Red Hat Enterprise Linux 8

Using SELinux

Basic and advanced configuration of Security-Enhanced Linux (SELinux)

Last Updated: 2022-11-30

Red Hat Enterprise Linux 8 Using SELinux

Basic and advanced configuration of Security-Enhanced Linux (SELinux)

Legal Notice

Abstract

This title assists users and administrators in learning the basics and principles upon which SELinux functions and describes practical tasks to set up and configure various services.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message.

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your feedback on our documentation. Let us know how we can improve it.

Submitting comments on specific passages

1. View the documentation in the **Multi-page HTML** format and ensure that you see the **Feedback** button in the upper right corner after the page fully loads.
2. Use your cursor to highlight the part of the text that you want to comment on.
3. Click the **Add Feedback** button that appears near the highlighted text.
4. Add your feedback and click **Submit**.

Submitting feedback through Bugzilla (account required)

1. Log in to the Bugzilla website.
2. Select the correct version from the **Version** menu.
3. Enter a descriptive title in the **Summary** field.
4. Enter your suggestion for improvement in the **Description** field. Include links to the relevant parts of the documentation.
5. Click **Submit Bug**.

CHAPTER 1. GETTING STARTED WITH SELINUX

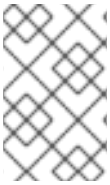
Security Enhanced Linux (SELinux) provides an additional layer of system security. SELinux fundamentally answers the question: *May <subject> do <action> to <object>?*, for example: *May a web server access files in users' home directories?*

1.1. INTRODUCTION TO SELINUX

The standard access policy based on the user, group, and other permissions, known as Discretionary Access Control (DAC), does not enable system administrators to create comprehensive and fine-grained security policies, such as restricting specific applications to only viewing log files, while allowing other applications to append new data to the log files.

Security Enhanced Linux (SELinux) implements Mandatory Access Control (MAC). Every process and system resource has a special security label called an *SELinux context*. A SELinux context, sometimes referred to as an *SELinux label*, is an identifier which abstracts away the system-level details and focuses on the security properties of the entity. Not only does this provide a consistent way of referencing objects in the SELinux policy, but it also removes any ambiguity that can be found in other identification methods. For example, a file can have multiple valid path names on a system that makes use of bind mounts.

The SELinux policy uses these contexts in a series of rules which define how processes can interact with each other and the various system resources. By default, the policy does not allow any interaction unless a rule explicitly grants access.



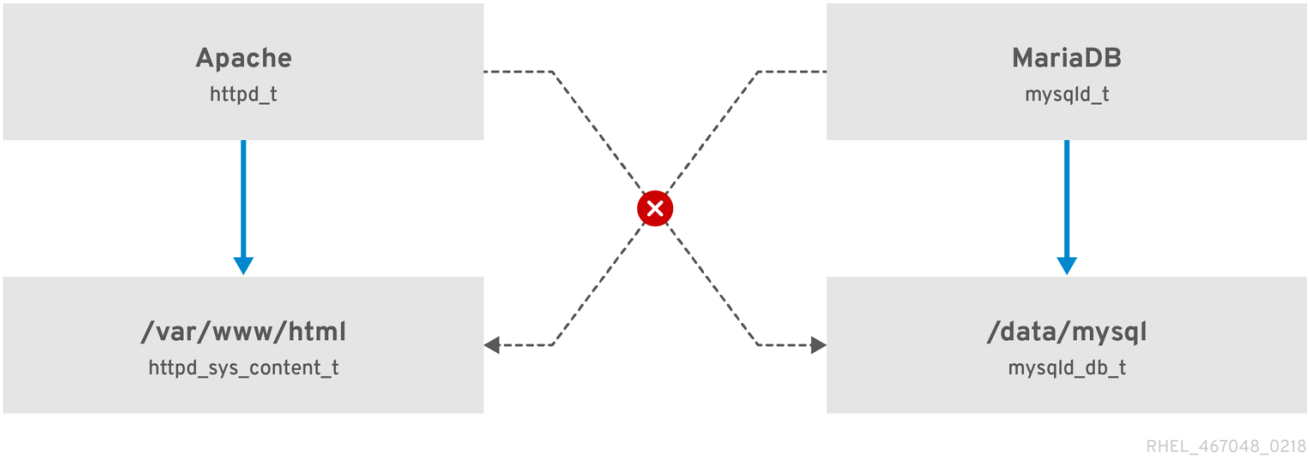
NOTE

Remember that SELinux policy rules are checked after DAC rules. SELinux policy rules are not used if DAC rules deny access first, which means that no SELinux denial is logged if the traditional DAC rules prevent the access.

SELinux contexts have several fields: user, role, type, and security level. The SELinux type information is perhaps the most important when it comes to the SELinux policy, as the most common policy rule which defines the allowed interactions between processes and system resources uses SELinux types and not the full SELinux context. SELinux types end with **_t**. For example, the type name for the web server is **httpd_t**. The type context for files and directories normally found in **/var/www/html/** is **httpd_sys_content_t**. The type contexts for files and directories normally found in **/tmp** and **/var/tmp/** is **tmp_t**. The type context for web server ports is **http_port_t**.

There is a policy rule that permits Apache (the web server process running as **httpd_t**) to access files and directories with a context normally found in **/var/www/html/** and other web server directories (**httpd_sys_content_t**). There is no allow rule in the policy for files normally found in **/tmp** and **/var/tmp/**, so access is not permitted. With SELinux, even if Apache is compromised, and a malicious script gains access, it is still not able to access the **/tmp** directory.

Figure 1.1. An example how can SELinux help to run Apache and MariaDB in a secure way.



As the previous scheme shows, SELinux allows the Apache process running as **httpd_t** to access the **/var/www/html/** directory and it denies the same process to access the **/data/mysql/** directory because there is no allow rule for the **httpd_t** and **mysqld_db_t** type contexts. On the other hand, the MariaDB process running as **mysqld_t** is able to access the **/data/mysql/** directory and SELinux also correctly denies the process with the **mysqld_t** type to access the **/var/www/html/** directory labeled as **httpd_sys_content_t**.

Additional resources

- **selinux(8)** man page and man pages listed by the **apropos selinux** command.
- Man pages listed by the **man -k _selinux** command when the **selinux-policy-doc** package is installed.
- The SELinux Coloring Book helps you to better understand SELinux basic concepts.
- SELinux Wiki FAQ

1.2. BENEFITS OF RUNNING SELINUX

SELinux provides the following benefits:

- All processes and files are labeled. SELinux policy rules define how processes interact with files, as well as how processes interact with each other. Access is only allowed if an SELinux policy rule exists that specifically allows it.
- Fine-grained access control. Stepping beyond traditional UNIX permissions that are controlled at user discretion and based on Linux user and group IDs, SELinux access decisions are based on all available information, such as an SELinux user, role, type, and, optionally, a security level.
- SELinux policy is administratively-defined and enforced system-wide.
- Improved mitigation for privilege escalation attacks. Processes run in domains, and are therefore separated from each other. SELinux policy rules define how processes access files and other processes. If a process is compromised, the attacker only has access to the normal functions of that process, and to files the process has been configured to have access to. For example, if the Apache HTTP Server is compromised, an attacker cannot use that process to read files in user home directories, unless a specific SELinux policy rule was added or configured to allow such access.

- SELinux can be used to enforce data confidentiality and integrity, as well as protecting processes from untrusted inputs.

However, SELinux is not:

- antivirus software,
- replacement for passwords, firewalls, and other security systems,
- all-in-one security solution.

SELinux is designed to enhance existing security solutions, not replace them. Even when running SELinux, it is important to continue to follow good security practices, such as keeping software up-to-date, using hard-to-guess passwords, and firewalls.

1.3. SELINUX EXAMPLES

The following examples demonstrate how SELinux increases security:

- The default action is deny. If an SELinux policy rule does not exist to allow access, such as for a process opening a file, access is denied.
- SELinux can confine Linux users. A number of confined SELinux users exist in the SELinux policy. Linux users can be mapped to confined SELinux users to take advantage of the security rules and mechanisms applied to them. For example, mapping a Linux user to the SELinux **user_u** user, results in a Linux user that is not able to run unless configured otherwise set user ID (setuid) applications, such as **sudo** and **su**.
- Increased process and data separation. The concept of SELinux *domains* allows defining which processes can access certain files and directories. For example, when running SELinux, unless otherwise configured, an attacker cannot compromise a Samba server, and then use that Samba server as an attack vector to read and write to files used by other processes, such as MariaDB databases.
- SELinux helps mitigate the damage made by configuration mistakes. Domain Name System (DNS) servers often replicate information between each other in a zone transfer. Attackers can use zone transfers to update DNS servers with false information. When running the Berkeley Internet Name Domain (BIND) as a DNS server in RHEL, even if an administrator forgets to limit which servers can perform a zone transfer, the default SELinux policy prevent updates for zone files [1] that use zone transfers, by the BIND **named** daemon itself, and by other processes.
- Without SELinux, an attacker can misuse a vulnerability to path traversal on an Apache web server and access files and directories stored on the file system by using special elements such as **../**. If an attacker attempts an attack on a server running with SELinux in enforcing mode, SELinux denies access to files that the **httpd** process must not access. SELinux cannot block this type of attack completely but it effectively mitigates it.
- SELinux in enforcing mode successfully prevents exploitation of kernel NULL pointer dereference operators on non-SMAP platforms (CVE-2019-9213). Attackers use a vulnerability in the **mmap** function, which does not check mapping of a null page, for placing arbitrary code on this page.
- The **deny_ptrace** SELinux boolean and SELinux in enforcing mode protect systems from the **PTRACE_TRACEME** vulnerability (CVE-2019-13272). Such configuration prevents scenarios when an attacker can get **root** privileges.

- The **nfs_export_all_rw** and **nfs_export_all_ro** SELinux booleans provide an easy-to-use tool to prevent misconfigurations of Network File System (NFS) such as accidental sharing **/home** directories.

Additional resources

- SELinux as a security pillar of an operating system - Real-world benefits and examples Knowledgebase article

1.4. SELINUX ARCHITECTURE AND PACKAGES

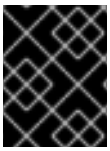
SELinux is a Linux Security Module (LSM) that is built into the Linux kernel. The SELinux subsystem in the kernel is driven by a security policy which is controlled by the administrator and loaded at boot. All security-relevant, kernel-level access operations on the system are intercepted by SELinux and examined in the context of the loaded security policy. If the loaded policy allows the operation, it continues. Otherwise, the operation is blocked and the process receives an error.

SELinux decisions, such as allowing or disallowing access, are cached. This cache is known as the Access Vector Cache (AVC). When using these cached decisions, SELinux policy rules need to be checked less, which increases performance. Remember that SELinux policy rules have no effect if DAC rules deny access first. Raw audit messages are logged to the **/var/log/audit/audit.log** and they start with the **type=AVC** string.

In RHEL 8, system services are controlled by the **systemd** daemon; **systemd** starts and stops all services, and users and processes communicate with **systemd** using the **systemctl** utility. The **systemd** daemon can consult the SELinux policy and check the label of the calling process and the label of the unit file that the caller tries to manage, and then ask SELinux whether or not the caller is allowed the access. This approach strengthens access control to critical system capabilities, which include starting and stopping system services.

The **systemd** daemon also works as an SELinux Access Manager. It retrieves the label of the process running **systemctl** or the process that sent a **D-Bus** message to **systemd**. The daemon then looks up the label of the unit file that the process wanted to configure. Finally, **systemd** can retrieve information from the kernel if the SELinux policy allows the specific access between the process label and the unit file label. This means a compromised application that needs to interact with **systemd** for a specific service can now be confined by SELinux. Policy writers can also use these fine-grained controls to confine administrators.

If a process is sending a **D-Bus** message to another process and if the SELinux policy does not allow the **D-Bus** communication of these two processes, then the system prints a **USER_AVC** denial message, and the D-Bus communication times out. Note that the D-Bus communication between two processes works bidirectionally.



IMPORTANT

To avoid incorrect SELinux labeling and subsequent problems, ensure that you start services using a **systemctl start** command.

RHEL 8 provides the following packages for working with SELinux:

- policies: **selinux-policy-targeted**, **selinux-policy-mls**
- tools: **policycoreutils**, **policycoreutils-gui**, **libselinux-utils**, **policycoreutils-python-utils**, **setools-console**, **checkpolicy**

1.5. SELINUX STATES AND MODES

SELinux can run in one of three modes: enforcing, permissive, or disabled.

- Enforcing mode is the default, and recommended, mode of operation; in enforcing mode SELinux operates normally, enforcing the loaded security policy on the entire system.
- In permissive mode, the system acts as if SELinux is enforcing the loaded security policy, including labeling objects and emitting access denial entries in the logs, but it does not actually deny any operations. While not recommended for production systems, permissive mode can be helpful for SELinux policy development and debugging.
- Disabled mode is strongly discouraged; not only does the system avoid enforcing the SELinux policy, it also avoids labeling any persistent objects such as files, making it difficult to enable SELinux in the future.

Use the **setenforce** utility to change between enforcing and permissive mode. Changes made with **setenforce** do not persist across reboots. To change to enforcing mode, enter the **setenforce 1** command as the Linux root user. To change to permissive mode, enter the **setenforce 0** command. Use the **getenforce** utility to view the current SELinux mode:

```
# getenforce
Enforcing
```

```
# setenforce 0
# getenforce
Permissive
```

```
# setenforce 1
# getenforce
Enforcing
```

In Red Hat Enterprise Linux, you can set individual domains to permissive mode while the system runs in enforcing mode. For example, to make the *httpd_t* domain permissive:

```
# semanage permissive -a httpd_t
```

Note that permissive domains are a powerful tool that can compromise security of your system. Red Hat recommends to use permissive domains with caution, for example, when debugging a specific scenario.

