CHAPTER 17

Security-Enhanced Linux

Though 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	**nsa.gov/selinux**
NSA SELinux FAQ	**nsa.gov/selinux/info/faq.cfm**
SELinux at sourceforge.net	**selinux.sourceforge.net**
Writing SELinux Policy HOWTO	Accessible from "SELinux resources at sourceforge" link at **selinux.sourceforge.net**
NSA SELinux Documentation	**nsa.gov/selinux/info/docs.cfm**
Configuring SELinux Policy	Accessible from NSA SELinux Documentation
SELinux Reference Policy Project	**http://oss.tresys.com/projects/refpolicy**

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.

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

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
```

```
Type_member:          0    Range_trans:         10
Constraints:          0    Validatetrans:        0
Fs_use:              12    Genfscon:            52
Portcon:            190    Netifcon:             0
Nodecon:              8    Initial SIDs:         0
```

Checking Security Context

The **-Z** option used with the **ls**, **id**, and **ps** commands can be used to check the security context for files, users, and processes respectively. The security context tells you the roles that users must have to access given processes or objects.

```
ls -lZ
id -Z
ps -eZ
```

SELinux Management Tools

SELinux provides a number of tools to let you manage your SELinux configuration and policy implementation, including semanage to configure your policy. The setools collection provides SELinux configuration and analysis tools including **apol**, the Security Policy Analysis tool for domain transition analysis, **sediffx** for policy differences, and **seaudit** to examine the auditd logs (see Table 17-2). The command line user management tools, **useradd**, **usermod**, and **userdel**, all have SELinux options that can be applied when SELinux is installed. In addition, the **audit2allow** tool will convert SELinux denial messages into policy modules that will allow access.

Command	Description
seinfo	Displays policy statistics.
sestatus	Checks status of SELinux on your system, including the contexts of processes and files.
sesearch	Searches for Type Enforcement rules in policies.
seaudit	Examines SELinux log files.
sediffx	Examines SELinux policy differences.
autid2allow	Generates policy to allow rules for modules using audit AVC denial messages.
apol	The SELinux Policy Analysis tool.
checkpolicy	The SELinux policy compiler.
fixfiles	Checks file systems and sets security contexts.
restorecon	Sets security features for particular files.
newrole	Assigns new role.
setfiles	Sets security context for files.
chcon	Changes context.
chsid	Changes security ID.

Table 17-2 SELinux Tools

With the modular version of SELinux, policy management is no longer handled by editing configuration files directly. Instead, you use the SELinux management tools such as the command line tool **semanage**. Such tools make use of interface files to generate changed policies.

semanage

semanage lets you change your SELinux configuration without having to edit SELinux source files directly. It covers several major categories including users, ports, file contexts, and logins. Check the Man page for **semanage** for detailed descriptions. Options let you modify specific security features such as **-s** for the username, **-R** for the role, **-t** for the type, and
-r for an MLS security range. The following example adds a user with role user_r.

```
semanage user -a -R user_r  justin
```

semanage is configured with the **/etc/selinux/semanage.conf** file, where you can set semanage to write directly on modules (the default) or work on the source.

The Security Policy Analysis Tool: apol

The SELinux Policy Analysis tool, **apol**, provides a complex and detailed analysis of a selected policy. Select the **apol** entry in the Administration menu to start it.

Checking SELinux Messages: seaudit

SELinux AVC messages are now saved in the **/var/log/audit/audit.log** file. These are particularly important if you are using the permissive mode to test a policy you want to later enforce. You need to find out if you are being denied access when appropriate, and afforded control when needed. To see only the SELinux messages, you can use the seaudit tool. Startup messages for the SELinux service are still logged in **/var/log/messages**.

Allowing Access: chcon and audit2allow

Whenever SELinux denies access to a file or application, the kernel issues an AVC notice. In many cases the problem can be fixed simply by renaming the security context of a file to allow access. You use the **chcon** command to change a file's security context. In this rename, access needs to be granted to the Samba server for a **log.richard3** file in the **/var/lib/samba** directory.

