INTRODUCTION TO SELINUX

Security-Enhanced Linux (SELinux) is a security architecture integrated into the 2.6.x kernel using the *Linux Security Modules* (LSM). It is a project of the United States National Security Agency (NSA) and the SELinux community. SELinux integration into Red Hat Enterprise Linux was a joint effort between the NSA and Red Hat.

49.2.1. SELinux Overview

SELinux provides a flexible *Mandatory Access Control* (MAC) system built into the Linux kernel. Under standard Linux *Discretionary Access Control* (DAC), an application or process running as a user (UID or SUID) has the user's permissions to objects such as files, sockets, and other processes. Running a MAC kernel protects the system from malicious or flawed applications that can damage or destroy the system.

SELinux defines the access and transition rights of every user, application, process, and file on the system. SELinux then governs the interactions of these entities using a security policy that specifies how strict or lenient a given Red Hat Enterprise Linux installation should be.

On a day-to-day basis, system users will be largely unaware of SELinux. Only system administrators need to consider how strict a policy to implement for their server environment. The policy can be as strict or as lenient as needed, and is very finely detailed. This detail gives the SELinux kernel complete, granular control over the entire system.

The SELinux Decision Making Process

When a subject, (for example, an application), attempts to access an object (for example, a file), the policy enforcement server in the kernel checks an *access vector cache* (AVC), where subject and object permissions are cached. If a decision cannot be made based on data in the AVC, the request continues to the security server, which looks up the *security context* of the application and the file in a matrix. Permission is then granted or denied, with an `avc: denied` message detailed in `/var/log/messages` if permission is denied. The security context of subjects and objects is applied from the

installed policy, which also provides the information to populate the security server's matrix.

Refer to the following diagram:

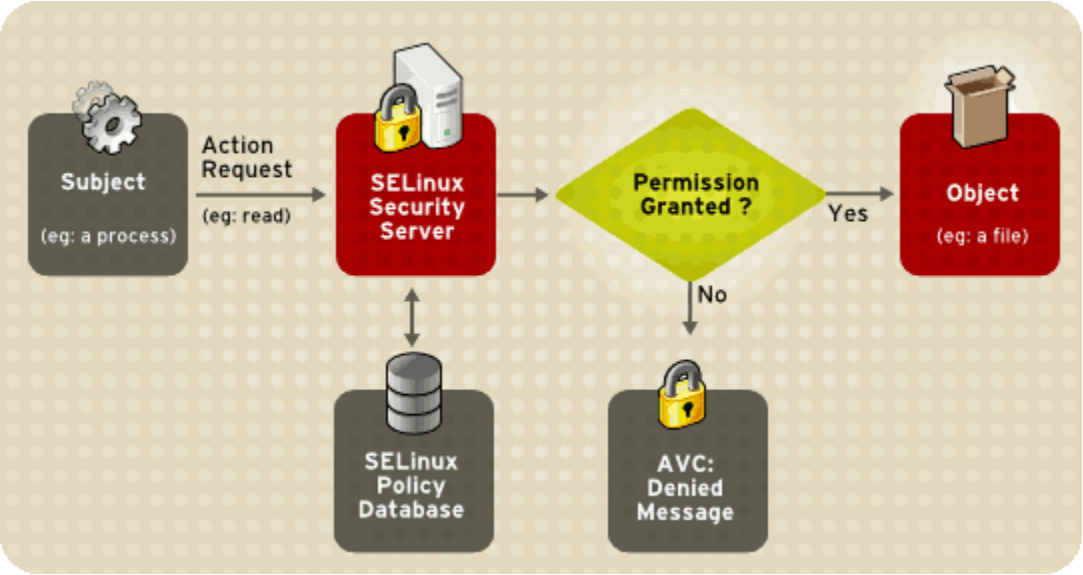


Figure 49.1. SELinux Decision Process

SELinux Operating Modes

Instead of running in *enforcing* mode, SELinux can run in *permissive* mode, where the AVC is checked and denials are logged, but SELinux does not enforce the policy. This can be useful for troubleshooting and for developing or fine-tuning SELinux policy.