[1] Text files that include DNS information, such as hostname to IP address mappings.

CHAPTER 2. CHANGING SELINUX STATES AND MODES

When enabled, SELinux can run in one of two modes: enforcing or permissive. The following sections show how to permanently change into these modes.

2.1. PERMANENT CHANGES IN SELINUX STATES AND MODES

As discussed in SELinux states and modes, SELinux can be enabled or disabled. When enabled, SELinux has two modes: enforcing and permissive.

Use the **getenforce** or **sestatus** commands to check in which mode SELinux is running. The **getenforce** command returns **Enforcing**, **Permissive**, or **Disabled**.

The **sestatus** command returns the SELinux status and the SELinux policy being used:

```
$ sestatus
SELinux status:                 enabled
SELinuxfs mount:                /sys/fs/selinux
SELinux root directory:         /etc/selinux
Loaded policy name:             targeted
Current mode:                   enforcing
Mode from config file:          enforcing
Policy MLS status:              enabled
Policy deny_unknown status:     allowed
Memory protection checking:     actual (secure)
Max kernel policy version:      31
```

WARNING

When systems run SELinux in permissive mode, users and processes might label various file-system objects incorrectly. File-system objects created while SELinux is disabled are not labeled at all. This behavior causes problems when changing to enforcing mode because SELinux relies on correct labels of file-system objects.

To prevent incorrectly labeled and unlabeled files from causing problems, SELinux automatically relabels file systems when changing from the disabled state to permissive or enforcing mode. Use the **fixfiles -F onboot** command as root to create the **/.autorelabel** file containing the **-F** option to ensure that files are relabeled upon next reboot.

Before rebooting the system for relabeling, make sure the system will boot in permissive mode, for example by using the **enforcing=0** kernel option. This prevents the system from failing to boot in case the system contains unlabeled files required by **systemd** before launching the **selinux-autorelabel** service. For more information, see RHBZ#2021835.

2.2. CHANGING TO PERMISSIVE MODE

Use the following procedure to permanently change SELinux mode to permissive. When SELinux is

running in permissive mode, SELinux policy is not enforced. The system remains operational and SELinux does not deny any operations but only logs AVC messages, which can be then used for troubleshooting, debugging, and SELinux policy improvements. Each AVC is logged only once in this case.

Prerequisites

- The **selinux-policy-targeted**, **libselinux-utils**, and **policycoreutils** packages are installed on your system.
- The **selinux=0** or **enforcing=0** kernel parameters are not used.

Procedure

1. Open the **/etc/selinux/config** file in a text editor of your choice, for example:

```
# vi /etc/selinux/config
```

2. Configure the **SELINUX=permissive** option:

```
# 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=permissive
# SELINUXTYPE= can take one of these two values:
#       targeted - Targeted processes are protected,
#       mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

3. Restart the system:

```
# reboot
```

Verification

1. After the system restarts, confirm that the **getenforce** command returns **Permissive**:

```
$ getenforce
Permissive
```

2.3. CHANGING TO ENFORCING MODE

Use the following procedure to switch SELinux to enforcing mode. When SELinux is running in enforcing mode, it enforces the SELinux policy and denies access based on SELinux policy rules. In RHEL, enforcing mode is enabled by default when the system was initially installed with SELinux.

Prerequisites

- The **selinux-policy-targeted**, **libselinux-utils**, and **policycoreutils** packages are installed on your system.

- The **selinux=0** or **enforcing=0** kernel parameters are not used.

Procedure

1. Open the **/etc/selinux/config** file in a text editor of your choice, for example:

```
# vi /etc/selinux/config
```

2. Configure the **SELINUX=enforcing** option:

```
# 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=enforcing
# SELINUXTYPE= can take one of these two values:
#       targeted - Targeted processes are protected,
#       mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

3. Save the change, and restart the system:

```
# reboot
```

On the next boot, SELinux relabels all the files and directories within the system and adds SELinux context for files and directories that were created when SELinux was disabled.

Verification

1. After the system restarts, confirm that the **getenforce** command returns **Enforcing**:

```
$ getenforce
Enforcing
```

NOTE

After changing to enforcing mode, SELinux may deny some actions because of incorrect or missing SELinux policy rules. To view what actions SELinux denies, enter the following command as root:

```
# ausearch -m AVC,USER_AVC,SELINUX_ERR,USER_SELINUX_ERR -ts today
```

Alternatively, with the **setroubleshoot-server** package installed, enter:

```
# grep "SELinux is preventing" /var/log/messages
```

If SELinux is active and the Audit daemon (**auditd**) is not running on your system, then search for certain SELinux messages in the output of the **dmesg** command:

```
# dmesg | grep -i -e type=1300 -e type=1400
```

See Troubleshooting problems related to SELinux for more information.

2.4. ENABLING SELINUX ON SYSTEMS THAT PREVIOUSLY HAD IT DISABLED

To avoid problems, such as systems unable to boot or process failures, follow this procedure when enabling SELinux on systems that previously had it disabled.

WARNING

When systems run SELinux in permissive mode, users and processes might label various file-system objects incorrectly. File-system objects created while SELinux is disabled are not labeled at all. This behavior causes problems when changing to enforcing mode because SELinux relies on correct labels of file-system objects.

To prevent incorrectly labeled and unlabeled files from causing problems, SELinux automatically relabels file systems when changing from the disabled state to permissive or enforcing mode.

Before rebooting the system for relabeling, make sure the system will boot in permissive mode, for example by using the **enforcing=0** kernel option. This prevents the system from failing to boot in case the system contains unlabeled files required by **systemd** before launching the **selinux-autorelabel** service. For more information, see RHBZ#2021835.

Procedure

1. Enable SELinux in permissive mode. For more information, see Changing to permissive mode.
2. Restart your system:

```
# reboot
```

3. Check for SELinux denial messages.For more information, see Identifying SELinux denials.

4. Ensure that files are relabeled upon the next reboot:

```
# fixfiles -F onboot
```

This creates the **/.autorelabel** file containing the **-F** option.

WARNING

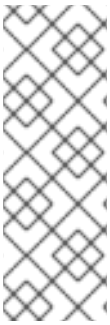
Always switch to permissive mode before entering the **fixfiles -F onboot** command. This prevents the system from failing to boot in case the system contains unlabeled files. For more information, see RHBZ#2021835.

5. If there are no denials, switch to enforcing mode. For more information, see Changing SELinux modes at boot time.

Verification

1. After the system restarts, confirm that the **getenforce** command returns **Enforcing**:

```
$ getenforce
Enforcing
```



NOTE

To run custom applications with SELinux in enforcing mode, choose one of the following scenarios:

- Run your application in the **unconfined_service_t** domain.
- Write a new policy for your application. See the Writing a custom SELinux policy section for more information.

Additional resources

- SELinux states and modes section covers temporary changes in modes.

2.5. DISABLING SELINUX

Use the following procedure to permanently disable SELinux.

IMPORTANT

When SELinux is disabled, SELinux policy is not loaded at all; it is not enforced and AVC messages are not logged. Therefore, all benefits of running SELinux are lost.

Red Hat strongly recommends to use permissive mode instead of permanently disabling SELinux. See Changing to permissive mode for more information about permissive mode.

WARNING

Disabling SELinux using the **SELINUX=disabled** option in the **/etc/selinux/config** results in a process in which the kernel boots with SELinux enabled and switches to disabled mode later in the boot process. Because memory leaks and race conditions causing kernel panics can occur, prefer disabling SELinux by adding the **selinux=0** parameter to the kernel command line as described in Changing SELinux modes at boot time if your scenario really requires to completely disable SELinux.

Procedure

1. Open the **/etc/selinux/config** file in a text editor of your choice, for example:

```
# vi /etc/selinux/config
```

2. Configure the **SELINUX=disabled** option:

```
# 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,
#       mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

3. Save the change, and restart your system:

```
# reboot
```

Verification

1. After reboot, confirm that the **getenforce** command returns **Disabled**:

```
$ getenforce
Disabled
```

2.6. CHANGING SELINUX MODES AT BOOT TIME

On boot, you can set several kernel parameters to change the way SELinux runs:

enforcing=0

Setting this parameter causes the system to start in permissive mode, which is useful when troubleshooting issues. Using permissive mode might be the only option to detect a problem if your file system is too corrupted. Moreover, in permissive mode, the system continues to create the labels correctly. The AVC messages that are created in this mode can be different than in enforcing mode.

In permissive mode, only the first denial from a series of the same denials is reported. However, in enforcing mode, you might get a denial related to reading a directory, and an application stops. In permissive mode, you get the same AVC message, but the application continues reading files in the directory and you get an AVC for each denial in addition.

selinux=0

This parameter causes the kernel to not load any part of the SELinux infrastructure. The init scripts notice that the system booted with the **selinux=0** parameter and touch the **/.autorelabel** file. This causes the system to automatically relabel the next time you boot with SELinux enabled.

IMPORTANT

Red Hat does not recommend using the **selinux=0** parameter. To debug your system, prefer using permissive mode.

autorelabel=1

This parameter forces the system to relabel similarly to the following commands:

```
# touch /.autorelabel
# reboot
```

If a file system contains a large amount of mislabeled objects, start the system in permissive mode to make the autorelabel process successful.

Additional resources

- For additional SELinux-related kernel boot parameters, such as **checkreqprot**, see the **/usr/share/doc/kernel-doc-*<KERNEL_VER>*/Documentation/admin-guide/kernel-parameters.txt** file installed with the **kernel-doc** package. Replace the *<KERNEL_VER>* string with the version number of the installed kernel, for example:

```
# yum install kernel-doc
$ less /usr/share/doc/kernel-doc-4.18.0/Documentation/admin-guide/kernel-parameters.txt
```

CHAPTER 3. MANAGING CONFINED AND UNCONFINED USERS

The following sections explain the mapping of Linux users to SELinux users, describe the basic confined user domains, and demonstrate mapping a new user to an SELinux user.

3.1. CONFINED AND UNCONFINED USERS

Each Linux user is mapped to an SELinux user using SELinux policy. This allows Linux users to inherit the restrictions on SELinux users.

To see the SELinux user mapping on your system, use the **semanage login -l** command as root:

```
# semanage login -l
Login Name           SELinux User         MLS/MCS Range        Service

__default__          unconfined_u         s0-s0:c0.c1023       *
root                 unconfined_u         s0-s0:c0.c1023       *
```

In Red Hat Enterprise Linux, Linux users are mapped to the SELinux ***default*** login by default, which is mapped to the SELinux **unconfined_u** user. The following line defines the default mapping:

```
__default__          unconfined_u         s0-s0:c0.c1023       *
```

Confined users are restricted by SELinux rules explicitly defined in the current SELinux policy. Unconfined users are subject to only minimal restrictions by SELinux.

Confined and unconfined Linux users are subject to executable and writable memory checks, and are also restricted by MCS or MLS.

To list the available SELinux users, enter the following command:

```
$ seinfo -u
Users: 8
   guest_u
   root
   staff_u
   sysadm_u
   system_u
   unconfined_u
   user_u
   xguest_u
```

Note that the **seinfo** command is provided by the **setools-console** package, which is not installed by default.

If an unconfined Linux user executes an application that SELinux policy defines as one that can transition from the **unconfined_t** domain to its own confined domain, the unconfined Linux user is still subject to the restrictions of that confined domain. The security benefit of this is that, even though a Linux user is running unconfined, the application remains confined. Therefore, the exploitation of a flaw in the application can be limited by the policy.

Similarly, we can apply these checks to confined users. Each confined user is restricted by a confined user domain. The SELinux policy can also define a transition from a confined user domain to its own

target confined domain. In such a case, confined users are subject to the restrictions of that target confined domain. The main point is that special privileges are associated with the confined users according to their role.

3.2. SELINUX USER CAPABILITIES

The SELinux policy maps each Linux user to an SELinux user. This allows Linux users to inherit the restrictions of SELinux users.

You can customize the permissions for confined users in your SELinux policy according to specific needs by adjusting booleans in the policy. You can determine the current state of these booleans by using the **semanage boolean -l** command. To list all SELinux users, their SELinux roles, and MLS/MCS levels and ranges, use the **semanage user -l** command as **root**.

Table 3.1. Roles of SELinux users

User	Default role	Additional roles
unconfined_u	**unconfined_r**	**system_r**
guest_u	**guest_r**	
xguest_u	**xguest_r**	
user_u	**user_r**	
staff_u	**staff_r**	**sysadm_r**
		unconfined_r
		system_r
sysadm_u	**sysadm_r**	
root	**staff_r**	**sysadm_r**
		unconfined_r
		system_r
system_u	**system_r**	

Note that **system_u** is a special user identity for system processes and objects, and **system_r** is the associated role. Administrators must never associate this **system_u** user and the **system_r** role to a Linux user. Also, **unconfined_u** and **root** are unconfined users. For these reasons, the roles associated to these SELinux users are not included in the following table Types and access of SELinux roles.

Each SELinux role corresponds to an SELinux type and provides specific access rights.

Table 3.2. Types and access of SELinux roles

Role	Type	Log in using X Window System	**su** and **sudo**	Execute in home directory and **/tmp** (default)	Networking
unconfined_r	**unconfined_t**	yes	yes	yes	yes
guest_r	**guest_t**	no	no	yes	no
xguest_r	**xguest_t**	yes	no	yes	web browsers only (Firefox, GNOME Web)
user_r	**user_t**	yes	no	yes	yes
staff_r	**staff_t**	yes	only **sudo**	yes	yes
auditadm_r	**auditadm_t**		yes	yes	yes
secadm_r	**secadm_t**		yes	yes	yes
sysadm_r	**sysadm_t**	only when the **xdm_sysadm_login** boolean is **on**	yes	yes	yes

- Linux users in the **user_t**, **guest_t**, and **xguest_t** domains can only run set user ID (setuid) applications if SELinux policy permits it (for example, **passwd**). These users cannot run the **su** and **sudo** setuid applications, and therefore cannot use these applications to become root.
- Linux users in the **sysadm_t**, **staff_t**, **user_t**, and **xguest_t** domains can log in using the X Window System and a terminal.
- By default, Linux users in the **staff_t**, **user_t**, **guest_t**, and **xguest_t** domains can execute applications in their home directories and **/tmp**.
  To prevent them from executing applications, which inherit users' permissions, in directories they have write access to, set the **guest_exec_content** and **xguest_exec_content** booleans to **off**. This helps prevent flawed or malicious applications from modifying users' files.
- The only network access Linux users in the **xguest_t** domain have is Firefox connecting to web pages.
- The **sysadm_u** user cannot log in directly using SSH. To enable SSH logins for **sysadm_u**, set the **ssh_sysadm_login** boolean to **on**:

```
# setsebool -P ssh_sysadm_login on
```

Alongside with the already mentioned SELinux users, there are special roles, that can be mapped to those users using the **semanage user** command. These roles determine what SELinux allows the user to do:

- **webadm_r** can only administrate SELinux types related to the Apache HTTP Server.
- **dbadm_r** can only administrate SELinux types related to the MariaDB database and the PostgreSQL database management system.
- **logadm_r** can only administrate SELinux types related to the **syslog** and **auditlog** processes.
- **secadm_r** can only administrate SELinux.
- **auditadm_r** can only administrate processes related to the Audit subsystem.

To list all available roles, enter the **seinfo -r** command:

```
$ seinfo -r
Roles: 14
   auditadm_r
   dbadm_r
   guest_r
   logadm_r
   nx_server_r
   object_r
   secadm_r
   staff_r
   sysadm_r
   system_r
   unconfined_r
   user_r
   webadm_r
   xguest_r
```

Note that the **seinfo** command is provided by the **setools-console** package, which is not installed by default.

Additional resources

- **seinfo(1)**, **semanage-login(8)**, and **xguest_selinux(8)** man pages

3.3. ADDING A NEW USER AUTOMATICALLY MAPPED TO THE SELINUX UNCONFINED_U USER

The following procedure demonstrates how to add a new Linux user to the system. The user is automatically mapped to the SELinux **unconfined_u** user.

Prerequisites

- The **root** user is running unconfined, as it does by default in Red Hat Enterprise Linux.

Procedure

1. Enter the following command to create a new Linux user named *example.user*:

```
# useradd example.user
```

2. To assign a password to the Linux *example.user* user:

```
# passwd example.user
Changing password for user example.user.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

3. Log out of your current session.

4. Log in as the Linux *example.user* user. When you log in, the **pam_selinux** PAM module automatically maps the Linux user to an SELinux user (in this case, **unconfined_u**), and sets up the resulting SELinux context. The Linux user's shell is then launched with this context.

Verification

1. When logged in as the *example.user* user, check the context of a Linux user:

```
$ id -Z
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

Additional resources

- **pam_selinux(8)** man page.

3.4. ADDING A NEW USER AS AN SELINUX-CONFINED USER

Use the following steps to add a new SELinux-confined user to the system. This example procedure maps the user to the SELinux **staff_u** user right with the command for creating the user account.

Prerequisites

- The **root** user is running unconfined, as it does by default in Red Hat Enterprise Linux.

Procedure

1. Enter the following command to create a new Linux user named *example.user* and map it to the SELinux **staff_u** user:

```
# useradd -Z staff_u example.user
```

2. To assign a password to the Linux *example.user* user:

```
# passwd example.user
Changing password for user example.user.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

3. Log out of your current session.

4. Log in as the Linux *example.user* user. The user's shell launches with the **staff_u** context.

Verification

1. When logged in as the *example.user* user, check the context of a Linux user:

```
$ id -Z
uid=1000(example.user) gid=1000(example.user) groups=1000(example.user)
context=staff_u:staff_r:staff_t:s0-s0:c0.c1023
```

Additional resources

- **pam_selinux(8)** man page.

3.5. CONFINING REGULAR USERS

You can confine all regular users on your system by mapping them to the **user_u** SELinux user.

By default, all Linux users in Red Hat Enterprise Linux, including users with administrative privileges, are mapped to the unconfined SELinux user **unconfined_u**. You can improve the security of the system by assigning users to SELinux confined users. This is useful to conform with the V-71971 Security Technical Implementation Guide.

Procedure

1. Display the list of SELinux login records. The list displays the mappings of Linux users to SELinux users:

```
# semanage login -l

Login Name    SELinux User  MLS/MCS Range   Service

__default__   unconfined_u  s0-s0:c0.c1023      *
root          unconfined_u  s0-s0:c0.c1023      *
```

2. Map the __default__ user, which represents all users without an explicit mapping, to the **user_u** SELinux user:

```
# semanage login -m -s user_u -r s0 __default__
```

Verification

1. Check that the __default__ user is mapped to the **user_u** SELinux user:

```
# semanage login -l

Login Name    SELinux User   MLS/MCS Range    Service

__default__   user_u         s0               *
root          unconfined_u   s0-s0:c0.c1023   *
```

2. Verify that the processes of a new user run in the **user_u:user_r:user_t:s0** SELinux context.

   a. Create a new user:

```
# adduser example.user
```

b. Define a password for *example.user*:

```
# passwd example.user
```

c. Log out as **root** and log in as the new user.

d. Show the security context for the user's ID:

```
[example.user@localhost ~]$ id -Z
user_u:user_r:user_t:s0
```

e. Show the security context of the user's current processes:

```
[example.user@localhost ~]$ ps axZ
LABEL                             PID TTY      STAT   TIME COMMAND
-                                   1 ?        Ss     0:05 /usr/lib/systemd/systemd --switched-root --
system --deserialize 18
-                                3729 ?        S      0:00 (sd-pam)
user_u:user_r:user_t:s0          3907 ?        Ss     0:00 /usr/lib/systemd/systemd --user
-                                3911 ?        S      0:00 (sd-pam)
user_u:user_r:user_t:s0          3918 ?        S      0:00 sshd: example.user@pts/0
user_u:user_r:user_t:s0          3922 pts/0    Ss     0:00 -bash
user_u:user_r:user_dbusd_t:s0    3969 ?        Ssl    0:00 /usr/bin/dbus-daemon --session --
address=systemd: --nofork --nopidfile --systemd-activation --syslog-only
user_u:user_r:user_t:s0          3971 pts/0    R+     0:00 ps axZ
```

3.6. CONFINING AN ADMINISTRATOR BY MAPPING TO SYSADM_U

You can confine a user with administrative privileges by mapping the user directly to the **sysadm_u** SELinux user. When the user logs in, the session runs in the **sysadm_u:sysadm_r:sysadm_t** SELinux context.

By default, all Linux users in Red Hat Enterprise Linux, including users with administrative privileges, are mapped to the unconfined SELinux user **unconfined_u**. You can improve the security of the system by assigning users to SELinux confined users. This is useful to conform with the V-71971 Security Technical Implementation Guide.

Prerequisites

- The **root** user runs unconfined. This is the Red Hat Enterprise Linux default.

Procedure

1. Optional: To allow **sysadm_u** users to connect to the system using SSH:

```
# setsebool -P ssh_sysadm_login on
```

2. Create a new user, add the user to the **wheel** user group, and map the user to the **sysadm_u** SELinux user:

```
# adduser -G wheel -Z sysadm_u example.user
```

3. Optional: Map an existing user to the **sysadm_u** SELinux user and add the user to the **wheel** user group:

```
# usermod -G wheel -Z sysadm_u example.user
```

Verification

1. Check that ***example.user*** is mapped to the **sysadm_u** SELinux user:

```
# semanage login -l | grep example.user
example.user     sysadm_u    s0-s0:c0.c1023   *
```

2. Log in as ***example.user***, for example, using SSH, and show the user's security context:

```
[example.user@localhost ~]$ id -Z
sysadm_u:sysadm_r:sysadm_t:s0-s0:c0.c1023
```

3. Switch to the **root** user:

```
$ sudo -i
[sudo] password for example.user:
```

4. Verify that the security context remains unchanged:

```
# id -Z
sysadm_u:sysadm_r:sysadm_t:s0-s0:c0.c1023
```

5. Try an administrative task, for example, restarting the **sshd** service:

```
# systemctl restart sshd
```

If there is no output, the command finished successfully.

If the command does not finish successfully, it prints the following message:

```
Failed to restart sshd.service: Access denied
See system logs and 'systemctl status sshd.service' for details.
```

3.7. CONFINING AN ADMINISTRATOR USING SUDO AND THE SYSADM_R ROLE

You can map a specific user with administrative privileges to the **staff_u** SELinux user, and configure **sudo** so that the user can gain the **sysadm_r** SELinux administrator role. This role allows the user to perform administrative tasks without SELinux denials. When the user logs in, the session runs in the **staff_u:staff_r:staff_t** SELinux context, but when the user enters a command using **sudo**, the session changes to the **staff_u:sysadm_r:sysadm_t** context.

By default, all Linux users in Red Hat Enterprise Linux, including users with administrative privileges, are mapped to the unconfined SELinux user **unconfined_u**. You can improve the security of the system by assigning users to SELinux confined users. This is useful to conform with the V-71971 Security Technical Implementation Guide.

Prerequisites

- The **root** user runs unconfined. This is the Red Hat Enterprise Linux default.

Procedure

1. Create a new user, add the user to the **wheel** user group, and map the user to the **staff_u** SELinux user:

```
# adduser -G wheel -Z staff_u example.user
```

2. Optional: Map an existing user to the **staff_u** SELinux user and add the user to the **wheel** user group:

```
# usermod -G wheel -Z staff_u example.user
```

3. To allow *example.user* to gain the SELinux administrator role, create a new file in the **/etc/sudoers.d/** directory, for example:

```
# visudo -f /etc/sudoers.d/example.user
```

4. Add the following line to the new file:

```
example.user ALL=(ALL) TYPE=sysadm_t ROLE=sysadm_r ALL
```

Verification

1. Check that ***example.user*** is mapped to the **staff_u** SELinux user:

```
# semanage login -l | grep example.user
example.user     staff_u    s0-s0:c0.c1023   *
```

2. Log in as *example.user*, for example, using SSH, and switch to the **root** user:

```
[example.user@localhost ~]$ sudo -i
[sudo] password for example.user:
```

3. Show the **root** security context:

```
# id -Z
staff_u:sysadm_r:sysadm_t:s0-s0:c0.c1023
```

4. Try an administrative task, for example, restarting the **sshd** service:

```
# systemctl restart sshd
```

If there is no output, the command finished successfully.

If the command does not finish successfully, it prints the following message:

```
Failed to restart sshd.service: Access denied
See system logs and 'systemctl status sshd.service' for details.
```

3.8. ADDITIONAL RESOURCES

- **unconfined_selinux(8)**, **user_selinux(8)**, **staff_selinux(8)**, and **sysadm_selinux(8)** man pages
- How to set up a system with SELinux confined users

CHAPTER 4. CONFIGURING SELINUX FOR APPLICATIONS AND SERVICES WITH NON-STANDARD CONFIGURATIONS

When SELinux is in enforcing mode, the default policy is the targeted policy. The following sections provide information on setting up and configuring the SELinux policy for various services after you change configuration defaults, such as ports, database locations, or file-system permissions for processes.

You learn to change SELinux types for non-standard ports, to identify and fix incorrect labels for changes of default directories, and to adjust the policy using SELinux booleans.

4.1. CUSTOMIZING THE SELINUX POLICY FOR THE APACHE HTTP SERVER IN A NON-STANDARD CONFIGURATION

You can configure the Apache HTTP server to listen on a different port and to provide content in a non-default directory. To prevent consequent SELinux denials, follow the steps in this procedure to adjust your system's SELinux policy.

Prerequisites

- The **httpd** package is installed and the Apache HTTP server is configured to listen on TCP port 3131 and to use the **/var/test_www/** directory instead of the default **/var/www/** directory.
- The **policycoreutils-python-utils** and **setroubleshoot-server** packages are installed on your system.

Procedure

1. Start the **httpd** service and check the status:

```
# systemctl start httpd
# systemctl status httpd
...
httpd[14523]: (13)Permission denied: AH00072: make_sock: could not bind to address
[::]:3131
...
systemd[1]: Failed to start The Apache HTTP Server.
...
```

2. The SELinux policy assumes that **httpd** runs on port 80:

```
# semanage port -l | grep http
http_cache_port_t              tcp      8080, 8118, 8123, 10001-10010
http_cache_port_t              udp      3130
http_port_t                    tcp      80, 81, 443, 488, 8008, 8009, 8443, 9000
pegasus_http_port_t            tcp      5988
pegasus_https_port_t           tcp      5989
```

3. Change the SELinux type of port 3131 to match port 80:

```
# semanage port -a -t http_port_t -p tcp 3131
```

4. Start **httpd** again:

```
# systemctl start httpd
```

5. However, the content remains inaccessible:

```
# wget localhost:3131/index.html
...
HTTP request sent, awaiting response... 403 Forbidden
...
```

Find the reason with the **sealert** tool:

```
# sealert -l "*"
...
SELinux is preventing httpd from getattr access on the file /var/test_www/html/index.html.
...
```

6. Compare SELinux types for the standard and the new path using the **matchpathcon** tool:

```
# matchpathcon /var/www/html /var/test_www/html
/var/www/html       system_u:object_r:httpd_sys_content_t:s0
/var/test_www/html  system_u:object_r:var_t:s0
```

7. Change the SELinux type of the new **/var/test_www/html/** content directory to the type of the default **/var/www/html** directory:

```
# semanage fcontext -a -e /var/www /var/test_www
```

8. Relabel the **/var** directory recursively:

```
# restorecon -Rv /var/
...
Relabeled /var/test_www/html from unconfined_u:object_r:var_t:s0 to
unconfined_u:object_r:httpd_sys_content_t:s0
Relabeled /var/test_www/html/index.html from unconfined_u:object_r:var_t:s0 to
unconfined_u:object_r:httpd_sys_content_t:s0
```

Verification

1. Check that the **httpd** service is running:

```
# systemctl status httpd
...
Active: active (running)
...
systemd[1]: Started The Apache HTTP Server.
httpd[14888]: Server configured, listening on: port 3131
...
```

2. Verify that the content provided by the Apache HTTP server is accessible:

```
# wget localhost:3131/index.html
...
HTTP request sent, awaiting response... 200 OK
```

```
Length: 0 [text/html]
Saving to: ‘index.html’
...
```

Additional resources

- The **semanage(8)**, **matchpathcon(8)**, and **sealert(8)** man pages.