```
chcon -R -t samba_share_t log.richard3
```

More complicated problems, especially ones that are unknown, may require you to create a new policy module using the AVC messages in the audit log. To do this, you can use the **audit2allow** command. This command will take an audit AVC messages and generate commands to allow SELinux access. The audit log is **/var/log/autid/audit.log**. This log is outputted to **audit2allow**, which then can use its **-M** option to create a policy module.

```
cat /var/log/audit/audit.log | audit2allow -M local
```

You then use the semodule command to load the module:

```
semodule -i local.pp
```

If you want to first edit the allowable entries, you can use the following to create a **.te** file of the local module, **local.te**, which you can then edit:

```
audit2allow -m local -i  /var/log/audit/audit.log   >  local.te
```

Once you have edited the **.te** file, you can use **checkmodule** to compile the module, and then **semodule_package** to create the policy module, **local.pp**. Then you can install it with **semodule**. You first create a **.mod** file with **checkmodule**, and then a **.pp** file with **semodule_package**.

```
checkmodule -M -m -o local.mod local.te
semodule_package -o local.pp  -m local.mod


semodule -i local.pp
```

In this example the policy module is called **local**. If you later want to create a new module with **audit2allow**, you should either use a different name or append the output to the **.te** file using the **-o** option.

TIP *On Red Hat and Fedora distributions, you can use the SELinux Troubleshooter to detect SELinux access problems.*

The SELinux Reference Policy

A system is secured using a policy. SELinux now uses a single policy, the reference policy, instead of the two separate targeted and strict policies used in previous editions (see **serefpolicy.sourceforge.net**). Instead of giving users just two alternatives, strict and targeted, the SELinux reference policy project aims to provide a basic policy that can be easily adapted and expanded as needed. The SELinux reference policy configures SELinux into modules that can be handled separately. You still have strict and targeted policies, but they are variations on a basic reference policy. In addition, you can have an MLS policy for Multi-Level Security. The targeted policy is installed by default, and you can install the strict or MLS policies yourself.

On some distributions, such as Fedora, there may be separate policy configurations already provided. For example, Fedora currently provides three effective policies : targeted, strict, and mls. The targeted policy is used to control specific services, like network and Internet servers such as web, DNS, and FTP servers. It also can control local services with network connections. The policy will not affect just the daemon itself, but all the resources it uses on your system.

The strict policy provides complete control over your system. It is under this kind of policy that your users, and even administrators, can be inadvertently locked out of the system. A strict policy needs to be carefully tested to make sure access is denied or granted when appropriate.

There will be **targeted**, **strict**, and **mls** subdirectories in your **/etc/selinux** directory, but they now each contain a **modules** directory. It is here that you will find your SELinux configurations.

Multi-Level Security (MLS)

Multi-Level Security (MLS) add a more refined security access method. 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)

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

Policy Methods

Operating system services and components are categorized in SELinux by their type and their role. Rules controlling these objects can be type based or role based. Policies are implemented using two different kinds of rules, Type Enforcement (TE) and Role Based Access Control (RBAC). Multi-Level Security (MLS) is an additional method further restricting access by security level. Security context now features both the role of an object, such as a user, and that object's security level.

Type Enforcement

With a type structure, the operating system resources are partitioned off into types, with each object assigned a type. Processes are assigned to domains. Users are restricted to certain domains and allowed to use only objects accessible in those domains.

Role-Based Access Control

A role-based approach focuses on controlling users. Users are assigned roles that define what resources they can use. In a standard system, file permissions, such as those for groups, can control user access to files and directories. With roles, permissions become more flexible and refined. Certain users can have more access to services than others.

SELinux Users

Users will retain the permissions available on a standard system. In addition, SELinux can set up its own controls for a given user, defining a role for that user. General security identities created by SELinux include:

- **system_u** The user for system processes
- **user_u** To allow normal users to use a service
- **root** For the root user

Policy Files

Policies are implemented in policy files. These are binary files compiled from source files. For a preconfigured targeted policy file, the policy binary files are in policy subdirectories in the **/etc/selinux** configuration directory, **/etc/selinux/targeted**. For example, the policy file for the targeted policy is

```
/etc/selinux/targeted/policy/policy.20
```

The targeted development files that hold the interface files are installed at **/usr/share/selinux**.

