

# 17

## CHAPTER

# Security-Enhanced Linux

**T**hough numerous security tools exist for protecting specific services, as well as user information and data, no tool has been available for protecting the entire system at the administrative level. Security-Enhanced Linux is a project to provide built-in administrative protection for aspects of your Linux system. Instead of relying on users to protect their files or on a specific network program to control access, security measures are built into the basic file management system and the network access methods. All controls can be managed directly by an administrator as part of Linux system administration.

Security-Enhanced Linux (SELinux) is a project developed and maintained by the National Security Agency (NSA), which chose Linux as its platform for implementing a secure operating system. Most Linux distributions have embraced SELinux and incorporated it as a standard feature of its distribution. Detailed documentation is available from resources listed in Table 17-1, including sites provided by the NSA and SourceForge. Also check your Linux distribution's site for any manuals, FAQs, or documentation on SELinux.

Linux and Unix systems normally use a discretionary access control (DAC) method for restricting access. In this approach, users and the objects they own, such as files, determine permissions. The user has complete discretion over the objects he or she owns. The weak point in many Linux/Unix systems has been the user administrative accounts. If an attacker managed to gain access to an administrative account, they would have complete control over the service the account managed. Access to the root user would give control over the entire system, all its users, and any network services it was running. To counter this weakness, the NSA set up a mandatory access control (MAC) structure. Instead of an all-or-nothing set of privileges based on accounts, services and administrative tasks are compartmentalized and separately controlled with policies detailing what can and cannot be done. Access is granted not just because one is an authenticated user, but when specific security criteria are met. Users, applications, processes, files, and devices can be given only the access they need to do their job, and nothing more.

## Flask Architecture

The Flask architecture organizes operating system components and data into subjects and objects. Subjects are processes: applications, drivers, system tasks that are currently running. Objects are fixed components such as files, directories, sockets, network interfaces, and devices. For each subject and object, a security context is defined. A *security context* is a set of

Resource	Location
NSA SELinux	<a href="http://nsa.gov/selinux">nsa.gov/selinux</a>
NSA SELinux FAQ	<a href="http://nsa.gov/selinux/info/faq.cfm">nsa.gov/selinux/info/faq.cfm</a>
SELinux at sourceforge.net	<a href="http://selinux.sourceforge.net">selinux.sourceforge.net</a>
Writing SELinux Policy HOWTO	Accessible from "SELinux resources at sourceforge" link at <a href="http://selinux.sourceforge.net">selinux.sourceforge.net</a>
NSA SELinux Documentation	<a href="http://nsa.gov/selinux/info/docs.cfm">nsa.gov/selinux/info/docs.cfm</a>
Configuring SELinux Policy	Accessible from NSA SELinux Documentation
SELinux Reference Policy Project	<a href="http://oss.tresys.com/projects/refpolicy">http://oss.tresys.com/projects/refpolicy</a>

**TABLE 17-1** SELinux Resources

security attributes that determines how a subject or object can be used. This approach provides very fine-grained control over every element in the operating system as well as all data on your computer.

The attributes designated for the security contexts and the degree to which they are enforced are determined by an overall security policy. The policies are enforced by a security server. Distributions may provide different preconfigured policies from which to work. For example, Fedora provides three policies, each in its own package: strict, targeted, and mls, all a variation of a single reference policy.

SELinux uses a combination of the Type Enforcement (TE), Role Based Access Control (RBAC), and Multi-Level Security (MLS) security models. Type Enforcement focuses on objects and processes like directories and applications, whereas Role Based Access Enforcement controls user access. For the Type Enforcement model, the security attributes assigned to an object are known as either domains or types. Types are used for fixed objects such as files, and domains are used for processes such as running applications. For user access to processes and objects, SELinux makes use of the Role Based Access Control model. When new processes or objects are created, transition rules specify the type or domain they belong to in their security contexts.

With the RBAC model, users are assigned roles for which permissions are defined. The roles restrict what objects and processes a user can access. The security context for processes will include a role attribute, controlling what objects it can assess. The new Multi-Level Security (MLS) adds a security level, containing both a sensitivity and capability value.