4.2. ADJUSTING THE POLICY FOR SHARING NFS AND CIFS VOLUMES USING SELINUX BOOLEANS

You can change parts of SELinux policy at runtime using booleans, even without any knowledge of SELinux policy writing. This enables changes, such as allowing services access to NFS volumes, without reloading or recompiling SELinux policy. The following procedure demonstrates listing SELinux booleans and configuring them to achieve the required changes in the policy.

NFS mounts on the client side are labeled with a default context defined by a policy for NFS volumes. In RHEL, this default context uses the **nfs_t** type. Also, Samba shares mounted on the client side are labeled with a default context defined by the policy. This default context uses the **cifs_t** type. You can enable or disable booleans to control which services are allowed to access the **nfs_t** and **cifs_t** types.

To allow the Apache HTTP server service (**httpd**) to access and share NFS and CIFS volumes, perform the following steps:

Prerequisites

- Optionally, install the **selinux-policy-devel** package to obtain clearer and more detailed descriptions of SELinux booleans in the output of the **semanage boolean -l** command.

Procedure

1. Identify SELinux booleans relevant for NFS, CIFS, and Apache:

```
# semanage boolean -l | grep 'nfs\|cifs' | grep httpd
httpd_use_cifs                 (off  ,  off)  Allow httpd to access cifs file systems
httpd_use_nfs                  (off  ,  off)  Allow httpd to access nfs file systems
```

2. List the current state of the booleans:

```
$ getsebool -a | grep 'nfs\|cifs' | grep httpd
httpd_use_cifs --> off
httpd_use_nfs --> off
```

3. Enable the identified booleans:

```
# setsebool httpd_use_nfs on
# setsebool httpd_use_cifs on
```

NOTE

Use **setsebool** with the **-P** option to make the changes persistent across restarts. A **setsebool -P** command requires a rebuild of the entire policy, and it might take some time depending on your configuration.

Verification

1. Check that the booleans are **on**:

```
$ getsebool -a | grep 'nfs\|cifs' | grep httpd
httpd_use_cifs --> on
httpd_use_nfs --> on
```

Additional resources

- **semanage-boolean(8)**, **sepolicy-booleans(8)**, **getsebool(8)**, **setsebool(8)**, **booleans(5)**, and **booleans(8)** man pages

4.3. ADDITIONAL RESOURCES

- Troubleshooting problems related to SELinux

CHAPTER 5. TROUBLESHOOTING PROBLEMS RELATED TO SELINUX

If you plan to enable SELinux on systems where it has been previously disabled or if you run a service in a non-standard configuration, you might need to troubleshoot situations potentially blocked by SELinux. Note that in most cases, SELinux denials are signs of misconfiguration.

5.1. IDENTIFYING SELINUX DENIALS

Follow only the necessary steps from this procedure; in most cases, you need to perform just step 1.

Procedure

1. When your scenario is blocked by SELinux, the **/var/log/audit/audit.log** file is the first place to check for more information about a denial. To query Audit logs, use the **ausearch** tool. Because the SELinux decisions, such as allowing or disallowing access, are cached and this cache is known as the Access Vector Cache (AVC), use the **AVC** and **USER_AVC** values for the message type parameter, for example:

   ```
   # ausearch -m AVC,USER_AVC,SELINUX_ERR,USER_SELINUX_ERR -ts recent
   ```

   If there are no matches, check if the Audit daemon is running. If it does not, repeat the denied scenario after you start **auditd** and check the Audit log again.

2. In case **auditd** is running, but there are no matches in the output of **ausearch**, check messages provided by the **systemd** Journal:

   ```
   # journalctl -t setroubleshoot
   ```

3. If SELinux is active and the Audit daemon is not running on your system, then search for certain SELinux messages in the output of the **dmesg** command:

   ```
   # dmesg | grep -i -e type=1300 -e type=1400
   ```

4. Even after the previous three checks, it is still possible that you have not found anything. In this case, AVC denials can be silenced because of **dontaudit** rules.
   To temporarily disable **dontaudit** rules, allowing all denials to be logged:

   ```
   # semodule -DB
   ```

   After re-running your denied scenario and finding denial messages using the previous steps, the following command enables **dontaudit** rules in the policy again:

   ```
   # semodule -B
   ```

5. If you apply all four previous steps, and the problem still remains unidentified, consider if SELinux really blocks your scenario:

   - Switch to permissive mode:

```
# setenforce 0
$ getenforce
Permissive
```

- Repeat your scenario.

If the problem still occurs, something different than SELinux is blocking your scenario.

5.2. ANALYZING SELINUX DENIAL MESSAGES

After identifying that SELinux is blocking your scenario, you might need to analyze the root cause before you choose a fix.

Prerequisites

- The **policycoreutils-python-utils** and **setroubleshoot-server** packages are installed on your system.

Procedure

1. List more details about a logged denial using the **sealert** command, for example:

```
$ sealert -l "*"
SELinux is preventing /usr/bin/passwd from write access on the file
/root/test.

*****  Plugin leaks (86.2 confidence) suggests ******************************

If you want to ignore passwd trying to write access the test file,
because you believe it should not need this access.
Then you should report this as a bug.
You can generate a local policy module to dontaudit this access.
Do
# ausearch -x /usr/bin/passwd --raw | audit2allow -D -M my-passwd
# semodule -X 300 -i my-passwd.pp

*****  Plugin catchall (14.7 confidence) suggests **************************

...

Raw Audit Messages
type=AVC msg=audit(1553609555.619:127): avc:  denied  { write } for
pid=4097 comm="passwd" path="/root/test" dev="dm-0" ino=17142697
scontext=unconfined_u:unconfined_r:passwd_t:s0-s0:c0.c1023
tcontext=unconfined_u:object_r:admin_home_t:s0 tclass=file permissive=0

...

Hash: passwd,passwd_t,admin_home_t,file,write
```

2. If the output obtained in the previous step does not contain clear suggestions:

   - Enable full-path auditing to see full paths to accessed objects and to make additional Linux Audit event fields visible:

     -

```
# auditctl -w /etc/shadow -p w -k shadow-write
```

- Clear the **setroubleshoot** cache:

```
# rm -f /var/lib/setroubleshoot/setroubleshoot.xml
```

- Reproduce the problem.
- Repeat step 1.
  After you finish the process, disable full-path auditing:

```
# auditctl -W /etc/shadow -p w -k shadow-write
```

3. If **sealert** returns only **catchall** suggestions or suggests adding a new rule using the **audit2allow** tool, match your problem with examples listed and explained in SELinux denials in the Audit log.

Additional resources

- **sealert(8)** man page

5.3. FIXING ANALYZED SELINUX DENIALS

In most cases, suggestions provided by the **sealert** tool give you the right guidance about how to fix problems related to the SELinux policy. See Analyzing SELinux denial messages for information how to use **sealert** to analyze SELinux denials.

Be careful when the tool suggests using the **audit2allow** tool for configuration changes. You should not use **audit2allow** to generate a local policy module as your first option when you see an SELinux denial. Troubleshooting should start with a check if there is a labeling problem. The second most often case is that you have changed a process configuration, and you forgot to tell SELinux about it.

Labeling problems

A common cause of labeling problems is when a non-standard directory is used for a service. For example, instead of using **/var/www/html/** for a website, an administrator might want to use **/srv/myweb/**. On Red Hat Enterprise Linux, the **/srv** directory is labeled with the **var_t** type. Files and directories created in **/srv** inherit this type. Also, newly-created objects in top-level directories, such as **/myserver**, can be labeled with the **default_t** type. SELinux prevents the Apache HTTP Server (**httpd**) from accessing both of these types. To allow access, SELinux must know that the files in **/srv/myweb/** are to be accessible by **httpd**:

```
# semanage fcontext -a -t httpd_sys_content_t "/srv/myweb(/.*)?"
```

This **semanage** command adds the context for the **/srv/myweb/** directory and all files and directories under it to the SELinux file-context configuration. The **semanage** utility does not change the context. As root, use the **restorecon** utility to apply the changes:

```
# restorecon -R -v /srv/myweb
```

Incorrect context

The **matchpathcon** utility checks the context of a file path and compares it to the default label for that path. The following example demonstrates the use of **matchpathcon** on a directory that contains incorrectly labeled files:

```
$ matchpathcon -V /var/www/html/*
/var/www/html/index.html has context unconfined_u:object_r:user_home_t:s0, should be
system_u:object_r:httpd_sys_content_t:s0
/var/www/html/page1.html has context unconfined_u:object_r:user_home_t:s0, should be
system_u:object_r:httpd_sys_content_t:s0
```

In this example, the **index.html** and **page1.html** files are labeled with the **user_home_t** type. This type is used for files in user home directories. Using the **mv** command to move files from your home directory may result in files being labeled with the **user_home_t** type. This type should not exist outside of home directories. Use the **restorecon** utility to restore such files to their correct type:

```
# restorecon -v /var/www/html/index.html
restorecon reset /var/www/html/index.html context unconfined_u:object_r:user_home_t:s0-
>system_u:object_r:httpd_sys_content_t:s0
```

To restore the context for all files under a directory, use the **-R** option:

```
# restorecon -R -v /var/www/html/
restorecon reset /var/www/html/page1.html context unconfined_u:object_r:samba_share_t:s0-
>system_u:object_r:httpd_sys_content_t:s0
restorecon reset /var/www/html/index.html context unconfined_u:object_r:samba_share_t:s0-
>system_u:object_r:httpd_sys_content_t:s0
```

Confined applications configured in non-standard ways

Services can be run in a variety of ways. To account for that, you need to specify how you run your services. You can achieve this through SELinux booleans that allow parts of SELinux policy to be changed at runtime. This enables changes, such as allowing services access to NFS volumes, without reloading or recompiling SELinux policy. Also, running services on non-default port numbers requires policy configuration to be updated using the **semanage** command.

For example, to allow the Apache HTTP Server to communicate with MariaDB, enable the **httpd_can_network_connect_db** boolean:

```
# setsebool -P httpd_can_network_connect_db on
```

Note that the **-P** option makes the setting persistent across reboots of the system.

If access is denied for a particular service, use the **getsebool** and **grep** utilities to see if any booleans are available to allow access. For example, use the **getsebool -a | grep ftp** command to search for FTP related booleans:

```
$ getsebool -a | grep ftp
ftpd_anon_write --> off
ftpd_full_access --> off
ftpd_use_cifs --> off
ftpd_use_nfs --> off

ftpd_connect_db --> off
httpd_enable_ftp_server --> off
tftp_anon_write --> off
```

To get a list of booleans and to find out if they are enabled or disabled, use the **getsebool -a** command. To get a list of booleans including their meaning, and to find out if they are enabled or disabled, install the **selinux-policy-devel** package and use the **semanage boolean -l** command as root.

Port numbers

Depending on policy configuration, services can only be allowed to run on certain port numbers. Attempting to change the port a service runs on without changing policy may result in the service failing to start. For example, run the **semanage port -l | grep http** command as root to list **http** related ports:

```
# semanage port -l | grep http
http_cache_port_t              tcp      3128, 8080, 8118
http_cache_port_t              udp      3130
http_port_t                    tcp      80, 443, 488, 8008, 8009, 8443
pegasus_http_port_t            tcp      5988
pegasus_https_port_t           tcp      5989
```

The **http_port_t** port type defines the ports Apache HTTP Server can listen on, which in this case, are TCP ports 80, 443, 488, 8008, 8009, and 8443. If an administrator configures **httpd.conf** so that **httpd** listens on port 9876 (**Listen 9876**), but policy is not updated to reflect this, the following command fails:

```
# systemctl start httpd.service
Job for httpd.service failed. See 'systemctl status httpd.service' and 'journalctl -xn' for details.

# systemctl status httpd.service
httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled)
   Active: failed (Result: exit-code) since Thu 2013-08-15 09:57:05 CEST; 59s ago
  Process: 16874 ExecStop=/usr/sbin/httpd $OPTIONS -k graceful-stop (code=exited,
status=0/SUCCESS)
  Process: 16870 ExecStart=/usr/sbin/httpd $OPTIONS -DFOREGROUND (code=exited,
status=1/FAILURE)
```

An SELinux denial message similar to the following is logged to **/var/log/audit/audit.log**:

```
type=AVC msg=audit(1225948455.061:294): avc:  denied  { name_bind } for  pid=4997
comm="httpd" src=9876 scontext=unconfined_u:system_r:httpd_t:s0
tcontext=system_u:object_r:port_t:s0 tclass=tcp_socket
```

To allow **httpd** to listen on a port that is not listed for the **http_port_t** port type, use the **semanage port** command to assign a different label to the port:

```
# semanage port -a -t http_port_t -p tcp 9876
```

The **-a** option adds a new record; the **-t** option defines a type; and the **-p** option defines a protocol. The last argument is the port number to add.

