Chapter 41 Linux Security Modules - Notes

41.3 Learning Objectives:

- Understand how the Linux Security Module framework works and how it is deployed.
- List the various LSM implementations available.
- Delineate the main features of SELinux.
- Explain the different modes and policies available.
- Grasp the importance of contexts and how to get and set them.
- Know how to use the important SELinux utility programs.
- Gain some familiarity with AppArmor.

41.4 What Are Linux Security Modules?

A modern computer system must be made secure, but needs vary according to sensitivity of data, number of users with accounts, exposure to outside networks, legal requirements, and other factors. Resposibility for enabling good security controls falls both on application designers and the Linux kernel developers and maintainers. Users should have to follow good procedures as well, but on a well run system, non-privileged users should have very limited ability to expose system to security violations. In this section, concerned with how Linux kernel enhances security through use of Linux Security Modules framework, particularly with deployment of SELinux. Idea is to implement **mandatory access controls** over variety of requests made to kernel, but to do so in a way that:

1. Minimizes changes to the kernel
2. Minimizes overhead on the kernel
3. Permits flexibility and choice between different implementations, each of which is presented as a self-contained **LSM** (**L**inux **S**ecurity **M**odule).

Basic idea to **hook system calls**; insert code wherever an application requests transition to kernel (system) mode in order to accomplish work that requires enhanced abilities; this code makes sure permissions are valid, malicious intent is protected against, etc. Does this by invoking security-related functional steps before and/or after a system call is fulfilled by kernel.

41.5 LSM Choices

For a long time, only enhanced security model implemented was **SELinux**. When project was first floated upstream in 2001 to be included directly in kernel, there were objections about using only one approach to enhanced security.

As a result, LSM approach was adopted, where alternative modules to SELinux could be used as they were developed and was incorporated into Linux kernel in 2003.

Current LSM implementations:

- SELinux
- AppArmor
- Smack
- Tomoyo

Only one LSM can be used at a time, as they potentially modify the same parts of the Linux kernel.

Will concentrate primarily on SELinux and secondarily on AppArmor in order of usage volume.

41.6 SELinux Overview

SELinux originally developed by United States NSA (**N**ational **S**ecurity **A**dministration)
and has been integral in RHEL for a very long time, which has brought it a large usage base.

Operationally, SELinux is a set of security rules that are used to determine which processes can access which files, directories, ports, and other items on the system.

Works with three conceptual quantities:

1. **Contexts**: Are labels to files, processes, and ports. Examples of contexts are SELinux user, role, and type.

2. **Rules**: Describe access control in terms of contexts, processes, files, ports, users, etc.

3. **Policies**: Are a set of **rules** that describe what system-wide access control decisions should be made by SELinux.

   A SELinux context is a name used by a rule to define how users, processes, files, and ports interact with each other. As the default policy is to deny any access, rules are used to describe allowed actions on the system.

41.7 SELinux Modes

SELinux can be run under one of the three following modes:

- **Enforcing**: All SELinux code is operative and access is denied according to policy. All violations are audited and logged.
- **Permissive**: Enables SELinux code, but only audits and warns about operations that would be denied in enforcing mode.
- **Disabled**: Completely disables SELinux kernel and application code, leaving the system without any of its protections.

These modes are selected (and explained) in a file (usually `/etc/selinux/config`) whose location varies by distribution (it is often either at `/etc/sysconfig/selinux` or linked from there). The file is well self-documented. The **sestatus** utility can display the current mode and policy.

SELinus_Mode_Enforcing_large **SELinux Enforcing Mode**

SELinus_Mode_Permissive_large **SELinux Permissive Mode**

To examine or set current mode, one can use **getenforce** and **setenforce**:

```
$ getenforce
Disabled
$ sudo setenforce Permissive
$ getenforce
Permissive
```

setenforce can be used to switch between **enforcing** and **permissive** modes on the fly while system is in operation.
However, changing in or out of the **disabled** mode cannot be done this way. While **setenforce** allows you to switch between **permissive** and **enforcing** modes, it does not allow you to disable SELinux completely. There are at least two different ways to disable SELinux:

- **Configuration file**

  Edit the SELinux configuration file (usually `/etc/selinux/config`) and set `SELINUX=disabled`. This is the default method and should be used to permanently disable SELinux.

- **Kernel parameter**

Add `selinux=0` to the kernel parameter list when rebooting.

