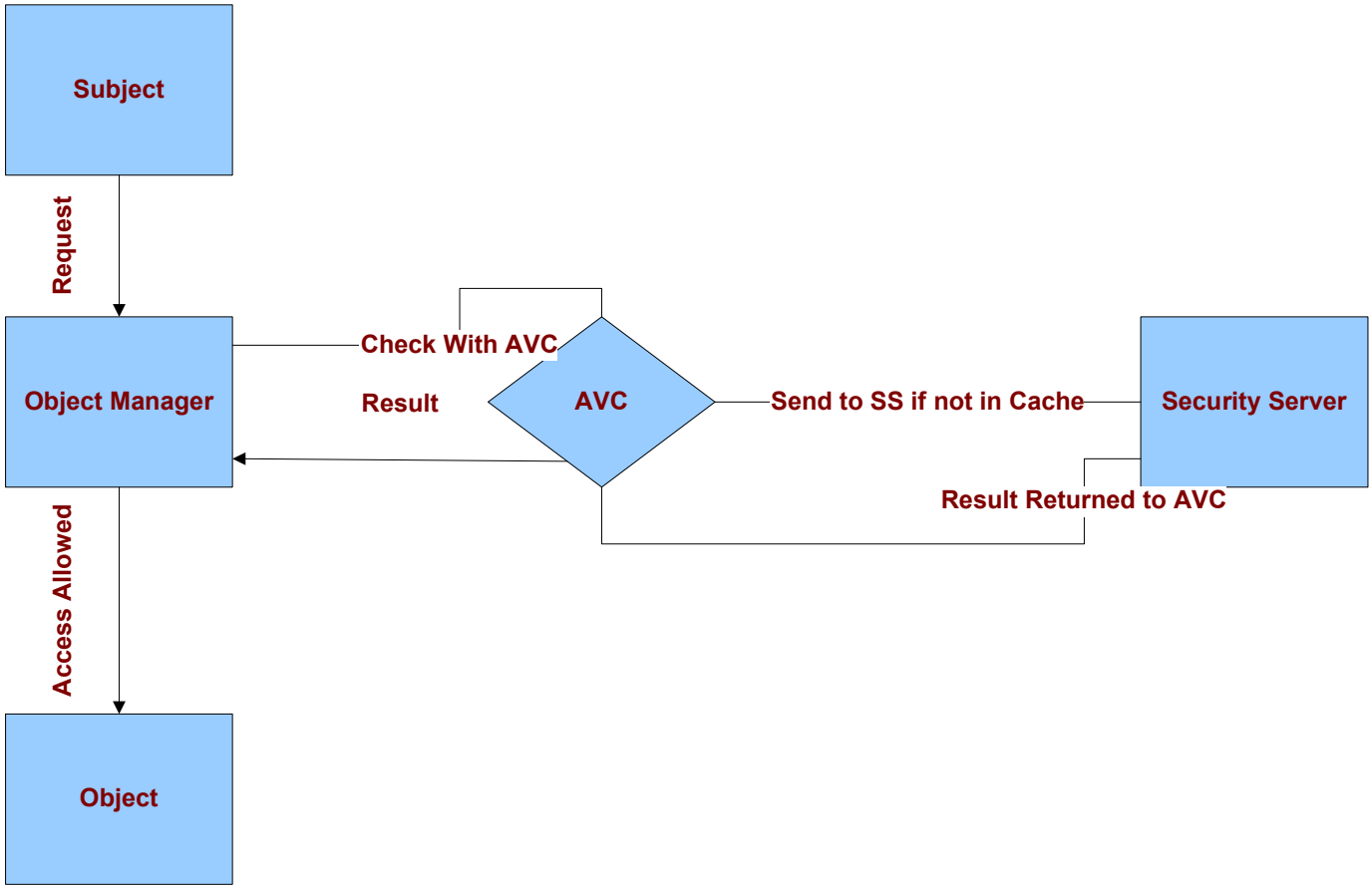

Fig 4. Decision Process with AVC.