Users are given separate SELinux user identities. Normally these correspond to the user IDs set up under the standard Linux user creation operations. Though they may have the same name, they are not the same identifiers. Standard Linux identities can be easily changed with commands like `setuid` and `su`. Changes to the Linux user ID will not affect the SELinux ID. This means that even if a user changes his or her ID, SELinux will still be able to track it, maintaining control over that user.

---

## System Administration Access

It is critically important that you make sure you have system administrative access under SELinux before you enforce its policies. This is especially true if you are using a strict or mls policy, which imposes restrictions on administrative access. You should always use SELinux

in permissive mode first and check for any messages denying access. With SELinux enforced, it may no longer matter whether you can access the root user. What matters is whether your user, even the root user, has `sysadm_r` role and `sysadm_t` object access and an administrative security level. You may not be able to use the `su` command to access the root user and expect to have root user administrative access. Recall that SELinux keeps its own security identities that are not the same as Linux user IDs. Though you might change your user ID with `su`, you still have not changed your security ID.

The targeted policy will set up rules that allow standard system administrator access using normal Linux procedures. The root user will be able to access the root user account normally. In the strict policy, however, the root user needs to access its account using the appropriate security ID. Both are now part of a single reference policy. If you want administrative access through the `su` command (from another user), you first use the `su` command to log in as the root user. You then have to change your role to that of the `sysadm_r` role, and you must already be configured by SELinux policy rules to be allowed to take on the `sysadm_r` role. A user can have several allowed possible roles he or she can assume.

To change the role, you use the `newrole` command with the `-r` option.

```
newrole -r sysadm_r
```

---

## Terminology

SELinux uses several terms that have different meanings in other contexts. The terminology can be confusing because some of the terms, such as *domain*, have different meanings in other, related, areas. For example, a domain in SELinux is a process as opposed to an object, whereas in networking the term refers to network DNS addresses.

## Identity

SELinux creates identities with which to control access. Identities are not the same as traditional user IDs. At the same time, each user normally has an SELinux identity, though the two are not linked. Affecting a user does not affect the corresponding SELinux identity. SELinux can set up a separate corresponding identity for each user, though on the less secure policies, such as targeted policies, general identities are used. A general user identity is used for all normal users, restricting users to user-level access, whereas administrators are given administrative identities. You can further define security identities for particular users.

The identity makes up part of a security context that determines what a user can or cannot do. Should a user change user IDs, that user's security identity will not change. A user will always have the same security identity. In traditional Linux systems, a user can use commands like `su` to change his or her user ID, becoming a different user. On SELinux, even though a user can still change his or her Linux user ID, the user still retains the same original security ID. You always know what a particular person is doing on your system, no matter what user ID that person may assume.

The security identity can have limited access. So, even though a user may use the Linux `su` command to become the root user, the user's security identity could prevent him or her from performing any root user administrative commands. As noted previously, to gain an administrative access, the role for their security identity would have to change as well.

## 330 Part V: Security

Use `id -z` to see what the security context for your security identity is, what roles you have, and what kind of objects you can access. This will list the user security context that starts with the security ID, followed by a colon, and then the roles a user has and the objects the user can control. Security identities can have roles that control what they can do. A user role is `user_r`, and a system administration role is `system_r`. The general security identity is `user_u`, whereas a particular security identity will normally use the username. The following example shows a standard user with the general security identity:

```
$ id -z
user_u:user_r:user_t
```

In this example the user has a security identity called `george`:

```
$ id -z
george:user_r:user_t
```

You can use the `newrole` command to change the role a user is allowed. Changing to a system administrative role, the user can then have equivalent root access.

```
$ id -z
george:sysadm_r:sysadm_t
```

### Domains

*Domains* are used to identify and control processes. Each process is assigned a domain within which it can run. A domain sets restrictions on what a process can do. Traditionally, a process was given a user ID to determine what it could do, and many had to have root user ID to gain access to the full file system. This also could be used to gain full administrative access over the entire system. A domain, on the other hand, can be tailored to access some areas but not others. Attempts to break into another domain, such as the administrative domain, would be blocked. For example, the administrative domain is `sysadm_t`, whereas the DNS server uses only `named_t`, and users have a `user_t` domain.

### Types

Whereas domains control processes, *types* control objects like files and directories. Files and directories are grouped into types that can be used to control who can have access to them. The type names have the same format as the domain names, ending with a `_t` suffix. Unlike domains, types reference objects, including files, devices, and network interfaces.