Note

The `/etc/sysconfig/selinux` contains a symbolic link to the actual configuration file, `/etc/selinux/config`.

The following explains the full subset of options available for configuration:

- `SELINUX=`*`enforcing`*`|`*`permissive`*`|`*`disabled`* — Defines the top-level state of SELinux on a system.
  - `enforcing` — The SELinux security policy is enforced.
  - `permissive` — The SELinux system prints warnings but does not enforce policy.
    This is useful for debugging and troubleshooting purposes. In permissive mode, more denials are logged because subjects can continue with actions that would otherwise be denied in enforcing mode. For example, traversing a directory tree in permissive mode produces `avc: denied` messages for every directory level read. In enforcing mode, SELinux would have stopped the initial traversal and kept further denial messages from occurring.
  - `disabled` — SELinux is fully disabled. SELinux hooks are disengaged from the kernel and the pseudo-file system is unregistered.

49.2.2.2.2. The `/etc/selinux/` Directory

The `/etc/selinux/` directory is the primary location for all policy files as well as the main configuration file.

The following example shows sample contents of the `/etc/selinux/` directory:

```
-rw-r--r--  1 root root  448 Sep 22 17:34 config
drwxr-xr-x  5 root root 4096 Sep 22 17:27 strict
drwxr-xr-x  5 root root 4096 Sep 22 17:28 targeted
```

The two subdirectories, `strict/` and `targeted/`, are the specific directories where the policy files of the same name (that is, `strict` and `targeted`) are contained.

49.2.2.3. SELinux Utilities

The following are some of the commonly used SELinux utilities:

- `/usr/sbin/setenforce` — Modifies in real-time the mode in which SELinux runs.
  For example:
  `setenforce 1` — SELinux runs in enforcing mode.
  `setenforce 0` — SELinux runs in permissive mode.

`/usr/sbin/sestatus -v` — Displays the detailed status of a system running SELinux. The following example shows an excerpt of `sestatus -v` output:

```
SELinux status:                 enabled

SELinuxfs mount:                /selinux
Current mode:                   enforcing
Mode from config file:          enforcing
Policy version:                 21
Policy from config file:        targeted

Process contexts:
Current context:                user_u:system_r:unconfined_t:s0
Init context:                   system_u:system_r:init_t:s0
```

- `/sbin/mingetty                  system_u:system_r:getty_t:s0`

49.3. BRIEF BACKGROUND AND HISTORY OF SELINUX

SELinux was originally a development project from the National Security Agency (*NSA*)[18]and others. It is an implementation of the *Flask* operating system security architecture.[19]The NSA integrated SELinux into the Linux kernel using the *Linux Security Modules*(*LSM*) framework. SELinux motivated the creation of LSM, at the suggestion of Linus Torvalds, who wanted a modular approach to security instead of just accepting SELinux into the kernel.

Originally, the SELinux implementation used *persistent security IDs* (PSIDs) stored in an unused field of the ext2 inode. These numerical representations (i.e., non-human-readable) were mapped by SELinux to a security context label. Unfortunately, this required modifying each file system type to support PSIDs, so was not a scalable solution or one that would be supported upstream in the Linux kernel.

The next evolution of SELinux was as a loadable kernel module for the 2.4.<*x*> series of Linux kernels. This module stored PSIDs in a normal file, and SELinux was able to support more file systems. This solution was not optimal for performance, and was inconsistent across platforms. Finally, the SELinux code was integrated upstream to the 2.6.*x* kernel, which has full support for LSM and has *extended attributes* (*xattrs*) in the ext3 file system. SELinux was moved to using xattrs to store security context information. The xattrnamespace provides useful separation for multiple security modules existing on the same system.

Much of the work to get the kernel ready for upstream, as well as subsequent SELinux development, has been a joint effort between the NSA, Red Hat, and the community of SELinux developers.

For more information about the history of SELinux, the definitive website is http://www.nsa.gov/research/selinux/index.shtml.

Why SELinux

Before we begin, let's understand a few concepts.

SELinux implements what's known as MAC (Mandatory Access Control). This is implemented on top of what's already present in every Linux distribution, the DAC (Discretionary Access Control).