Corner cases, evolving or broken applications, and compromised systems

Applications may contain bugs, causing SELinux to deny access. Also, SELinux rules are evolving – SELinux may not have seen an application running in a certain way, possibly causing it to deny access, even though the application is working as expected. For example, if a new version of PostgreSQL is released, it may perform actions the current policy does not account for, causing access to be denied, even though access should be allowed.

For these situations, after access is denied, use the **audit2allow** utility to create a custom policy module

to allow access. You can report missing rules in the SELinux policy in **Red Hat Bugzilla**. For Red Hat Enterprise Linux 8, create bugs against the **Red Hat Enterprise Linux 8** product, and select the **selinux-policy** component. Include the output of the **audit2allow -w -a** and **audit2allow -a** commands in such bug reports.

If an application asks for major security privileges, it could be a signal that the application is compromised. Use intrusion detection tools to inspect such suspicious behavior.

The **Solution Engine** on the Red Hat Customer Portal can also provide guidance in the form of an article containing a possible solution for the same or very similar problem you have. Select the relevant product and version and use SELinux-related keywords, such as *selinux* or *avc*, together with the name of your blocked service or application, for example: **selinux samba**.

5.4. SELINUX DENIALS IN THE AUDIT LOG

The Linux Audit system stores log entries in the **/var/log/audit/audit.log** file by default.

To list only SELinux-related records, use the **ausearch** command with the message type parameter set to **AVC** and **AVC_USER** at a minimum, for example:

```
# ausearch -m AVC,USER_AVC,SELINUX_ERR,USER_SELINUX_ERR
```

An SELinux denial entry in the Audit log file can look as follows:

```
type=AVC msg=audit(1395177286.929:1638): avc:  denied  { read } for  pid=6591 comm="httpd"
name="webpages" dev="0:37" ino=2112 scontext=system_u:system_r:httpd_t:s0
tcontext=system_u:object_r:nfs_t:s0 tclass=dir
```

The most important parts of this entry are:

- **avc: denied** - the action performed by SELinux and recorded in Access Vector Cache (AVC)
- **{ read }** - the denied action
- **pid=6591** - the process identifier of the subject that tried to perform the denied action
- **comm="httpd"** - the name of the command that was used to invoke the analyzed process
- **httpd_t** - the SELinux type of the process
- **nfs_t** - the SELinux type of the object affected by the process action
- **tclass=dir** - the target object class

The previous log entry can be translated to:

*SELinux denied the **httpd** process with PID 6591 and the **httpd_t** type to read from a directory with the **nfs_t** type.*

The following SELinux denial message occurs when the Apache HTTP Server attempts to access a directory labeled with a type for the Samba suite:

```
type=AVC msg=audit(1226874073.147:96): avc:  denied  { getattr } for  pid=2465 comm="httpd"
path="/var/www/html/file1" dev=dm-0 ino=284133 scontext=unconfined_u:system_r:httpd_t:s0
tcontext=unconfined_u:object_r:samba_share_t:s0 tclass=file
```

- **{ getattr }** - the **getattr** entry indicates the source process was trying to read the target file's status information. This occurs before reading files. SELinux denies this action because the process accesses the file and it does not have an appropriate label. Commonly seen permissions include **getattr**, **read**, and **write**.

- **path="/var/www/html/file1"** - the path to the object (target) the process attempted to access.

- **scontext="unconfined_u:system_r:httpd_t:s0"** - the SELinux context of the process (source) that attempted the denied action. In this case, it is the SELinux context of the Apache HTTP Server, which is running with the **httpd_t** type.

- **tcontext="unconfined_u:object_r:samba_share_t:s0"** - the SELinux context of the object (target) the process attempted to access. In this case, it is the SELinux context of **file1**.

This SELinux denial can be translated to:

*SELinux denied the **httpd** process with PID 2465 to access the **/var/www/html/file1** file with the **samba_share_t** type, which is not accessible to processes running in the **httpd_t** domain unless configured otherwise.*

Additional resources

- **auditd(8)** and **ausearch(8)** man pages

5.5. ADDITIONAL RESOURCES

- Basic SELinux Troubleshooting in CLI

- What is SELinux trying to tell me? The 4 key causes of SELinux errors

CHAPTER 6. USING MULTI-LEVEL SECURITY (MLS)

The Multi-Level Security (MLS) policy uses *levels* of clearance as originally designed by the US defense community. MLS meets a very narrow set of security requirements based on information management in rigidly controlled environments such as the military.

Using MLS is complex and does not map well to general use-case scenarios.

6.1. MULTI-LEVEL SECURITY (MLS)

The Multi-Level Security (MLS) technology classifies data in a hierarchical classification using information security levels, for example:

- [lowest] Unclassified
- [low] Confidential
- [high] Secret
- [highest] Top secret

By default, the MLS SELinux policy uses 16 sensitivity levels:

- **s0** is the least sensitive.
- **s15** is the most sensitive.

MLS uses specific terminology to address sensitivity levels:

- Users and processes are called **subjects**, whose sensitivity level is called **clearance**.
- Files, devices, and other passive components of the system are called **objects**, whose sensitivity level is called **classification**.

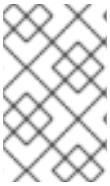
To implement MLS, SELinux uses the **Bell-La Padula Model** (BLP) model. This model specifies how information can flow within the system based on labels attached to each subject and object.

The basic principle of BLP is "**No read up, no write down.**" This means that users can only read files at their own sensitivity level and lower, and data can flow only from lower levels to higher levels, and never the reverse.

The MLS SELinux policy, which is the implementation of MLS on RHEL, applies a modified principle called **Bell-La Padula with write equality**. This means that users can read files at their own sensitivity level and lower, but can write only at exactly their own level. This prevents, for example, low-clearance users from writing content into top-secret files.

For example, by default, a user with clearance level **s2**:

- Can read files with sensitivity levels **s0**, **s1**, and **s2**.
- Cannot read files with sensitivity level **s3** and higher.
- Can modify files with sensitivity level of exactly **s2**.
- Cannot modify files with sensitivity level different than **s2**.



NOTE

Security administrators may adjust this behavior by modifying the system's SELinux policy. For example, they can allow users to modify files at lower levels, which increases the file's sensitivity level to the user's clearance level.

In practice, users are typically assigned to a range of clearance levels, for example **s1-s2**. A user can read files with sensitivity levels lower than the user's maximum level, and write to any files within that range.

For example, by default, a user with a clearance range **s1-s2**:

- Can read files with sensitivity levels **s0** and **s1**.
- Cannot read files with sensitivity level **s2** and higher.
- Can modify files with sensitivity level **s1**.
- Cannot modify files with sensitivity level different than **s1**.
- Can change own clearance level to **s2**.

The security context for a non-privileged user in an MLS environment is, for example:

```
user_u:user_r:user_t:s1
```

Where:

user_u
is the SELinux user.

user_r
is the SELinux role.

user_t
is the SELinux type.

s1
is the range of MLS sensitivity levels.

The system always combines MLS access rules with conventional file access permissions. For example, if a user with a security level of "Secret" uses Discretionary Access Control (DAC) to block access to a file by other users, even "Top Secret" users cannot access that file. A high security clearance does not automatically permit a user to browse the entire file system.

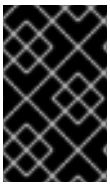
Users with top-level clearances do not automatically acquire administrative rights on multi-level systems. While they may have access to all sensitive information on the system, this is different from having administrative rights.

In addition, administrative rights do not provide access to sensitive information. For example, even when someone logs in as **root**, they still cannot read top-secret information.

You can further adjust access within an MLS system by using categories. With Multi-Category Security (MCS), you can define categories such as projects or departments, and users will only be allowed to access files in the categories to which they are assigned. For additional information, see Using Multi-Category Security (MCS) for data confidentiality .

6.2. SELINUX ROLES IN MLS

The SELinux policy maps each Linux user to an SELinux user. This allows Linux users to inherit the restrictions of SELinux users.



IMPORTANT

The MLS policy does not contain the **unconfined** module, including unconfined users, types, and roles. As a result, users that would be unconfined, including **root**, cannot access every object and perform every action they could in the targeted policy.

You can customize the permissions for confined users in your SELinux policy according to specific needs by adjusting the booleans in policy. You can determine the current state of these booleans by using the **semanage boolean -l** command. To list all SELinux users, their SELinux roles, and MLS/MCS levels and ranges, use the **semanage user -l** command as **root**.

Table 6.1. Roles of SELinux users in MLS

User	Default role	Additional roles
guest_u	**guest_r**	
xguest_u	**xguest_r**	
user_u	**user_r**	
staff_u	**staff_r**	**auditadm_r**
		secadm_r
		sysadm_r
		staff_r
sysadm_u	**sysadm_r**	
root	**staff_r**	**auditadm_r**
		secadm_r
		sysadm_r
		system_r
system_u	**system_r**	

Note that **system_u** is a special user identity for system processes and objects, and **system_r** is the associated role. Administrators must never associate this **system_u** user and the **system_r** role to a Linux user. Also, **unconfined_u** and **root** are unconfined users. For these reasons, the roles associated

to these SELinux users are not included in the following table Types and access of SELinux roles.

Each SELinux role corresponds to an SELinux type and provides specific access rights.

Table 6.2. Types and access of SELinux roles in MLS

Role	Type	Login using X Window System	**su** and **sudo**	Execute in home directory and **/tmp** (default)	Networking
guest_r	**guest_t**	no	no	yes	no
xguest_r	**xguest_t**	yes	no	yes	web browsers only (Firefox, GNOME Web)
user_r	**user_t**	yes	no	yes	yes
staff_r	**staff_t**	yes	only **sudo**	yes	yes
auditadm_r	**auditadm_t**		yes	yes	yes
secadm_r	**secadm_t**		yes	yes	yes
sysadm_r	**sysadm_t**	only when the **xdm_sysadm_login** boolean is **on**	yes	yes	yes

- By default, the **sysadm_r** role has the rights of the **secadm_r** role, which means a user with the **sysadm_r** role can manage the security policy. If this does not correspond to your use case, you can separate the two roles by disabling the **sysadm_secadm** module in the policy. For additional information, see Separating system administration from security administration in MLS
- Non-login roles **dbadm_r**, **logadm_r**, and **webadm_r** can be used for a subset of administrative tasks. By default, these roles are not associated with any SELinux user.

6.3. SWITCHING THE SELINUX POLICY TO MLS

Use the following steps to switch the SELinux policy from targeted to Multi-Level Security (MLS).

IMPORTANT

Red Hat does not recommend to use the MLS policy on a system that is running the X Window System. Furthermore, when you relabel the file system with MLS labels, the system may prevent confined domains from access, which prevents your system from starting correctly. Therefore ensure that you switch SELinux to permissive mode before you relabel the files. On most systems, you see a lot of SELinux denials after switching to MLS, and many of them are not trivial to fix.

Procedure

1. Install the **selinux-policy-mls** package:

```
# yum install selinux-policy-mls
```

2. Open the **/etc/selinux/config** file in a text editor of your choice, for example:

```
# vi /etc/selinux/config
```

3. Change SELinux mode from enforcing to permissive and switch from the targeted policy to MLS:

```
SELINUX=permissive
SELINUXTYPE=mls
```

   Save the changes, and quit the editor.

4. Before you enable the MLS policy, you must relabel each file on the file system with an MLS label:

```
# fixfiles -F onboot
System will relabel on next boot
```

5. Restart the system:

```
# reboot
```

6. Check for SELinux denials:

```
# ausearch -m AVC,USER_AVC,SELINUX_ERR,USER_SELINUX_ERR -ts recent -i
```

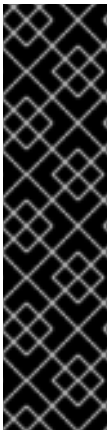
   Because the previous command does not cover all scenarios, see Troubleshooting problems related to SELinux for guidance on identifying, analyzing, and fixing SELinux denials.

7. After you ensure that there are no problems related to SELinux on your system, switch SELinux back to enforcing mode by changing the corresponding option in **/etc/selinux/config**:

```
SELINUX=enforcing
```

8. Restart the system:

```
# reboot
```



IMPORTANT

If your system does not start or you are not able to log in after you switch to MLS, add the **enforcing=0** parameter to your kernel command line. See Changing SELinux modes at boot time for more information.

Also note that in MLS, SSH logins as the **root** user mapped to the **sysadm_r** SELinux role differ from logging in as **root** in **staff_r**. Before you start your system in MLS for the first time, consider allowing SSH logins as **sysadm_r** by setting the **ssh_sysadm_login** SELinux boolean to **1**. To enable **ssh_sysadm_login** later, already in MLS, you must log in as **root** in **staff_r**, switch to **root** in **sysadm_r** using the **newrole -r sysadm_r** command, and then set the boolean to **1**.

Verification

1. Verify that SELinux runs in enforcing mode:

```
# getenforce
Enforcing
```

2. Check that the status of SELinux returns the **mls** value:

```
# sestatus | grep mls
Loaded policy name:             mls
```

Additional resources

- The **fixfiles(8)**, **setsebool(8)**, and **ssh_selinux(8)** man pages.

6.4. ESTABLISHING USER CLEARANCE IN MLS

After you switch SELinux policy to MLS, you must assign security clearance levels to users by mapping them to confined SELinux users. By default, a user with a given security clearance:

- Cannot read objects that have a higher sensitivity level.
- Cannot write to objects at a different sensitivity level.