However, important to note that disabling SELinux on systems in which SELinux will be re-enabled is not recommended. Preferable to use the **permissive** mode instead of disabling SELinux, so as to avoid relabeling the entire filesystem, which can be time-consuming.

41.8 SELinux Policies

The same configuration file, usually `/etc/sysconfig/selinux`, also sets the **SELinux policy**. Multiple policies are allowed, but only one can be active at a time. Changing the policy may require a reboot of the system and a time-consuming re-labeling of filesystem contents. Each policy has files which must be installed under `/etc/selinux/[SELINUXTYPE]`.

Most common policies:

- **targeted**

  The **default** policy in which SELinux is more restricted to targeted processes. User processes and **init** processes are not targeted. SELinux enforces memory restrictions for **all** processes, which reduces the vulnerability to buffer overflow attacks.

- **minimum**

  A modification of the targeted policy where only selected processes are protected.

- **MLS**

  The Multi-Level Security policy is much more restrictive; all processes are placed in fine-grained security domains with particular policies.

41.9 Context Utilities

As mentioned earlier, contexts are labels applied to files, directories, ports, and processes. Those labels are used to describe access rules. There are four SELinux contexts:

- **User**
- **Role**
- **Type**
- **Level**.

However, will focus on **type**, which is most commonly utilized context. Label naming convention determines that type context labels should end with `_t` as in `kernel_t`.

Use `-Z` option to see context:

```
$ ls -Z
$ ps auZ
```

Use **chcon** command to change context:

```
$ chcon -t etc_t somefile
$ chcon --reference somefile so
```

41.10 SELinux and Standard Command Line Tools]

Many standard command line commands, such as **ls** and **ps**, were extended to support SELinux, and corresponding sections were added to their **man** pages explaining the details. Often the parameter `z` is passed to standard command line tools as in:

```
selinux_command_line
```

Other tools that were extended to support SELinux include **cp**, **mv**, and **mkdir**.

Note: if you have disabled SELinux, no useful information is displayed in the related fields from these utilities.

41.11 SELinux Context Inheritance and Preservation

Newly created files inherit the context from their parent directory, but when moving files, it is the context of the source directory which may be preserved, which can cause problems.

Continuing the previous example, can see the context of `tmpfile` not changed by moving the file from `/tmp` to `/home/peter`:

```
selinux_command_line_context
```

Classical example in which moving files creates a SELinux issue is moving files to the `DocumentRoot` directory of the **httpd** server. On SELinux-enabled systems, the web server can only access files with the correct context labels. Creating a file in `/tmp`, and then moving it to the `DocumentRoot` directory, will make the file unaccessible to the **httpd** server until the SELinux context of the file is adjusted.

41.12 restorecon

restorecon resets files contexts, based on parent directory settings. In the following example, **restorecon** resets the default label recursively for all files at the home directory:

```
restorecon
```

Note: context for `tmpfile` has been reset to the default context for files created at the home directory. Type was changed from `user_tmp_t` to `user_home_t`.

41.13 semanage

Another issue is how to configure the default context for a newly created directory. **semanage fcontext** (provided by the **policycoreutils-python** package) can change and display the default context of files and directories. Note: **semanage fcontext** only changes the default settings; does not apply them to existing objects. This requires calling **restorecon** afterwards. For example:

```
semanage
```

The context change from `default_t` to `httpd_sys_content_t` is thus only applied after the call to **restorecon**.

41.14 Using SELinux Booleans

SELinux policy behavior can be configured at runtime without rewriting policy. Accomplished by configuring SELinux Booleans, which are policy paramaters that can be enabled and disabled:

- **getsebool** - to see booleans

- **setsebool** - to set booleans
- **semanage booleans -i** - to see persistent boolean settings.

Can see what needs to be done to list all booleans of current policy, including current status and short description, below.

semanage_screenshot

41.15 getsebool and setsebool

Alternative for displaying boolean information with simpler output: **getsebool -a**, which prints only the boolean name and its current status.

setsebool used for changing boolean status. Default behavior is to apply changes immediately that are not persistent across a reboot. However, `-P` parameter can be supplied in order to make changes persistent.

An example of non-persistent change using **setsebool**:

setsebool_non-persistent

An example of persistent change using **setsebool -P**:

setsebool_persistent

41.16 Troubleshooting Tools

SELinux comes with a set of tools that collect issues at run time, log these issues, and propose solutions to prevent same issues from happening again. These utilities are provided by the **setroubleshoot-server** package. An example of their use:

setroubleshoot1
setroubleshoot2
setroubleshoot3

Note: on RHEL 7, the suggestion is to run:

```
$ grep httpd /var/log/audit/audit.log |  audit2allow -M mypol
```

audit2allow: a tool that generates SELinux policy rules from logs of denied operations. **audit2why**: a similar tool, which translates SELinux audit messages into a description of why the access was denied.