```
/usr/share/selinux/targeted
```

You can use the development files to create your own policy modules that you can then load.

SELinux Configuration

Configuration for general SELinux server settings is carried out in the **/etc/selinux/config** file. Currently there are only two settings to make: the state and the policy. You set the SELINUX variable to the state, such as enforcing or permissive, and the SELINUXTYPE variable to the kind of policy you want. These correspond to the Securitylevel-config SELinux settings for disabled and enforcing, as well as the policy to use, such as targeted (the targeted name may be slightly different on different distributions, like **refpolicy-targeted** used on Debian). A sample config file is shown here:

```
# 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 - SELinux is fully disabled.
SELINUX=permissive
# SELINUXTYPE= type of policy in use. Possible values are:
#     targeted - Only targeted network daemons are protected.
#     strict - Full SELinux protection.
SELINUXTYPE=targeted
```

SELinux Policy Rules

Policy rules can be made up of either type (Type Enforcement, or TE) or RBAC (Role Based Access Control) statements, along with security levels (Multi-Level Security). A type statement can be a type or attribute declaration or a transition, change, or assertion rule. The RBAC statements can be role declarations or dominance, or they can allow roles. A security level specifies a number corresponding to the level of access permitted. Policy configuration can be difficult, using extensive and complicated rules. For this reason, many rules are implemented using M4 macros in **fi** files that will in turn generate the appropriate rules (Sendmail uses M4 macros in a similar way). You will find these rules in files in the SELinux reference policy source code package that you need to download and install.

Type and Role Declarations

A type declaration starts with the keyword **type**, followed by the type name (identifier) and any optional attributes or aliases. The type name will have a _t suffix. Standard type definitions are included for objects such as files. The following is a default type for any file, with attributes file_type and sysadmfile:

```
type file_t, file_type, sysadmfile;
```

The root will have its own type declaration:

```
type root_t, file_type, sysadmfile;
```

Specialized directories such as the boot directory will also have their own type:

```
type boot_t, file_type, sysadmfile;
```

More specialized rules are set up for specific targets like the Amanda server. The following example is the general type definition for amanda_t objects, those objects used by the Amanda backup server, as listed in the targeted policy's **src/program/amanda.te** file:

```
type amanda_t, domain, privlog, auth, nscd_client_domain;
```

A role declaration determines the roles that can access objects of a certain type. These rules begin with the keyword **role** followed by the role and the objects associated with that role. In this example, the amanda objects (amanda_t) can be accessed by a user or process with the system role (system_r):

```
role system_r types amanda_t;
```

A more specific type declaration is provided for executables, such as the following for the Amanda server (amanda_exec_t). This defines the Amanda executable as a system administration–controlled executable file.

```
type amanda_exec_t, file_type, sysadmfile, exec_type;
```

Associated configuration files often have their own rules:

```
type amanda_config_t, file_type, sysadmfile;
```

In the targeted policy, a general unconfined type is created that user and system roles can access, giving complete unrestricted access to the system. More specific rules will restrict access to certain targets like the web server.

```
type unconfined_t, domain, privuser, privhome, privrole, privowner, admin,
auth_write, fs_domain, privmem;
role system_r types unconfined_t;
role user_r types unconfined_t;
role sysadm_r types unconfined_t;
```

Types are also set up for the files created in the user home directory:

```
type user_home_t, file_type, sysadmfile, home_type;
type user_home_dir_t, file_type, sysadmfile, home_dir_type;
```

File Contexts

File contexts associate specific files with security contexts. The file or files are listed first, with multiple files represented with regular expressions. Then the role, type, and security level are specified. The following creates a security context for all files in the **/etc** directory (configuration files). These are accessible from the system user (system_u) and are objects of the etc_t type with a security level of 0, s0.