Prerequisites

- The SELinux policy is set to **mls**.
- The SELinux mode is set to **enforcing**.
- The **policycoreutils-python-utils** package is installed.
- A user assigned to an SELinux confined user:
  - For a non-privileged user, assigned to **user_u** (*example_user* in the following procedure).
  - For a privileged user, assigned to **staff_u** (*staff* in the following procedure) .

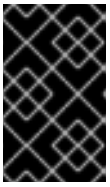
NOTE

Make sure that the users have been created when the MLS policy was active. Users created in other SELinux policies cannot be used in MLS.

Procedure

1. Optional: To prevent adding errors to your SELinux policy, switch to the **permissive** SELinux mode, which facilitates troubleshooting:

```
# setenforce 0
```



IMPORTANT

In permissive mode, SELinux does not enforce the active policy but only logs Access Vector Cache (AVC) messages, which can be then used for troubleshooting and debugging.

2. Define a clearance range for the **staff_u** SELinux user. For example, this command sets the clearance range from **s1** to **s15** with **s1** being the default clearance level:

```
# semanage user -m -L s1 -r s1-s15 _staff_u
```

3. Generate SELinux file context configuration entries for user home directories:

```
# genhomedircon
```

4. Restore file security contexts to default:

```
# restorecon -R -F -v /home/
Relabeled /home/staff from staff_u:object_r:user_home_dir_t:s0 to
staff_u:object_r:user_home_dir_t:s1
Relabeled /home/staff/.bash_logout from staff_u:object_r:user_home_t:s0 to
staff_u:object_r:user_home_t:s1
Relabeled /home/staff/.bash_profile from staff_u:object_r:user_home_t:s0 to
staff_u:object_r:user_home_t:s1
Relabeled /home/staff/.bashrc from staff_u:object_r:user_home_t:s0 to
staff_u:object_r:user_home_t:s1
```

5. Assign a clearance level to the user:

```
# semanage login -m -r s1 example_user
```

Where ***s1*** is the clearance level assigned to the user.

6. Relabel the user's home directory to the user's clearance level:

```
# chcon -R -l s1 /home/example_user
```

7. Optional: If you previously switched to the **permissive** SELinux mode, and after you verify that everything works as expected, switch back to the **enforcing** SELinux mode:

```
# setenforce 1
```

-

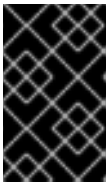
Verification steps

1. Verify that the user is mapped to the correct SELinux user and has the correct clearance level assigned:

```
# semanage login -l
Login Name      SELinux User        MLS/MCS Range        Service
__default__     user_u              s0-s0                *
example_user    user_u              s1                   *
...
```

2. Log in as the user within MLS.

3. Verify that the user's security level works correctly:



IMPORTANT

The files you use for verification should not contain any sensitive information in case the configuration is incorrect and the user actually can access the files without authorization.

   a. Verify that the user cannot read a file with a higher-level sensitivity.

   b. Verify that the user can write to a file with the same sensitivity.

   c. Verify that the user can read a file with a lower-level sensitivity.

Additional resources

- Section 6.3, "Switching the SELinux policy to MLS" .
- Section 3.4, "Adding a new user as an SELinux-confined user" .
- Chapter 2, *Changing SELinux states and modes* .
- Chapter 5, *Troubleshooting problems related to SELinux* .
- The Basic SELinux Troubleshooting in CLI Knowledgebase article.

6.5. CHANGING A USER'S CLEARANCE LEVEL WITHIN THE DEFINED SECURITY RANGE IN MLS

As a user in Multi-Level Security (MLS), you can change your current clearance level within the range the administrator assigned to you. You can never exceed the upper limit of your range or reduce your level below the lower limit of your range. This allows you, for example, to modify lower-sensitivity files without increasing their sensitivity level to your highest clearance level.

For example, as a user assigned to range **s1-s3**:

- You can switch to levels **s1**, **s2**, and **s3**.
- You can switch to ranges **s1-s2**, and **s2-s3**.

- You cannot switch to ranges **s0-s3** or **s1-s4**.

NOTE

Switching to a different level opens a new shell with the different clearance. This means you cannot return to your original clearance level in the same way as decreasing it. However, you can always return to the previous shell by entering **exit**.

Prerequisites

- SELinux policy is set to **mls**.
- SELinux mode is set to **enforcing**.
- You can log in as a user assigned to a range of MLS clearance levels.

Procedure

1. Log in as the user from a secure terminal.

NOTE

Secure terminals are defined in the **/etc/selinux/mls/contexts/securetty_types** file. By default, the console is a secure terminal, but SSH is not.

2. Check the current user's security context:

```
$ id -Z
user_u:user_r:user_t:s0-s2
```

In this example, the user is assigned to the **user_u** SELinux user, **user_r** role, **user_t** type, and the MLS security range **s0-s2**.

3. Check the current user's security context:

```
$ id -Z
user_u:user_r:user_t:s1-s2
```

4. Switch to a different security clearance range within the user's clearance range:

```
$ newrole -l s1
```

You can switch to any range whose maximum is lower or equal to your assigned range. Entering a single-level range changes the lower limit of the assigned range. For example, entering **newrole -l s1** as a user with a **s0-s2** range is equivalent to entering **newrole -l s1-s2**.

Verification

1. Display the current user's security context:

```
$ id -Z
user_u:user_r:user_t:s1-s2
```

2. Return to the previous shell with the original range by terminating the current shell:

```
$ exit
```

Additional resources

- Section 6.4, "Establishing user clearance in MLS"
- **newrole(1)** man page
- **securetty_types(5)** man page

6.6. INCREASING FILE SENSITIVITY LEVELS IN MLS

By default, Multi-Level Security (MLS) users cannot increase file sensitivity levels. However, the security administrator (**secadm_r**) can change this default behavior to allow users to increase the sensitivity of files by adding the local module **mlsfilewrite** to the system's SELinux policy. Then, users assigned to the SELinux type defined in the policy module can increase file classification levels by modifying the file. Any time a user modifies a file, the file's sensitivity level increases to the lower value of the user's current security range.

NOTE

The security administrator, when logged in as a user assigned to the **secadm_r** role, can change the security levels of files by using the **chcon -l *s0 /path/to/file*** command. For more information, see Section 6.7, "Changing file sensitivity in MLS"

Prerequisites

- The SELinux policy is set to **mls**.
- The SELinux mode is set to **enforcing**.
- The **policycoreutils-python-utils** package is installed.
- The **mlsfilewrite** local module is installed in the SELinux MLS policy.
- You are logged in as a user in MLS which is:
  - Assigned to a defined security range. This example shows a user with a security range **s0-s2**.
  - Assigned to the same SELinux type defined in the **mlsfilewrite** module. This example requires the **(typeattributeset mlsfilewrite (user_t))** module.

Procedure

1. Optional: Display the security context of the current user:

```
$ id -Z
user_u:user_r:user_t:s0-s2
```

2. Change the lower level of the user's MLS clearance range to the level which you want to assign to the file:

```
$ newrole -l s1-s2
```

3. Optional: Display the security context of the current user:

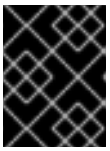
```
$ id -Z
user_u:user_r:user_t:s1-s2
```

4. Optional: Display the security context of the file:

```
$ ls -Z /path/to/file
user_u:object_r:user_home_t:s0 /path/to/file
```

5. Change the file's sensitivity level to the lower level of the user's clearance range by modifying the file:

```
$ touch /path/to/file
```



IMPORTANT

The classification level reverts to the default value if the **restorecon** command is used on the system.

6. Optional: Exit the shell to return to the user's previous security range:

```
$ exit
```

Verification

- Display the security context of the file:

```
$ ls -Z /path/to/file
user_u:object_r:user_home_t:s1 /path/to/file
```

Additional resources

- Section 6.10, "Allowing MLS users to edit files on lower levels" .

6.7. CHANGING FILE SENSITIVITY IN MLS

In the MLS SELinux policy, users can only modify files at their own sensitivity level. This is intended to prevent any highly sensitive information to be exposed to users at lower clearance levels, and also prevent low-clearance users creating high-sensitivity documents. Administrators, however, can manually increase a file's classification, for example for the file to be processed at the higher level.

Prerequisites

- SELinux policy is set to **mls**.
- SELinux mode is set to enforcing.
- You have security administration rights, which means that you are assigned to either:

- The **secadm_r** role.
- If the **sysadm_secadm** module is enabled, to the **sysadm_r** role. The **sysadm_secadm** module is enabled by default.

- The **policycoreutils-python-utils** package is installed.
- A user assigned to any clearance level. For additional information, see Establishing user clearance levels in MLS .
  In this example, ***User1*** has clearance level **s1**.
- A file with a classification level assigned and to which you have access.
  In this example, ***/path/to/file*** has classification level **s1**.

Procedure

1. Check the file's classification level:

```
# ls -lZ /path/to/file
-rw-r-----. 1 User1 User1 user_u:object_r:user_home_t:s1 0 12. Feb 10:43 /path/to/file
```

2. Change the file's default classification level:

```
# semanage fcontext -a -r s2 /path/to/file
```

3. Force the relabeling of the file's SELinux context:

```
# restorecon -F -v /path/to/file
Relabeled /path/to/file from user_u:object_r:user_home_t:s1 to
user_u:object_r:user_home_t:s2
```

Verification

1. Check the file's classification level:

```
# ls -lZ /path/to/file
-rw-r-----. 1 User1 User1 user_u:object_r:user_home_t:s2 0 12. Feb 10:53 /path/to/file
```

2. Optional: Verify that the lower-clearance user cannot read the file:

```
$ cat /path/to/file
cat: file: Permission denied
```

Additional resources

- Section 6.4, "Establishing user clearance in MLS" .

6.8. SEPARATING SYSTEM ADMINISTRATION FROM SECURITY ADMINISTRATION IN MLS

By default, the **sysadm_r** role has the rights of the **secadm_r** role, which means a user with the **sysadm_r** role can manage the security policy. If you need more control over security authorizations, you

can separate system administration from security administration by assigning a Linux user to the **secadm_r** role and disabling the **sysadm_secadm** module in the SELinux policy.

Prerequisites

- The SELinux policy is set to **mls**.
- The SELinux mode is set to **enforcing**.
- The **policycoreutils-python-utils** package is installed.
- A Linux user which will be assigned to the **secadm_r** role:
  - The user is assigned to the **staff_u** SELinux user
  - A password for this user has been defined.

WARNING

Make sure you can log in as the user which will be assigned to the **secadm** role. If not, you can prevent any future modifications of the system's SELinux policy.

Procedure

1. Create a new **sudoers** file in the **/etc/sudoers.d** directory for the user:

```
# visudo -f /etc/sudoers.d/<sec_adm_user>
```

   To keep the **sudoers** files organized, replace ***<sec_adm_user>*** with the Linux user which will be assigned to the **secadm** role.

2. Add the following content into the **/etc/sudoers.d/*<sec_adm_user>*** file:

```
<sec_adm_user> ALL=(ALL) TYPE=secadm_t ROLE=secadm_r ALL
```

   This line authorizes ***<secadmuser>*** on all hosts to perform all commands, and maps the user to the **secadm** SELinux type and role by default.

3. Log in as the *<sec_adm_user>* user:

NOTE

To make sure that the SELinux context (which consists of SELinux user, role, and type) is changed, log in using **ssh**, the console, or **xdm**. Other ways, such as **su** and **sudo**, cannot change the entire SELinux context.

4. Verify the user's security context:

```
$ id
uid=1000(<sec_adm_user>) gid=1000(<sec_adm_user>) groups=1000(<sec_adm_user>)
context=staff_u:staff_r:staff_t:s0-s15:c0.c1023
```

5. Run the interactive shell for the root user:

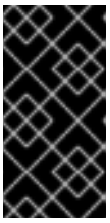
```
$ sudo -i
[sudo] password for <sec_adm_user>:
```

6. Verify the current user's security context:

```
# id
uid=0(root) gid=0(root) groups=0(root) context=staff_u:secadm_r:secadm_t:s0-s15:c0.c1023
```

7. Disable the **sysadm_secadm** module from the policy:

```
# semodule -d sysadm_secadm
```



IMPORTANT

Use the **semodule -d** command instead of removing the system policy module by using the **semodule -r** command. The **semodule -r** command deletes the module from your system's storage, which means it cannot be loaded again without reinstalling the **selinux-policy-mls** package.

Verification

1. As the user assigned to the **secadm** role, and in the interactive shell for the root user, verify that you can access the security policy data:

```
# seinfo -xt secadm_t

Types: 1
   type secadm_t, can_relabelto_shadow_passwords, (...) userdomain;
```

2. Log out from the root shell:

```
# logout
```

3. Log out from the ***<sec_adm_user>*** user:

```
$ logout
Connection to localhost closed.
```

4. Display the current security context:

```
# id
uid=0(root) gid=0(root) groups=0(root) context=root:sysadm_r:sysadm_t:s0-s15:c0.c1023
```

5. Attempt to enable the **sysadm_secadm** module. The command should fail:

```
# semodule -e sysadm_secadm
```

```
SELinux:  Could not load policy file /etc/selinux/mls/policy/policy.31:  Permission denied
/sbin/load_policy:  Can't load policy:  Permission denied
libsemanage.semanage_reload_policy: load_policy returned error code 2. (No such file or
directory).
SELinux:  Could not load policy file /etc/selinux/mls/policy/policy.31:  Permission denied
/sbin/load_policy:  Can't load policy:  Permission denied
libsemanage.semanage_reload_policy: load_policy returned error code 2. (No such file or
directory).
semodule:  Failed!
```

6. Attempt to display the details about the **sysadm_t** SELinux type. The command should fail:

```
# seinfo -xt sysadm_t
[Errno 13] Permission denied: '/sys/fs/selinux/policy'
```

6.9. DEFINING A SECURE TERMINAL IN MLS

The SELinux policy checks the type of the terminal from which a user is connected, and allows running of certain SELinux applications, for example **newrole**, only from secure terminals. Attempting this from a non-secure terminal produces an error: **Error: you are not allowed to change levels on a non secure terminal;**.