Architecture of SE Linux

Main components are:

Kernel – Needed to make decisions and log the allowed/ denied activity.

SE Linux Shared Library (*libselinux1.so*) – Since many executables are modified by SE Linux so without support of this Library they will fail.

SE Linux Security Policy – Heart of the decision making process, discussed in details later.

Modes of Operation

SE Linux can operate in two modes, which cater different needs

Permissive Mode – Used for troubleshooting and system Administration all the activity and the operations are allowed "even the ones which violate security policy"

This mode is available only if the kernel was comiled with **NSA Linux Dev Support** most of the vendors do that.

Since this mode allows all operations, thereby make system vulnerable, care must be taken when putting the system on this mode. However even in this mode logs of the activities will be maintained.

If **GRUB** is used as a bootloader following can be done to put the system in this mode at boot time:

kernel /vmlinuz-2.6.x-x.x ro root=LABEL/ enforcing=0

Enforcing Mode – This mode is the normal mode of operation where the system enforces the policy.

It should be noted that the modes can be changed dynamically by **"root"** user if operating in **"system_r"** role.

setenforce 0

Installating SE Linux:

SE Linux can be installed in three ways:

1. As as integral part of the standard linux package e.g Fedora Core, Redhat Enterprise Linux 4. etc.

2. As a separate package like rpms

3. Getting the source from the NSA website compiling and installing.

I will not go into the details of the installation procedure as there are several good links which describe the step-by-step of doing so in much detail also the NSA website is a great reserve, there are also several mailing lists for SE Linux (see http://www.nsa.gov/selinux/info/list.cfm?MenuID=41.1.1.9) for details. I will concentrate on the working of SE Linux.

Policy implementation in the SE Linux

Actually from the a birds eye view the policy is contained in one file called *policy.conf* in */etc/security/selinux/src/policy* dir.

checkpolicy is the utility which converts the *policy.conf* (a source file) into *policy.??* (binary file which can actually be implemented)

But in reality this policy is a concatenation of several files located in the subdirectories of */etc/security/selinux/src/policy* the structure of this directory looks like:

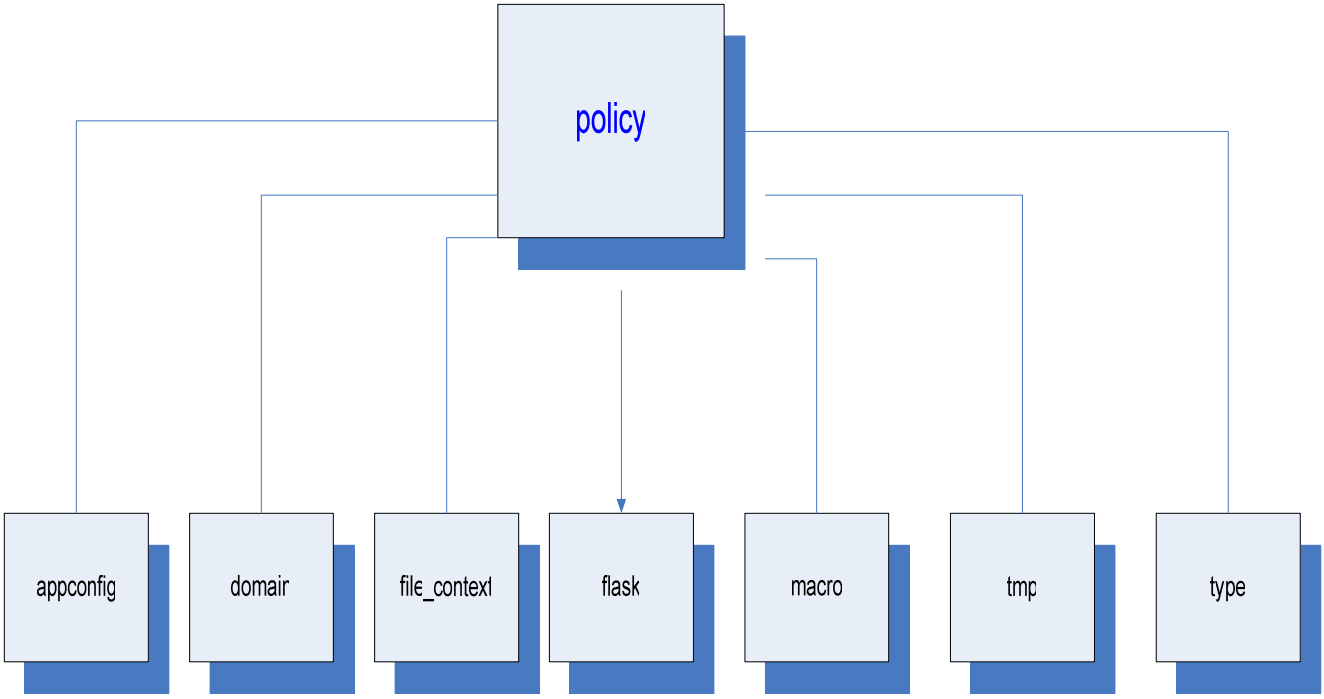

Fig 5. SE Linux Policy structure.

I will dicuss these directories and there uses later.

Type Enforcement (TE) is related to **Domains**; these domains constrain the priviledges of each of the running process and are enforced with the help of (majorly) the following files:

1. **FC** (File Context) files or .**fc** files**:** These files provide the security context of the files and directories associated with a domain.

2. **TE** (Type Enforcement) files or .**te** files: These files define the access vectors and acceptable transitions in a domain.

Each application will have these files lets say that an application "**foo**" will have ***foo.te*** and ***foo.fc***

foo.fc: This file will define the file context of the files and directories related to "**foo"** application.

e.g lets say **foo** has */usr/sbin/foo*, */etc/foo* and */usr/local/bin/foo* are the directories of this application then this fc file might contain the following lines defining the security context of these directories:

/usr/sbin/foo	system_u:object_r:foo_t
/usr/local/bin/foo	system_u:object_r:foo_t
/etc/foo(/.*)?	system_u:object_r:foo_etc_t

foo.te: Will typically have –

1. **Types :** Types associated with the object

type foo_t file, file_type,sysadmfile;

Defines a type by the name of **foo_t** attributes **file_type** and **sysadmfile** define that files associated with this type can be accessed by sysadm_r.

So the general syntax of this can be written as:

type type_name_t, attribute_1, attribute_2, attribute_3…;

These attributes are defined in **attrib.te**

2. **Allow lines:** Specifies what can the processes running in the domains, do to the associated files of the domain.

allow foo_t etc_t:file {getattr read}

Files running under domain **foo_t** can read and see the attributes of the files labeled with **etc_t.**

Policy Directory Structure in Detail:

1. **Flask** – This directory has three files

    a. **initial_sids –** used to label transient objects during the boot up
    b. **security_classes –** security classes like files and directories.
    c. **access_vectors –** defines the access vectors like read, write, create etc.

2. **file_context –** This directory has two files and two directories

    Directories:

    a. **program –** Contains the security of the files that are the part of the installed packages and programs.
    b. **misc**.

    Files:

    a. **file_context –** This file is automatically created when policy is installed.
    b. Types.fc – Security Context of the general system files like usr, home

3. **types –** defines general types like *device.te, file.te* and there usage rules.

4. **domains –** Contains two files and two directories:

    Directories:

    a. misc - Contains domains not related to a specific program e.g. startx
    b. program – Domains related to specific programs.

    Files:

    a. admin.te – used for defining sysadmin role permissions.
    b. user.te – used for defining permissions of normal users.