```
/etc(/.*)?              system_u:object_r:etc_t:s0
```

Certain files can belong to other types; for instance, the **resolve.conf** configuration file belongs to the net_conf type:

```
/etc/resolv\.conf.*     --      system_u:object_r:net_conf_t:s0
```

Certain services will have their own security contexts for their configuration files:

```
/etc/amanda(/.*)?            system_u:object_r:amanda_config_t:s0
```

File contexts are located in the **file_contexts** file in the policy's contexts directory, such as **/etc/selinux/targeted/contexts/files/file_contexts**. The version used to create or modify the policy is located in the policy modules active directory, as in **targeted/modules/active/file_contexts**.

User Roles

User roles define what roles a user can take on. Such a role begins with the keyword **user** followed by the username, then the keyword **roles**, and finally the roles it can use. You will find these rules in the SELinux reference policy source code files. The following example is a definition of the system_u user:

```
user system_u roles system_r;
```

If a user can have several roles, then they are listed in brackets. The following is the definition of the standard user role in the targeted policy, which allows users to take on system administrative roles:

```
user user_u roles { user_r sysadm_r system_r };
```

The strict policy lists only the user_r role:

```
user user_u roles { user_r };
```

Access Vector Rules: allow

Access vector rules are used to define permissions for objects and processes. The **allow** keyword is followed by the object or process type and then the types it can access or be accessed by and the permissions used. The following allows processes in the amanda_t domain to search the Amanda configuration directories (any directories of type amanda_config_t):

```
allow amanda_t amanda_config_t:dir search;
```

The following example allows Amanda to read the files in a user home directory:

```
allow amanda_t user_home_type:file { getattr read };
```

The next example allows Amanda to read, search, and write files in the Amanda data directories:

```
allow amanda_t amanda_data_t:dir { read search write };
```

Role Allow Rules

Roles can also have allow rules. Though they can be used for domains and objects, they are usually used to control role transitions, specifying whether a role can transition to another role. These rules are listed in the RBAC configuration file. The following entry allows the user to transition to a system administrator role:

```
allow user_r sysadm_r;
```

Transition and Vector Rule Macros

The type transition rules set the type used for rules to create objects. Transition rules also require corresponding access vector rules to enable permissions for the objects or processes. Instead of creating separate rules, macros are used that will generate the needed rules. The following example sets the transition and access rules for user files in the home directory, using the file_type_auto_trans macro:

```
file_type_auto_trans(privhome, user_home_dir_t, user_home_t)
```

The next example sets the Amanda process transition and acce ss rules for creating processes:

```
domain_auto_trans(inetd_t, amanda_inetd_exec_t, amanda_t)
```

Constraint Rules

Restrictions can be further placed on processes such as transitions to ensure greater security. These are implemented with constraint definitions in the constraints file. Constraint rules are often applied to transition operations, such as requiring that, in a process transition, user identities remain the same, or that process 1 be in a domain that has the privuser attribute and process 2 be in a domain with the userdomain attribute. The characters u, t, and r refer to user, type, and role, respectively.

```
constrain process transition
    ( u1 == u2 or ( t1 == privuser and t2 == userdomain )
```

SELinux Policy Configuration Files

Configuration files are normally changed using **.te** and **.fc** files. These are missing from the module headers in **/usr/share/selinux**. If you are adding a module you will need to create the **.te** and **.fc** files for it. Then you can create a module and add it as described in the next section. If you want to create or modify your own policy, you will need to download and

install the source code files for the SELinux reference policy, as described the section after "**Using SELinux Source Configuration**". The reference policy code holds the complete set of **.te** and **.fc** configuration files.

Compiling SELinux Modules

Instead of compiling the entire source each time you want to make a change, you can just compile a module for the area you changed. The modules directory holds the different modules. Each module is built from a corresponding **.te** file. The **checkmodule** command is used to create a **.mod** module file from the **.te** file, and then the **semodule_package** command is used to create the loadable **.pp** module file as well as a **.fc** file context file. As noted in the SELinux documentation, if you need to change the configuration for **syslogd**, you first use the following to create a **syslogd.mod** file using **syslogd.te**. The **-M** option specifies support for MLS security levels.

```
checkmodule -M -m syslogd.te  -o syslogd.mod
```

Then use the **semodule_package** command to create a **syslogd.pp** file from the **syslogd .mod** file. The **-f** option specifies the file context file.

```
semodule_package -m syslogd.mod  -o syslogd.pp -f syslogd.fc
```

To add the module you use **semodule** and the **-i** option. You can check if a module is loaded with the **-l** option.