### Roles

Types and domains are assigned to roles. Users (security identities) with a given role can access types and domains assigned to that role. For example, most users can access `user_t` type objects but not `sysadm_t` objects. The types and domains a user can access are set by the role entry in configuration files. The following example allows users to access objects with the user password type:

```
role user_r types user_passwd_t
```

## Security Context

Each object has a security context that set its security attributes. These include identity, role, domain or type. A file will have a security context listing the kind of identity that can access it, the role under which it can be accessed, and the security type it belongs to. Each component adds its own refined level of security. Passive objects are usually assigned a generic role, `object_r`, which has no effect, as such objects cannot initiate actions.

A normal file created by users in their own directories will have the following identity, role, and type. The identity is a user and the role is that of an object. The type is the user's home directory. This type is used for all subdirectories and their files created within a user's home directory.

```
user_u:object_r:user_home_t
```

A file or directory created by that same user in a different part of the file system will have a different type. For example, the type for files created in the `/tmp` directory will be `tmp_t`.

```
user_u:object_r:tmp_t
```

## Transition: Labeling

A *transition*, also known as labeling, assigns a security context to a process or file. For a file, the security context is assigned when it is created, whereas for a process the security context is determined when the process is run.

Making sure every file has an appropriate security context is called *labeling*. Adding another file system requires that you label (add security contexts) to the directories and files on it. Labeling varies, depending on the policy you use. Each policy may have different security contexts for objects and processes. Relabeling is carried out using the `fixfiles` command in the policy source directory.

```
fixfiles relabel
```

## Policies

A *policy* is a set of rules to determine the relationships between users, roles, and types or domains. These rules state what types a role can access and what roles a user can have.

---

## Multi-Level Security (MLS) and Multi-Category Security (MCS)

Multi-Level Security (MLS) adds a more refined security access method, designed for servers. MLS adds a security level value to resources. Only users with access to certain levels can access the corresponding files and applications. Within each level, access can be further controlled with the use of categories. Categories work much like groups, allowing access only to users cleared for that category. Access becomes more refined, instead of an all-or-nothing situation.

Multi-Category Security (MCS) extends SELinux to use not only by administrators, but also by users. Users can set categories that can restrict and control access to their files and applications. Though based on MLS, it uses only categories, not security levels. Users can select a category for a file, but only the administrator can create a category and determine

## 332 Part V: Security

what users can access it. Though similar in concept to an ACL (Access Control List), it differs in that it makes use of the SELinux security structure, providing user-level control enforced by SELinux.

---

### Management Operations for SELinux

Certain basic operations, such as checking the SELinux status, checking a user's or file's security context, or disabling SELinux at boot, can be very useful.

#### Turning Off SELinux

Should you want to turn off SELinux before you even start up your system, you can do so at the boot prompt. Just add the following parameter to the end of your GRUB boot line:

```
selinux=0
```

To turn off SELinux permanently, set the **SELINUX** variable in the `/etc/selinux/config` file to **disabled**:

```
SELINUX=disabled
```

To turn off (permissive mode) SELinux temporarily without rebooting, use the **setenforce** command with the **0** option; use **1** to turn it back on (enforcing mode). You can also use the terms **permissive** or **enforcing** at the arguments instead of **0** or **1**. You must first have the `sysadm_r` role, which you can obtain by logging in as the root user.

```
setenforce 1
```

#### Checking Status and Statistics

To check the current status of your SELinux system, use **sestatus**. Adding the **-v** option will also display process and file contexts, as listed in `/etc/sestatus.conf`. The contexts will specify the roles and types assigned to a particular process, file, or directory.

```
sestatus -v
```

Use the **seinfo** command to display your current SELinux statistics:

```
# seinfo
Statistics for policy file: /etc/selinux/targeted/policy/policy.21
Policy Version & Type: v.21 (binary, MLS)

Classes:           55      Permissions:       206
Types:             1043    Attributes:        85
Users:             3       Roles:             6
Booleans:         135     Cond. Expr.:      138
Sensitivities:    1       Categories:       256
Allow:            46050   Neverallow:       0
Auditallow:       97      Dontaudit:        3465
Role allow:       5       Role trans:       0
Type_trans:      987     Type_change:     14
```