To understand DAC, let's first consider how traditional Linux file security works.

In a traditional security model, we have three entities: User, Group, and Other (u,g,o) who can have a combination of Read, Write, and Execute (r,w,x) permissions on a file or directory. If a user jo creates a file in their home directory, that user will have read/write access to it, and so will the jo group. The "other" entity will possibly have no access to it. In the following code block, we can consider the hypothetical contents of jo's home directory.

You don't need to set up this jo user - we'll be setting up plenty of users later in the tutorial.

Running a command like this:

```
ls -l /home/jo/
```

can show output like the following:

```
total 4
-rwxrw-r--. 1 jo jo 41 Aug  6 22:45 myscript.sh
```

Now jo can change this access. jo can grant (and restrict) access to this file to other users and groups or change the owner of the file. These actions can leave critical files exposed to accounts who don't need this access. jo can also restrict to be more secure, but that's discretionary: there's no way for the system administrator to enforce it for every single file in the system.

Consider another case: when a Linux process runs, it may run as the root user or another account with superuser privileges. That means if a black-hat hacker takes control of the application, they can use that application to get access to whatever resource the user account has access to. For processes running as the root user, basically this means everything in the Linux server.

Think about a scenario where you want to restrict users from executing shell scripts from their home directories. This can happen when you have developers working on a

production system. You would like them to view log files, but you don't want them to use su or sudo commands, and you don't want them to run any scripts from their home directories. How do you do that?

SELinux is a way to fine-tune such access control requirements. With SELinux, you can define what a user or process can do. It confines every process to its own domain so the process can interact with only certain types of files and other processes from allowed domains. This prevents a hacker from hijacking any process to gain system-wide access.

Installing Apache and SFTP Services

First, let's log in to the server as the root user and run the following command to install Apache:

```
yum install httpd
```

The output will show the package being downloaded and ask you for permission to install:

```
Loaded plugins: fastestmirror, langpacks
...
...
================================================================================
 Package      Arch         Version                 Repository      Size
================================================================================
Installing:
 httpd        x86_64       2.4.6-18.el7.centos     updates         2.7 M

Transaction Summary
================================================================================
Install  1 Package

Total download size: 2.7 M
Installed size: 9.3 M
Is this ok [y/d/N]:
```

Pressing y will install the Apache web server daemon.

```
Downloading packages:
httpd-2.4.6-18.el7.centos.x86_64.rpm                    | 2.7 MB   00:01
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  Installing : httpd-2.4.6-18.el7.centos.x86_64                    1/1
  Verifying  : httpd-2.4.6-18.el7.centos.x86_64                    1/1

Installed:
  httpd.x86_64 0:2.4.6-18.el7.centos

Complete!
```

Start the daemon manually:

```
service httpd start
```

Running the service httpd status command will show the service is now running:

```
Redirecting to /bin/systemctl status  httpd.service
httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled)
   Active: active (running) since Tue 2014-08-19 13:39:48 EST; 1min 40s ago
 Main PID: 339 (httpd)
...
...
```

Next we will install vsftp:

```
yum install vsftpd
```

The output should look similar to the following:

```
Loaded plugins: fastestmirror, langpacks
...
...
```

```
================================================================================
================================
 Package             Arch              Version                 Repository        Size
================================================================================
================================
Installing:
 vsftpd              x86_64            3.0.2-9.el7             base              165 k

Transaction Summary
================================================================================
================================
Install  1 Package

Total download size: 165 k
Installed size: 343 k
Is this ok [y/d/N]:
```

Press y to install the package.