The **/etc/selinux/mls/contexts/securetty_types** file defines secure terminals for the Multi-Level Security (MLS) policy.

Default contents of the file:

```
console_device_t
sysadm_tty_device_t
user_tty_device_t
staff_tty_device_t
auditadm_tty_device_t
secureadm_tty_device_t
```

WARNING

Adding terminal types to the list of secure terminals can expose your system to security risks.

Prerequisites

- SELinux policy is set to **mls**.
- You are connected from an already secure terminal, or SELinux is in permissive mode.
- You have security administration rights, which means that you are assigned to either:
  - The **secadm_r** role.

- If the **sysadm_secadm** module is enabled, to the **sysadm_r** role. The **sysadm_secadm** module is enabled by default.

- The **policycoreutils-python-utils** package is installed.

Procedure

1. Determine the current terminal type:

```
# ls -Z `tty`
root:object_r:user_devpts_t:s0 /dev/pts/0
```

   In this example output, **user_devpts_t** is the current terminal type.

2. Add the relevant SELinux type on a new line in the **/etc/selinux/mls/contexts/securetty_types** file.

3. Optional: Switch SELinux to enforcing mode:

```
# setenforce 1
```

Verification

- Log in from the previously insecure terminal you have added to the **/etc/selinux/mls/contexts/securetty_types** file.

Additional resources

- **securetty_types(5)** man page

6.10. ALLOWING MLS USERS TO EDIT FILES ON LOWER LEVELS

By default, MLS users cannot write to files which have a sensitivity level below the lower value of the clearance range. If your scenario requires allowing users to edit files on lower levels, you can do so by creating a local SELinux module. However, writing to a file will increase its sensitivity level to the lower value of the user's current range.

Prerequisites

- The SELinux policy is set to **mls**.

- The SELinux mode is set to **enforcing**.

- The **policycoreutils-python-utils** package is installed.

- The **setools-console** and **audit** packages for verification.

Procedure

1. Optional: Switch to permissive mode for easier troubleshooting.

```
# setenforce 0
```

2. Open a new **.cil** file with a text editor, for example **~/local_mlsfilewrite.cil**, and insert the following custom rule:

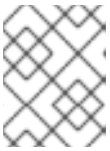
```
(typeattributeset mlsfilewrite (_staff_t_))
```

You can replace ***staff_t*** with a different SELinux type. By specifying SELinux type here, you can control which SELinux roles can edit lower-level files.

To keep your local modules better organized, use the **local_** prefix in the names of local SELinux policy modules.

3. Install the policy module:

```
# semodule -i ~/local_mlsfilewrite.cil
```



NOTE

To remove the local policy module, use **semodule -r ~/*local_mlsfilewrite***. Note that you must refer to the module name without the **.cil** suffix.

4. Optional: If you previously switched back to permissive mode, return to enforcing mode:

```
# setenforce 1
```

Verification

1. Find the local module in the list of installed SELinux modules:

```
# semodule -lfull | grep "local_mls"
400 local_mlsfilewrite  cil
```

Because local modules have priority **400**, you can list them also by using the **semodule -lfull | grep -v ^100** command.

2. Log in as a user assigned to the type defined in the custom rule, for example, ***staff_t***.

3. Attempt to write to a file with a lower sensitivity level. This increases the file's classification level to the user's clearance level.

IMPORTANT

The files you use for verification should not contain any sensitive information in case the configuration is incorrect and the user actually can access the files without authorization.

CHAPTER 7. USING MULTI-CATEGORY SECURITY (MCS) FOR DATA CONFIDENTIALITY

You can use MCS to enhance the data confidentiality of your system by categorizing data, and then granting certain processes and users access to specific categories

7.1. MULTI-CATEGORY SECURITY (MCS)

Multi-Category Security (MCS) is an access control mechanism that uses categories assigned to processes and files. Files can then be accessed only by processes that are assigned to the same categories. The purpose of MCS is to maintain data confidentiality on your system.

MCS categories are defined by the values **c0** to **c1023**, but you can also define a text label for each category or combination of categories, such as “Personnel”, “ProjectX”, or “ProjectX.Personnel”. The MCS Translation service (**mcstrans**) then replaces the category values with the appropriate labels in system inputs and outputs, so that users can use these labels instead of the category values.

When users are assigned to categories, they can label any of their files with any of the categories to which they have been assigned.

MCS works on a simple principle: to access a file, a user must be assigned to all of the categories that have been assigned to the file. The MCS check is applied after normal Linux Discretionary Access Control (DAC) and SELinux Type Enforcement (TE) rules, so it can only further restrict existing security configuration.

MCS within Multi-Level Security

You can use MCS on its own as a non-hierarchical system, or you can use it in combination with Multi-Level Security (MLS) as a non-hierarchical layer within a hierarchical system.

An example of MCS within MLS could be a secretive research organization, where files are classified like this:

Table 7.1. Example of combinations of security levels and categories

Security level	Category			
	Not specified	Project X	Project Y	Project Z
Unclassified	**s0**	**s0:c0**	**s0:c1**	**s0:c2**
Confidential	**s1**	**s1:c0**	**s1:c1**	**s1:c2**
Secret	**s2**	**s2:c0**	**s2:c1**	**s2:c2**
Top secret	**s3**	**s3:c0**	**s3:c1**	**s3:c2**

NOTE

A user with a range **s0:c0.1023** would be able to access all files assigned to all categories on level **s0**, unless the access is prohibited by other security mechanisms, such as DAC or type enforcement policy rules.

The resulting security context of a file or process is a combination of:

- SELinux user
- SELinux role
- SELinux type
- MLS sensitivity level
- MCS category

For example, a non-privileged user with access to sensitivity level 1 and category 2 in an MLS/MCS environment could have the following SELinux context:

```
user_u:user_r:user_t:s1:c2
```

Additional resources

- Using Multi-Level Security (MLS) .

7.2. CONFIGURING MULTI-CATEGORY SECURITY FOR DATA CONFIDENTIALITY

By default, MCS is active in the **targeted** and **mls** SELinux policies but is not configured for users. In the **targeted** policy, MCS is configured only for:

- OpenShift
- virt
- sandbox
- network labeling
- containers (**container-selinux**)

You can configure MCS to categorize users by creating a local SELinux module with a rule that constrains the **user_t** SELinux type by MCS rules in addition to type enforcement.

WARNING

Changing the categories of certain files may render some services non-operational. If you are not an expert, contact your Red Hat sales representative and request consulting services.

Prerequisites

- The SELinux mode is set to **enforcing**.

- The SELinux policy is set to **targeted** or **mls**.
- The **policycoreutils-python-utils** and **setools-console** packages are installed.

Procedure

1. Create a new file named, for example, **local_mcs_user.cil**:

```
# vim local_mcs_user.cil
```

2. Insert the following rule:

```
(typeattributeset mcs_constrained_type (user_t))
```

3. Install the policy module:

```
# semodule -i local_mcs_user.cil
```

Verification

- For each user domain, display additional details for all the components:

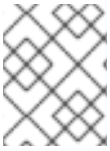
```
# seinfo -xt user_t

Types: 1
type user_t, application_domain_type, nsswitch_domain, corenet_unlabeled_type, domain,
kernel_system_state_reader, mcs_constrained_type, netlabel_peer_type, privfd,
process_user_target, scsi_generic_read, scsi_generic_write, syslog_client_type,
pcmcia_typeattr_1, user_usertype, login_userdomain, userdomain, unpriv_userdomain,
userdom_home_reader_type, userdom_filetrans_type, xdmhomewriter, x_userdomain,
x_domain, dridomain, xdrawable_type, xcolormap_type;
```

Additional resources

- Creating a local SELinux policy module
- For more information about MCS in the context of containers, see the blog posts How SELinux separates containers using Multi-Level Security and Why you should be using Multi-Category Security for your Linux containers.

7.3. DEFINING CATEGORY LABELS IN MCS

You can manage and maintain labels for MCS categories, or combinations of MCS categories with MLS levels, on your system by editing the **setrans.conf** file. In this file, SELinux maintains a mapping between internal sensitivity and category levels and their human-readable labels.



NOTE

Category labels only make it easier for users to use the categories. MCS works the same whether you define labels or not.

Prerequisites

- The SELinux mode is set to **enforcing**.
- The SELinux policy is set to **targeted** or **mls**.
- The **policycoreutils-python-utils** and **mcstrans** packages are installed.

Procedure

1. Modify existing categories or create new categories by editing the **/etc/selinux/*<selinuxpolicy>*/setrans.conf** file in a text editor. Replace *<selinuxpolicy>* with **targeted** or **mls** depending on the SELinux policy you use. For example:

```
# vi /etc/selinux/targeted/setrans.conf
```

2. In the **setrans.conf** file for your policy, define the combinations of categories required by your scenario using the syntax **s_<security level>_:c_<category number>_=*<category.name>***, for example:

```
s0:c0=Marketing
s0:c1=Finance
s0:c2=Payroll
s0:c3=Personnel
```

   - You can use category numbers from **c0** to **c1023**.
   - In the **targeted** policy, use the **s0** security level.
   - In the **mls** policy, you can label each combination of sensitivity levels and categories.

3. Optional: In the **setrans.conf** file, you can also label the MLS sensitivity levels.
4. Save and exit the file.
5. To make the changes effective, restart the MCS translation service:

```
# systemctl restart mcstrans
```

Verification

- Display the current categories:

```
# chcat -L
```

  The example above produces the following output:

```
s0:c0                   Marketing
s0:c1                   Finance
s0:c2                   Payroll
s0:c3                   Personnel
s0
s0-s0:c0.c1023              SystemLow-SystemHigh
s0:c0.c1023                 SystemHigh
```

Additional resources

- The **setrans.conf(5)** man page.

7.4. ASSIGNING CATEGORIES TO USERS IN MCS

You can define user authorizations by assigning categories to Linux users. A user with assigned categories can access and modify files that have a subset of the user's categories. Users can also assign files they own to categories they have been assigned to.

A Linux user cannot be assigned to a category that is outside of the security range defined for the relevant SELinux user.

NOTE

Category access is assigned during login. Consequently, users do not have access to newly assigned categories until they log in again. Similarly, if you revoke a user's access to a category, this is effective only after the user logs in again.

Prerequisites

- The SELinux mode is set to **enforcing**.
- The SELinux policy is set to **targeted** or **mls**.
- The **policycoreutils-python-utils** package is installed.
- Linux users are assigned to SELinux confined users:
  - Non-privileged users are assigned to **user_u**.
  - Privileged users are assigned to **staff_u**.

Procedure

1. Define the security range for the SELinux user.

```
# semanage user -m -rs0:c0,c1-s0:c0.c9 <user_u>
```

   Use category numbers **c0** to **c1023** or category labels as defined in the **setrans.conf** file. For additional information, see Defining category labels in MCS .

2. Assign MCS categories to a Linux user. You can specify only a range within the range defined to the relevant SELinux user:

```
# semanage login -m -rs0:c1 <Linux.user1>
```

NOTE

You can add or remove categories from Linux users by using the **chcat** command. The following example adds ***<category1>*** and removes ***<category2>*** from ***<Linux.user1>*** and ***<Linux.user2>***:

```
# chcat -l -- +<category1>,-<category2> <Linux.user1>,<Linux.user2>
```

Note that you must specify **--** on the command line before using the ***-<category>*** syntax. Otherwise, the **chcat** command misinterprets the category removal as a command option.

Verification

- List the categories assigned to Linux users:

```
# chcat -L -l <Linux.user1>,<Linux.user2>
<Linux.user1>: <category1>,<category2>
<Linux.user2>: <category1>,<category2>
```

Additional resources

- The **chcat(8)** man page.

7.5. ASSIGNING CATEGORIES TO FILES IN MCS

You need administrative privileges to assign categories to users. Users can then assign categories to files. To modify the categories of a file, users must have access rights to that file. Users can only assign a file to a category that is assigned to them.

NOTE

The system combines category access rules with conventional file access permissions. For example, if a user with a category of **bigfoot** uses Discretionary Access Control (DAC) to block access to a file by other users, other **bigfoot** users cannot access that file. A user assigned to all available categories still may not be able to access the entire file system.

Prerequisites

- The SELinux mode is set to **enforcing**.
- The SELinux policy is set to **targeted** or **mls**.
- The **policycoreutils-python-utils** package is installed.
- Access and permissions to a Linux user that is:
  - Assigned to an SELinux user.
  - Assigned to the category to which you want to assign the file. For additional information, see Assigning categories to users in MCS .
- Access and permissions to the file you want to add to the category.

- For verification purposes: Access and permissions to a Linux user not assigned to this category

Procedure

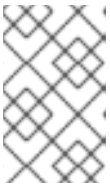
- Add categories to a file:

```
$ chcat -- +<category1>,+<category2> <path/to/file1>
```

Use category numbers **c0** to **c1023** or category labels as defined in the **setrans.conf** file. For additional information, see Defining category labels in MCS .

You can remove categories from a file by using the same syntax:

```
$ chcat -- -<category1>,-<category2> <path/to/file1>
```



NOTE

When removing a category, you must specify **--** on the command line before using the ***-<category>*** syntax. Otherwise, the **chcat** command could misinterpret the category removal as a command option.

Verification

1. Display the security context of the file to verify that it has the correct categories:

```
$ ls -lZ <path/to/file>
-rw-r--r--  <LinuxUser1> <Group1> root:object_r:user_home_t:_<sensitivity>_:_<category>_
<path/to/file>
```

The specific security context of the file may differ.