```
semodule -i syslogd.pp
```

Changes to the base policy are made to the **policy.conf** file, which is compiled into the **base.pp** module.

Using SELinux Source Configuration

To perform you own configuration, you will have to download and install the source code file for the SELinux reference policy. For RPM distributions, this will be an SRPMS file. The **.te** files used for configuring SELinux are no longer part of the SELinux binary packages.

NOTE *On Red Hat and Fedora distributions, the compressed archive of the source, a tgz file, along with various policy configuration files, will be installed to* ***/usr/src/redhat/SOURCES****. (Be sure you have already installed* ***rpm-build****; it is not installed by default.) You use an* ***rpmbuild*** *operation with the* ***security-policy.spec*** *file to extract the file to the* ***serefpolicy*** *directory in* ***/usr/src/redhat/BUILD****.*

Change to the **seref-policy** directory and run the following command to install the SELinux source to **/etc/selinux/serefpolicy/src**.

```
make install-src
```

The rules are held in configuration files located in various subdirectories in a policy's **src** directory. Within this directory you will find a **policy/modules** subdirectory. There, organized into several directories, such as **admin** and **apps**, you will find the **.tc**, **.fc**, and **.if** configuration files.

You will have configuration files for both Type Enforcement and security contexts. Type Enforcement files have the extension **.te**, whereas security contexts have an **.sc** extension.

Reflecting the fine-grained control that SELinux provides, you have numerous module configuration files for the many kinds of objects and processes on your system. The primary configuration files and directories are listed in Table 17-3, but several expand to detailed listing of subdirectories and files.

Interface Files

File *interface* files allow management tools to generate policy modules. They define interface macros for your current policy. The **refpolicy** SELinux source file will hold **.if** files for each module, along with **.te** and **.fc** files. Also, the **.if** files in the **/usr/share/selinux/devel** directory can be used to generate modules.

Directories and Files	Description
assert.te	Access vector assertions
config/appconfig-*	Application runtime configuration files
policy/booleans.conf	Tunable features
file_contexts	Security contexts for files and directories
policy/flask	Flask configuration
policy/mcs	Multi-Category Security (MCS) configuration
doc	Policy documentation support
policy/modules	Security policy modules
policy/modules.conf	Module list and use
policy/modules/admin	Administration modules
policy/modules/apps	Application modules
policy/modules/kernel	Kernel modules
policy/modules/services	Services and server modules
policy/modules/system	System modules
policy/rolemap	User domain types and roles
policy/users	General users definition
config/local.users	Your own SELinux users
policy/constraints	Additional constraints for role transition and object access
policygentool	Script to generate policies
policy/global_tunables	Policy tunables for customization
policy/mls	Multi-Level Security (MLS) configuration

Table 17-3 SELinux Policy Configuration Files

Types Files

In the targeted policy, the modules directory that defines types holds a range of files, including **nfs.te** and **network.te** configuration files. Here you will find type declarations for the different kinds of objects on your system. The **.te** files are no longer included with your standard SELinux installation. Instead, you have to download and install the serefpolicy source package. This is the original source and allows you to completely reconfigure your SELinux policy, instead of managing modules with management tools like **semanage**. The modules directory will hold **.te** files for each module, listing their TE rules.

Module Files

Module are located among several directories in the **policy/modules** directory. Here you will find three corresponding files for each application or service. There will be a **.te** file that contains the actual Type Enforcement rules, an **.if**, for interface (a file that allows other applications to interact with the module), and the **.fc** files that define the file contexts.