Next, we will use the service vsftpd start command to start the vsftpd daemon. The output should show something like the following:

```
Redirecting to /bin/systemctl status  vsftpd.service
vsftpd.service - Vsftpd ftp daemon
   Loaded: loaded (/usr/lib/systemd/system/vsftpd.service; disabled)
   Active: active (running) since Tue 2014-08-19 13:48:57 EST; 4s ago
  Process: 599 ExecStart=/usr/sbin/vsftpd /etc/vsftpd/vsftpd.conf (code=exited, status=0/SUCCESS)
 Main PID: 600 (vsftpd)
...
...
```

Installing SELinux Packages

A number of packages are used in SELinux. Some are installed by default. Here is a list for Red Hat-based distributions:

- *policycoreutils* (provides utilities for managing SELinux)
- *policycoreutils-python* (provides utilities for managing SELinux)
- *selinux-policy* (provides SELinux reference policy)
- *selinux-policy-targeted* (provides SELinux targeted policy)

- *libselinux-utils* (provides some tools for managing SELinux)
- *setroubleshoot-server* (provides tools for deciphering audit log messages)
- *setools* (provides tools for audit log monitoring, querying policy, and file context management)
- *setools-console* (provides tools for audit log monitoring, querying policy, and file context management)
- *mcstrans* (tools to translate different levels to easy-to-understand format)

Some of these are installed already. To check what SELinux packages are installed on your CentOS 7 system, you can run a few commands like the one below (with different search terms after grep) as the root user:

```
rpm -qa | grep selinux
```

The output should look something like this:

```
libselinux-utils-2.2.2-6.el7.x86_64
libselinux-2.2.2-6.el7.x86_64
selinux-policy-targeted-3.12.1-153.el7.noarch
selinux-policy-3.12.1-153.el7.noarch
libselinux-python-2.2.2-6.el7.x86_64
```

SELinux Modes

It's time to start playing around with SELinux, so let's begin with SELinux modes. At any one time, SELinux can be in any of three possible modes:

- Enforcing
- Permissive
- Disabled

In enforcing mode SELinux will *enforce* its policy on the Linux system and make sure any unauthorized access attempts by users and processes are denied. The access denials are also written to relevant log files. We will talk about SELinux policies and audit logs later.

Permissive mode is like a semi-enabled state. SELinux doesn't apply its policy in permissive mode, so no access is denied. However any policy violation is still logged in the audit logs. It's a great way to test SELinux before enforcing it.

The disabled mode is self-explanatory – the system won't be running with enhanced security.

Checking SELinux Modes and Status

We can run the getenforce command to check the current SELinux mode.

```
getenforce
```

SELinux should currently be disabled, so the output will look like this:

```
Disabled
```

We can also run the sestatus command:

```
sestatus
```

When SELinux is disabled the output will show:

```
SELinux status:         disabled
```

SELinux Configuration File

The main configuration file for SELinux is /etc/selinux/config. We can run the following command to view its contents:

```
cat /etc/selinux/config
```