5. **appconfig -** Stores configuration information used by security-aware programs modified to work with SELinux, this directory has five files:

   a. **default_context – Used by programs like login, sshd to get the security context of the given user.**

   b. **default_type – Used by login to get the default role of the user.**

   c. **failsafe_context – Used by X mostly.**

   d. **initrc_context – Used by /etc/rc.d**

   e. **root_default_contexts – Used by login to define the default context of root.**

Conclusion:

SE Linux gives a granular control to the administrator however this control comes for a price of sacrificing the ease of administration and difficulty of troubleshooting. However the risks as experienced in traditional linux or unix systems is mitigated to a large extent as explained in the beginning of the paper. This provides a granular policy which can be enforced at the level of processes however the more granular the policy is more is the difficulty to mainitain it. As of the users in this scenario they are easy to administer since we have defined roles and each role has specific priviledges associated with it so the fear of priviledge escalation and abuse is mitigated. O - Day attacks are contained this is perhaps one of the most noticeable advantage of SE Linux.

Glossary:

DAC: Discretionary Access Control (the typical Linux procedure) allows users to modify access privileges to their own objects at their own discretion.DAC commonly refers to user ID based access control. Whether or not an action is permitted is decided by evaluating the user ID of the subject and the object owner.There are only two types of users: normal users and superusers.

Domain: Security attribute of a process within the TE (Type Enforcement) model.SE Linux TE does not differentiate between types and domains. However, types that refer to subjects (that is, processes) are commonly referred to as domains.

Label: Symbolic descriptor for subjects and objects that allow SE Linux to reach a decision on whether to allow or deny access.A label contains the security attributes which are applied by a central policy.In the case of SE Linux the label resides within the security context.

MAC: Mandatory Access Control refers to a policy administrator defining access privileges centrally. Users and their processes are not allowed to edit the policy, which governs all access. Many definitions assume a special form of MLS for MAC, and thus refer to generic MAC as non-discretionary access control.

MLS: Multi Level Security assigns a security level to subjects and objects, in line with layers of security for important documents: confidential, secret, top secret. Only users with sufficient security clearance are allowed access to objects.

Object: Refers to any component accessed, such as files, directories or network sockets.

Permissions: Depend on the object type. For files they could be read, write, create, rename, orexecute, for example – for processes possibly fork,ptrace,or signal.

PSID: The persistent SID is the permanent version of a security ID.A PSID is considered persistent when it survives after rebooting.It represents the binding between an object and its security context, as in the case of files, for example.

Policy: This set of rules defines who can access what, where and with what privileges.

RBAC: Role Based Access Control describes access control by means of roles.In SE Linux permissionsderive from the types and domains associated with a role.

Role: Roles simplify user management.Users are assigned roles depending on the tasks they need to perform.Permissions are assigned to users via their roles; users can be assigned to roles independently.

Security Context: A combination of user ID, role and type.To retain compatibility to other security models,the security context is a text string whose content is parsed by the security server.

SID: The security ID is a number that points to a tangible security context.This binding is applied by the SE Linux security server at runtime.

Subject: Active component in a system i.e. is a process.

TE: Type enforcement defines access by domains (subject classes) to types (classes of objects), or other domains by reference to an access matrix.SE Linux simplifies this model and also describes domains as types.The matrix defines permitted interactions between types.

Type: Security attributes of an object within the TE (type enforcement) model.

User: SE Linux user management is independent of the Linux user ID.

References:

1. www.nsa.gov
2. "Meeting Critical Security Objectives with Security-Enhanced Linux" by Peter A. Loscocco, NSA, loscocco@tycho.nsa.gov Stephen D. Smalley, NAI Labs, ssmalley@nai.com.
3. "The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments " by Peter A. Loscocco, Stephen D. Smalley, Patrick A. Muckelbauer, Ruth C. Taylor,S. Jeff Turner, John F. Farrelltos@epoch.ncsc.mil , National Security Agency .
4. SE Linux By Bill McCarty ISBN 0-596-00716-7 Publisher Oreilly.
5. www.linux-magazine.com article on SE Linux by Carsten Grohmann and Konstantin Agouros.
6. "A Security Policy Configuration for the Security-Enhanced Linux" by Stephen Smalley, NAI Labs, sds@tislabs.com, Timothy Fraser, NAI Labs, tfraser@tislabs.com.
7. "Configuring the SELinux Policy" by Stephen Smalley