Security Context Files

Security contexts for different files are detailed in security context files. The **file_contexts** file holds security context configurations for different groups, directories, and files. Each configuration file has an **.fc** extension. The **types.fc** file holds security contexts for various system files and directories, particularly access to configuration files in the **/etc** directory. In the SELinux source, each module will have its own **.fc** file, along with corresponding **.te** and **.if** files. The **distros.fc** file defines distribution-dependent configurations. The **homedir_template** file defines security contexts for dot files that may be set up in a user's home directory, such as **.mozilla**, **.gconf**, and **.java**.

A modules directory has file context files for particular applications and services. For example, **apache.fc** has the security contexts for all the files and directories used by the Apache web server, such as **/var/www** and **/etc/httpd**.

User Configuration: Roles

Global user configuration is defined in the policy directory's **users** file. Here you find the user definitions and the roles they have for standard users (user_u) and administrators (admin_u). To add your own users, you use the **local.users** file. Here you will find examples for entering your own SELinux users. Both the strict and targeted policies use the general user_u SELinux identity for users. To set up a separate SELinux identity for a user, you define that user in the **local.users** file.

The **rbac** file defines the allowed roles one role can transition to. For example, can the user role transition to an system administration role? The targeted policy has several entries allowing a user to freely transition to an administrator, and vice versa. The strict policy has no such definitions.

Role transitions are further restricted by rules in the **constraints** file. Here the change to other users is controlled, and changing object security contexts (labeling) is restricted.

Policy Module Tools

To create a policy module and load it, you use several policy module tools. First the **checkmodule** command is used to create **.mod** file from a **.te** file. Then the **semodule_package** tool takes the **.mod** file and any supporting **.fc** file, and generates a module policy

package file, **.pp**. Finally, the **semodule** tool can take the policy package file and install it as part of your SELinux policy.

Application Configuration: appconfig

Certain services and applications are security aware and will request default security contexts and types from SELinux (see also the upcoming section "Runtime Security Contexts and Types: contexts"). The configuration is kept in files located in the **policy/config/appconfig-*** directory. The **default_types** file holds type defaults; **default_contexts** holds default security contexts. The **initrc_context** file has the default security context for running **/etc/rc.d** scripts. A special **root_default_contexts** file details how the root user can be accessed. The **removable_context** file holds the security context for removable devices, and **media** lists media devices, such as cdrom for CD-ROMs. Runtime values can also be entered in corresponding files in the policy contexts directory, such as **/etc/selinux/targeted/contexts**.

Creating an SELinux Policy: make and checkpolicy

If you want to create an entirely new policy, you use the SELinux reference policy source, **/etc/selinux/serefpolicy**. Once you have configured your policy, you can create it with the **`make policy`** and **`checkpolicy`** commands. The **`make policy`** command generates a **policy.conf** file for your configuration files, which **`checkpolicy`** can then use to generate a policy binary file. A policy binary file will be created in the **policy** subdirectory with a numeric extension for the policy version, such as **policy.20**.

You will have to generate a new **policy.conf** file. To do this you enter the following command in the policy src directory, which will be **/etc/selinux/serefpolicy/src/policy**.