The output will look something like this:

```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#     enforcing - SELinux security policy is enforced.
#     permissive - SELinux prints warnings instead of enforcing.
#     disabled - No SELinux policy is loaded.
SELINUX=disabled
# SELINUXTYPE= can take one of these two values:
#     targeted - Targeted processes are protected,
#     minimum - Modification of targeted policy. Only selected processes are protected.
#     mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

There are two directives in this file. The SELINUX directive dictates the SELinux mode and it can have three possible values as we discussed before.

SELinux Policy

At the heart of SELinux' security engine is its *policy*. A policy is what the name implies: a set of rules that define the security and access rights for everything in the system. And when we say *everything*, we mean users, roles, processes, and files. The policy defines how each of these entities are related to one another.

Some Basic Terminology

To understand policy, we have to learn some basic terminology. We will go into the details later, but here is a brief introduction. An SELinux policy defines user access to roles, role access to domains, and domain access to types.

Users

SELinux has a set of pre-built users. Every regular Linux user account is mapped to one or more SELinux users.

In Linux, a user runs a process. This can be as simple as the user jo opening a document in the vi editor (it will be jo's account running the vi process) or a service account running the httpd daemon. In the SELinux world, a process (a daemon or a running program) is called a *subject*.

Roles

A *role* is like a gateway that sits between a user and a process. A role defines which users can access that process. Roles are not like groups, but more like filters: a user may enter or assume a role at any time provided the role grants it. The definition of a role in SELinux policy defines which users have access to that role. It also defines what process domains the role itself has access to. Roles come into play because part of SELinux implements what's known as Role Based Access Control (RBAC).

Subjects and Objects

A *subject* is a process and can potentially affect an *object*.

An *object* in SELinux is anything that can be acted upon. This can be a file, a directory, a port, a tcp socket, the cursor, or perhaps an X server. The actions that a subject can perform on an object are the subject's *permissions*.

Domains are for Subjects

A *domain* is the context within which an SELinux subject (process) can run. That context is like a wrapper around the subject. It tells the process what it can and can't do. For example, the domain will define what files, directories, links, devices, or ports are accessible to the subject.

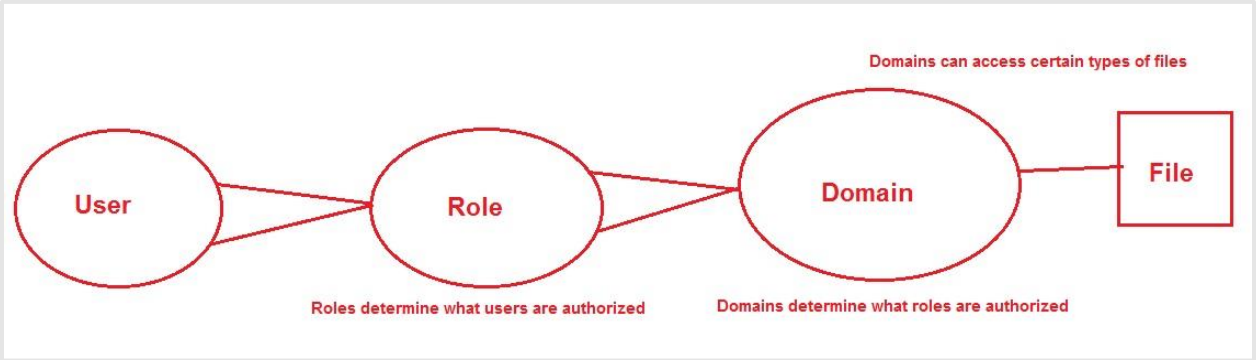
Types are for Objects

A *type* is the context for a file's context that stipulates the file's purpose. For example, the context of a file may dictate that it's a web page, or that the file belongs to the /etc directory, or that the file's owner is a specific SELinux user. A file's context is called its *type* in SELinux lingo.

So what is SELinux policy?

SELinux policy defines user access to roles, role access to domains, and domain access to types. First the user has to be authorized to enter a role, and then the role has to be authorized to access the domain. The domain in turn is restricted to access only certain types of files.

The policy itself is a bunch of rules that say that so-and-so users can assume only so-and-so roles, and those roles will be authorized to access only so-and-so domains. The domains in turn can access only so-and-so file types. The following image shows the concept:



Terminology tip: The last bit, where a process running within a particular domain can perform only certain operations on certain types of objects, is called *Type Enforcement* (TE).

Coming back to the topic of policies, SELinux policy implementations are also typically *targeted* by default. If you remember the SELinux config file that we saw before, the SELINUXTYPE directive is set to be targeted. What this means is that, by default,

SELinux will restrict only certain processes in the system (i.e. only certain processes are targeted). The ones that are not targeted will run in unconfined domains.

first line of defense has already blocked access. SELinux security decisions come into play *after* DAC security has been evaluated.

Conclusion

In the first part of this tutorial we have tried to understand a few basic concepts around SELinux. We have seen how SELinux can secure a system, how we can enable it and what modes it can be running in. We have also touched on the topic of SELinux policy. Next, we will learn how to use SELinux to restrict access to files and processes.

IPTABLES

Packet Filtering

The Linux kernel uses the **Netfilter** facility to filter packets, allowing some of them to be received by or pass through the system while stopping others. This facility is built in to the Linux kernel, and has three built-in *tables* or *rules lists*, as follows:

- `filter` — The default table for handling network packets.
- `nat` — Used to alter packets that create a new connection and used for *Network Address Translation* (*NAT*).
- `mangle` — Used for specific types of packet alteration.

Each table has a group of built-in *chains*, which correspond to the actions performed on the packet by `netfilter`.

The built-in chains for the `filter` table are as follows:

- *INPUT* — Applies to network packets that are targeted for the host.
- *OUTPUT* — Applies to locally-generated network packets.
- *FORWARD* — Applies to network packets routed through the host.

The built-in chains for the `nat` table are as follows:

- *PREROUTING* — Alters network packets when they arrive.
- *OUTPUT* — Alters locally-generated network packets before they are sent out.
- *POSTROUTING* — Alters network packets before they are sent out.

The built-in chains for the `mangle` table are as follows:

- *INPUT* — Alters network packets targeted for the host.
- *OUTPUT* — Alters locally-generated network packets before they are sent out.
- *FORWARD* — Alters network packets routed through the host.
- *PREROUTING* — Alters incoming network packets before they are routed.
- *POSTROUTING* — Alters network packets before they are sent out.

Every network packet received by or sent from a Linux system is subject to at least one table. However, a packet may be subjected to multiple rules within each table before emerging at the end of the chain. The structure and purpose of these rules may vary, but they usually seek to identify a packet coming from or going to a particular IP address, or set of addresses, when using a particular protocol and network service.

Note

By default, firewall rules are saved in the `/etc/sysconfig/iptables` or `/etc/sysconfig/ip6tables` files.

The `iptables` service starts before any DNS-related services when a Linux system is booted. This means that firewall rules can only reference numeric IP addresses (for example, 192.168.0.1). Domain names (for example, host.example.com) in such rules produce errors.

Regardless of their destination, when packets match a particular rule in one of the tables, a *target* or action is applied to them. If the rule specifies an `ACCEPT` target for a matching packet, the packet skips the rest of the rule checks and is allowed to continue to its destination. If a rule specifies a `DROP` target, that packet is refused access to the system and nothing is sent back to the host that sent the packet. If a rule specifies a `QUEUE`target, the packet is passed to user-space. If a rule specifies the optional `REJECT` target, the packet is dropped, but an error packet is sent to the packet's originator.

Every chain has a default policy to `ACCEPT`, `DROP`, `REJECT`, or `QUEUE`. If none of the rules in the chain apply to the packet, then the packet is dealt with in accordance with the default policy.

The `iptables` command configures these tables, as well as sets up new tables if necessary.

Network interfaces must be associated with the correct chains in firewall rules.

For example, incoming interfaces (`-i` option) can only be used in INPUT or FORWARD chains. Similarly, outgoing interfaces (`-o` option) can only be used in FORWARD or OUTPUT chains.

In other words, INPUT chains and incoming interfaces work together; OUTPUT chains and outgoing interfaces work together. FORWARD chains work with both incoming and outgoing interfaces.

OUTPUT chains are no longer used by incoming interfaces, and INPUT chains are not seen by packets moving through outgoing interfaces.

Command Options for IPTables

Rules for filtering packets are created using the `iptables` command. The following aspects of the packet are most often used as criteria:

- *Packet Type* — Specifies the type of packets the command filters.

- *Packet Source/Destination* — Specifies which packets the command filters based on the source or destination of the packet.
- *Target* — Specifies what action is taken on packets matching the above criteria.

Structure of IPTables Command Options

Many `iptables` commands have the following structure:

```
iptables [-t <table-name>] <command> <chain-name> \
                  <parameter-1> <option-1> \
                  <parameter-n> <option-n>
```

<*table-name*> — Specifies which table the rule applies to. If omitted, the `filter` table is used.

<*command*> — Specifies the action to perform, such as appending or deleting a rule.

<*chain-name*> — Specifies the chain to edit, create, or delete.

<*parameter*>-<*option*> pairs — Parameters and associated options that specify how to process a packet that matches the rule.

The length and complexity of an `iptables` command can change significantly, based on its purpose.

For example, a command to remove a rule from a chain can be very short:

```
iptables -D <chain-name> <line-number>
```

ICMP Protocol

The following match options are available for the Internet Control Message Protocol (ICMP) (`-p icmp`):

- `--icmp-type` — Sets the name or number of the ICMP type to match with the rule. A list of valid ICMP names can be retrieved by typing the `iptables -p icmp -h` command.