Next example shows how to solve this issue using the **restorecon** tool which was described earlier. Feel free to try both approaches for fixing the SELinux issue.

setroubleshoot_eg

41.17 Additional Online Resources

This section has covered the basics and most common system administration tasks related to SELinux. There are freely available online resources for SELinux advanced topics, including:

- SELinux User's and Administrator's Guide
- Red Hat Enterprise Linux 6 Security-Enhanced Linux

41.18 AppArmor

AppArmor is an LSM alternative to SELinux. Support for it has been incorporated in the Linux kernel since 2006. It has been used by SUSE, Ubuntu, and other distributions.

AppArmor:

- Provides Mandatory Access Control (MAC)
- Allows administrators to associate a security profile to a program which restricts its capabilities
- Is considered easier (by some but not all) to use than SELinux
- Is considered filesystem-neutral (no security labels required).

AppArmor supplements the traditional UNIX Discretionary Access Control (DAC) model by providing Mandatory Access Control (MAC).

In addition to manually specifying profiles, AppArmor includes a learning mode, in which violations of the profile are logged, but not prevented. This log can then be turned into a profile, based on the program's typical behavior.

41.19 Checking Status

Distributions that come with AppArmor tend to enable it and load it by default. Note: Linux kernel has to have it turned on as well, and, in most cases, only one LSM can run at a time.

Assuming you have the AppArmor kernel module available, on a systemd-equipped system you can do:

```
$ sudo systemctl [start|stop|restart|status] apparmor
```

to change or inquire about the current state of operation, or do:

```
$ sudo systemctl [enable|disable] apparmor
```

to cause to be loaded or not loaded at boot.

In order to see the current status, do:

```
$ sudo apparmor_status
apparmor module is loaded.
25 profiles are loaded.
25 profiles are in enforce mode.
    /sbin/dhclient
...
```

Profiles and **processes** are in either **enforce** or **complain** mode, directly analogous to SELinux's **enforcing** and **permissive** modes. Note that in the process, listing the PID is given:

```
$ ps aux | grep libvirtd
root      787  0.0  0.9  527200  35936 ?       Ssl  10:54  0:00  /usr/sbin/libvirtd
student  3346  0.0  0.0   13696   2204 pts/16  S+   11:42  0:00  grep --color=auto libvirtd
```

41.20 Modes and Profiles

Profiles restrict how executable programs, which have pathnames on your system, such as `/usr/bin/evince`, can be used.

Processes can be run in either of the two modes:

- **Enforce Mode**

  Applications are prevented from acting in ways which are restricted. Attempted violations are reported to the system logging files. This is the default mode. A profie can be set to this mode with **aa-enforce**.

- **Complain Mode**

  Policies are not enforced, but attempted policy violations are reported. This is also called the learning mode. A profile can be set to this mode with **aa-complain**

Linux distributions come with pre-packaged profiles, typically installed either when a given package is installed, or with an AppArmor package, such as **apparmor-profiles**. These profiles are stored in `/etc/apparmor.d`.

When installing new software, new profiles can be created specific to any executables in the package.

Exactly what AppArmor profiles are installed on your system depends on your selection of software packages. For example, on one particular Ubuntu system:

```
apparmor_profiles
```

Full documentation on what can go in these files can be obtained by doing **man apparmor.d**.

41.21 AppArmor Utilities

AppArmor has quite a few administrative utilities for monitoring and control. For example, on an OpenSUSE system:

```
apparmor_utilities1
```

Note: many of these utilities can be invoked with either their short or long names:

```
apparmor_utilities2
```

AppArmor Utilities

Program	Use
`apparmor_status`	Show status of all profiles and processes with profiles
`apparmor_notify`	Show a summary for AppArmor log messages
`complain`	Set a specified profile to complain mode
`enforce`	Set a specified profile to enforce mode
`disable`	Unload a specified profile from the current kernel and prevent from being loaded on system startup
`logprof`	Scan log files, and, if AppArmor events that are not covered by existing profiles have been recorded, suggest how to take into account, and, if approved, modify and reload
`easyprof`	Help setup a basic AppArmor profile for a program

##

Back to top

Previous Chapter - Table of Contents - Next Chapter