```
make policy
```

Then you can use **`checkpolicy`** to create the new policy.

Instead of compiling the entire source each time you want to make a change, you can just compile a module for the area you changed. (In the previous SELinux version, you always had to recompile the entire policy every time you made a change.) The modules directory holds the different modules. Each module is built from a corresponding **.te** file. The **`checkmodule`** command is used to create a **.mod** module file from the **.te** file, and then the **`semanage_module`** command is used to create the loadable policy package **.pp** module file. As noted in the SELinux documentation, if you need to just change the configuration for **syslogd**, you would first use the following to create a **syslogd.mod** file using **syslogd.te**. The **`-M`** option specifies support for MLS security levels.

```
checkmodule -M  -m syslogd.te  -o syslogd.mod
```

Then use the **`semanage_module`** command to create a **syslogd.pp** file from the syslogd .mod file. The **`-f`** option specifies the file context file.

```
semanage_module -m syslogd.mod  -o syslogd.pp -f syslogd.fc
```

To add the module, you use **`semodule`** and the **`-i`** option. You can check if a module is loaded with the **`-l`** option.

```
semodule -i syslogd.pp
```

Changes to the base policy are made to the **policy.conf** file, which is compiled into the **base.pp** module.

To perform your own configuration, you will now have to download the source code files. The **.te** files used for configuring SELinux are no longer part of the SELinux binary packages. Once installed, the source will be in the **sefepolicy** directory in **/etc/selinux**.

SELinux: Administrative Operations

There are several tasks you can perform on your SELinux system without having to recompile your entire configuration. Security contexts for certain files and directories can be changed as needed. For example, when you add a new file system, you will need to label it with the appropriate security contexts. Also, when you add users, you may need to have a user be given special attention by the system.

Using Security Contexts: fixfiles, setfiles, restorecon, and chcon

Several tools are available for changing your objects' security contexts. The **fixfiles** command can set the security context for file systems. You use the **relabel** option to set security contexts and the **check** option to see what should be changed. The **fixfiles** tool is a script that uses **setfiles** and **restorecon** to make actual changes.

The **restorecon** command will let you restore the security context for files and directories, but **setfiles** is the basic tool for setting security contexts. It can be applied to individual files or directories. It is used to label the file when a policy is first installed.

With **chcon**, you can change the permissions of individual files and directories, much as **chmod** does for general permissions.

Adding New Users

If a new user needs no special access, you can generally just use the generic SELinux user_u identity. If, however, you need to allow the user to take on roles that would otherwise be restricted, such as a system administrator role in the strict policy, you need to configure the user accordingly. To do this, you add the user to the **local.users** file in the policy users directory, as in **/etc/selinux/targeted/policy/users/local.users**. Note that this is different from the **local.users** file in the **src** directory, which is compiled directly into the policy. The user rules have the syntax

```
user username roles { rolelist };
```

The following example adds the sysadm role to the george user:

```
user george roles { user_r sysadm_r };
```

Once the role is added, you have to reload the policy.

```
make reload
```

You can also manage users with the **semanage** command with the **user** option. To see what users are currently active, you can list them with the **semanage user** command and the **-l** option.

```
# semanage user -l
system_u: system_r
user_u: user_r sysadm_r system_r
root: user_r sysadm_r system_r
```

The **semanage user** command has **a**, **d**, **m**, options for adding, removing, or changing users, respectively. The **a** and **m** options let you specify roles to add to a user, whereas the **d** option will remove the user.

Runtime Security Contexts and Types: contexts

Several applications and services are security aware, and will need default security configuration information such as security contexts. Runtime configurations for default security contexts and types are kept in files located in the policy context directory, such as **/etc/selinux/targeted/contexts**. Types files will have the suffix **_types**, and security context files will use **_context**. For example, the default security context for removable files is located in the **removable_context** file. The contents of that file are shown here:

```
system_u:object_r:removable_t
```

The **default_context** file is used to assign a default security context for applications. In the strict policy it is used to control system admin access, providing it where needed, for instance, during the login process.

The following example sets the default roles for users in the login process:

```
system_r:local_login_t user_r:user_t
```

This allows users to log in either as administrators or as regular users.

```
system_r:local_login_t sysadm_r:sysadm_t user_r:user_t
```

This next example is for remote user logins, for which system administration is not included:

```
system_r:remote_login_t user_r:user_t staff_r:staff_t
```

The **default_types** file defines default types for roles. This files has role/type entries, and when a transition takes place to a new role, the default type specified here is used. For example, the default type for the sysadm_r role is sysadm_t.

```
sysadm_r:sysadm_t
user_r:user_t
```

Of particular interest is the **initrc_context** file, which sets the context for running the system scripts in the **/etc/rc.d** directory. In the targeted policy these are open to all users.

```
user_u:system_r:unconfined_t
```

In the strict policy these are limited to the system user.

```
system_u:system_r:initrc_t
```

users

Default security contexts may also need to be set up for particular users such as the root user. In the **sesusers** file you will find a root entry that lists roles, types, and security levels the root user can take on, such as the following example for the su operation (on some distributions this may be a **users** directory with separate files for different users):

```
sysadm_r:sysadm_su_t  sysadm_r:sysadm_t staff_r:staff_t user_r:user_t
```

context/files

Default security contexts for your files and directories are located in the **contexts/files** directory. The **file_contexts** directory lists the default security contexts for all your files and directories, as set up by your policy. The **file_context.homedirs** directory sets the file contexts for user home directory files as well as the root directory, including dot configuration files like **.mozilla** and **.gconf**. The media file sets the default context for media devices such as CD-ROMs and disks.

```
cdrom system_u:object_r:removable_device_t
floppy system_u:object_r:removable_device_t
disk system_u:object_r:fixed_disk_device_t
```