2. Optional: Attempt to access the file when logged in as a Linux user not assigned to the same category as the file:

```
$ cat <path/to/file>
cat: <path/to/file>: Permission Denied
```

Additional resources

- The **semanage(8)** man page.
- The **chcat(8)** man page.

CHAPTER 8. WRITING A CUSTOM SELINUX POLICY

This section guides you on how to write and use a custom policy that enables you to run your applications confined by SELinux.

8.1. CUSTOM SELINUX POLICIES AND RELATED TOOLS

An SELinux security policy is a collection of SELinux rules. A policy is a core component of SELinux and is loaded into the kernel by SELinux user-space tools. The kernel enforces the use of an SELinux policy to evaluate access requests on the system. By default, SELinux denies all requests except for requests that correspond to the rules specified in the loaded policy.

Each SELinux policy rule describes an interaction between a process and a system resource:

```
ALLOW apache_process apache_log:FILE READ;
```

You can read this example rule as: *The **Apache** process can **read** its **logging file***. In this rule, **apache_process** and **apache_log** are **labels**. An SELinux security policy assigns labels to processes and defines relations to system resources. This way, a policy maps operating-system entities to the SELinux layer.

SELinux labels are stored as extended attributes of file systems, such as **ext2**. You can list them using the **getfattr** utility or a **ls -Z** command, for example:

```
$ ls -Z /etc/passwd
system_u:object_r:passwd_file_t:s0 /etc/passwd
```

Where **system_u** is an SELinux user, **object_r** is an example of the SELinux role, and **passwd_file_t** is an SELinux domain.

The default SELinux policy provided by the **selinux-policy** packages contains rules for applications and daemons that are parts of Red Hat Enterprise Linux 8 and are provided by packages in its repositories. Applications not described in a rule in this distribution policy are not confined by SELinux. To change this, you have to modify the policy using a policy module, which contains additional definitions and rules.

In Red Hat Enterprise Linux 8, you can query the installed SELinux policy and generate new policy modules using the **sepolicy** tool. Scripts that **sepolicy** generates together with the policy modules always contain a command using the **restorecon** utility. This utility is a basic tool for fixing labeling problems in a selected part of a file system.

Additional resources

- **sepolicy(8)** and **getfattr(1)** man pages

8.2. CREATING AND ENFORCING AN SELINUX POLICY FOR A CUSTOM APPLICATION

This example procedure provides steps for confining a simple daemon by SELinux. Replace the daemon with your custom application and modify the example rule according to the requirements of that application and your security policy.

Prerequisites

- The **policycoreutils-devel** package and its dependencies are installed on your system.

Procedure

1. For this example procedure, prepare a simple daemon that opens the **/var/log/messages** file for writing:

   a. Create a new file, and open it in a text editor of your choice:

   ```
   $ vi mydaemon.c
   ```

   b. Insert the following code:

   ```
   #include <unistd.h>
   #include <stdio.h>

   FILE *f;

   int main(void)
   {
   while(1) {
   f = fopen("/var/log/messages","w");
           sleep(5);
           fclose(f);
       }
   }
   ```

   c. Compile the file:

   ```
   $ gcc -o mydaemon mydaemon.c
   ```

   d. Create a **systemd** unit file for your daemon:

   ```
   $ vi mydaemon.service
   [Unit]
   Description=Simple testing daemon

   [Service]
   Type=simple
   ExecStart=/usr/local/bin/mydaemon

   [Install]
   WantedBy=multi-user.target
   ```

   e. Install and start the daemon:

   ```
   # cp mydaemon /usr/local/bin/
   # cp mydaemon.service /usr/lib/systemd/system
   # systemctl start mydaemon
   # systemctl status mydaemon
   ● mydaemon.service - Simple testing daemon
     Loaded: loaded (/usr/lib/systemd/system/mydaemon.service; disabled; vendor preset:
   disabled)
     Active: active (running) since Sat 2020-05-23 16:56:01 CEST; 19s ago
   ```

```
 Main PID: 4117 (mydaemon)
   Tasks: 1
  Memory: 148.0K
  CGroup: /system.slice/mydaemon.service
          └─4117 /usr/local/bin/mydaemon

May 23 16:56:01 localhost.localdomain systemd[1]: Started Simple testing daemon.
```

f. Check that the new daemon is not confined by SELinux:

```
$ ps -efZ | grep mydaemon
system_u:system_r:unconfined_service_t:s0 root 4117    1  0 16:56 ?        00:00:00
/usr/local/bin/mydaemon
```

2. Generate a custom policy for the daemon:

```
$ sepolicy generate --init /usr/local/bin/mydaemon
Created the following files:
/home/example.user/mysepol/mydaemon.te # Type Enforcement file
/home/example.user/mysepol/mydaemon.if # Interface file
/home/example.user/mysepol/mydaemon.fc # File Contexts file
/home/example.user/mysepol/mydaemon_selinux.spec # Spec file
/home/example.user/mysepol/mydaemon.sh # Setup Script
```

3. Rebuild the system policy with the new policy module using the setup script created by the previous command:

```
# ./mydaemon.sh
Building and Loading Policy
+ make -f /usr/share/selinux/devel/Makefile mydaemon.pp
Compiling targeted mydaemon module
Creating targeted mydaemon.pp policy package
rm tmp/mydaemon.mod.fc tmp/mydaemon.mod
+ /usr/sbin/semodule -i mydaemon.pp
...
```

Note that the setup script relabels the corresponding part of the file system using the **restorecon** command:

```
restorecon -v /usr/local/bin/mydaemon /usr/lib/systemd/system
```

4. Restart the daemon, and check that it now runs confined by SELinux:

```
# systemctl restart mydaemon
$ ps -efZ | grep mydaemon
system_u:system_r:mydaemon_t:s0 root        8150       1  0 17:18 ?        00:00:00
/usr/local/bin/mydaemon
```

5. Because the daemon is now confined by SELinux, SELinux also prevents it from accessing **/var/log/messages**. Display the corresponding denial message:

```
# ausearch -m AVC -ts recent
...
type=AVC msg=audit(1590247112.719:5935): avc:  denied  { open } for  pid=8150
```

```
comm="mydaemon" path="/var/log/messages" dev="dm-0" ino=2430831
scontext=system_u:system_r:mydaemon_t:s0 tcontext=unconfined_u:object_r:var_log_t:s0
tclass=file permissive=1
...
```

6. You can get additional information also using the **sealert** tool:

```
$ sealert -l "*"
SELinux is preventing mydaemon from open access on the file /var/log/messages.

 Plugin catchall (100. confidence) suggests *

If you believe that mydaemon should be allowed open access on the messages file by
default.
Then you should report this as a bug.
You can generate a local policy module to allow this access.
Do
allow this access for now by executing:
# ausearch -c 'mydaemon' --raw | audit2allow -M my-mydaemon
# semodule -X 300 -i my-mydaemon.pp

Additional Information:
Source Context                system_u:system_r:mydaemon_t:s0
Target Context                unconfined_u:object_r:var_log_t:s0
Target Objects                /var/log/messages [ file ]
Source                        mydaemon

...
```

7. **Use the audit2allow tool to suggest changes:**

```
$ ausearch -m AVC -ts recent | audit2allow -R

require {
 type mydaemon_t;
}

#============= mydaemon_t ==============
logging_write_generic_logs(mydaemon_t)
```

8. **Because rules suggested by audit2allow can be incorrect for certain cases, use only a part of its output to find the corresponding policy interface:**

```
$ grep -r "logging_write_generic_logs" /usr/share/selinux/devel/include/ | grep .if
/usr/share/selinux/devel/include/system/logging.if:interface(`logging_write_generic_logs',`
```

9. **Check the definition of the interface:**

```
$ cat /usr/share/selinux/devel/include/system/logging.if
...
interface(`logging_write_generic_logs',`
        gen_require(`
                type var_log_t;
        ')
```

```
        files_search_var($1)
        allow $1 var_log_t:dir list_dir_perms;
        write_files_pattern($1, var_log_t, var_log_t)
')
...
```

10. **In this case, you can use the suggested interface. Add the corresponding rule to your type enforcement file:**

```
$ echo "logging_write_generic_logs(mydaemon_t)" >> mydaemon.te
```

Alternatively, you can add this rule instead of using the interface:

```
$ echo "allow mydaemon_t var_log_t:file { open write getattr };" >> mydaemon.te
```

11. **Reinstall the policy:**

```
# ./mydaemon.sh
Building and Loading Policy
+ make -f /usr/share/selinux/devel/Makefile mydaemon.pp
Compiling targeted mydaemon module
Creating targeted mydaemon.pp policy package
rm tmp/mydaemon.mod.fc tmp/mydaemon.mod
+ /usr/sbin/semodule -i mydaemon.pp
...
```

Verification

1. **Check that your application runs confined by SELinux, for example:**

```
$ ps -efZ | grep mydaemon
system_u:system_r:mydaemon_t:s0 root       8150       1  0 17:18 ?        00:00:00
/usr/local/bin/mydaemon
```

2. **Verify that your custom application does not cause any SELinux denials:**

```
# ausearch -m AVC -ts recent
<no matches>
```

Additional resources

- **sepolgen(8)**, **ausearch(8)**, **audit2allow(1)**, **audit2why(1)**, **sealert(8)**, and **restorecon(8)** man pages

8.3. CREATING A LOCAL SELINUX POLICY MODULE

Adding specific SELinux policy modules to an active SELinux policy can fix certain problems with the SELinux policy. You can use this procedure to fix a specific Known Issue described in Red Hat release notes, or to implement a specificRed Hat Solution.

WARNING

Use only rules provided by Red Hat. Red Hat does not support creating SELinux policy modules with custom rules, because this falls outside of the Production Support Scope of Coverage. If you are not an expert, contact your Red Hat sales representative and request consulting services.

Prerequisites

- The **setools-console** and **audit** packages for verification.

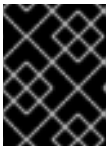
Procedure

1. Open a new **.cil** file with a text editor, for example:

```
# vim <local_module>.cil
```

To keep your local modules better organized, use the **local_** prefix in the names of local SELinux policy modules.

2. Insert the custom rules from a Known Issue or a Red Hat Solution.



IMPORTANT

Do not write your own rules. Use only the rules provided in a specific Known Issue or Red Hat Solution.

For example, to implement the SELinux denies cups-lpd read access to cups.sock in RHEL solution, insert the following rule:

NOTE

The example solution has been fixed permanently for RHEL in RHBA-2021:4420. Therefore, the parts of this procedure specific to this solution have no effect on updated RHEL 8 and 9 systems, and are included only as examples of syntax.

```
(allow cupsd_lpd_t cupsd_var_run_t (sock_file (read)))
```

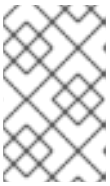
Note that you can use either of the two SELinux rule syntaxes, Common Intermediate Language (CIL) and m4. For example, **(allow cupsd_lpd_t cupsd_var_run_t (sock_file (read)))** in CIL is equivalent to the following in m4:

```
module local_cupslpd-read-cupssock 1.0;

require {
    type cupsd_var_run_t;
    type cupsd_lpd_t;
    class sock_file read;
```

```
}

#============= cupsd_lpd_t ==============
allow cupsd_lpd_t cupsd_var_run_t:sock_file read;
```

3. Save and close the file.
4. Install the policy module:

```
# semodule -i <local_module>.cil
```



NOTE

When you want to remove a local policy module which you created by using **semodule -i**, refer to the module name without the **.cil** suffix. To remove a local policy module, use **semodule -r *<local_module>***.

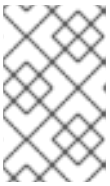
5. Restart any services related to the rules:

```
# systemctl restart <service-name>
```

Verification

1. List the local modules installed in your SELinux policy:

```
# semodule -lfull | grep "local_"
400 local_module  cil
```



NOTE

Because local modules have priority **400**, you can filter them from the list also by using that value, for example, by using the **semodule -lfull | grep -v ^100** command.

2. Search the SELinux policy for the relevant allow rules:

```
# sesearch -A --source=<SOURCENAME> --target=<TARGETNAME> --
class=<CLASSNAME> --perm=<P1>,<P2>
```

Where ***<SOURCENAME>*** is the source SELinux type, ***<TARGETNAME>*** is the target SELinux type, ***<CLASSNAME>*** is the security class or object class name, and ***<P1>*** and ***<P2>*** are the specific permissions of the rule.

For example, for the SELinux denies cups-lpd read access to cups.sock in RHEL solution:

```
# sesearch -A --source=cupsd_lpd_t --target=cupsd_var_run_t --class=sock_file --
perm=read
allow cupsd_lpd_t cupsd_var_run_t:sock_file { append getattr open read write };
```

The last line should now include the **read** operation.

3. Verify that the relevant service runs confined by SELinux:

a. **Identify the process related to the relevant service:**

```
$ systemctl status <service-name>
```

b. **Check the SELinux context of the process listed in the output of the previous command:**

```
$ ps -efZ | grep <process-name>
```

4. **Verify that the service does not cause any SELinux denials:**

```
# ausearch -m AVC -ts recent
<no matches>
```

Additional resources

- Chapter 5, *Troubleshooting problems related to SELinux*

8.4. ADDITIONAL RESOURCES

- SELinux Policy Workshop

CHAPTER 9. CREATING SELINUX POLICIES FOR CONTAINERS

Red Hat Enterprise Linux 8 provides a tool for generating SELinux policies for containers using the **udica** package. With **udica**, you can create a tailored security policy for better control of how a container accesses host system resources, such as storage, devices, and network. This enables you to harden your container deployments against security violations and it also simplifies achieving and maintaining regulatory compliance.

9.1. INTRODUCTION TO THE UDICA SELINUX POLICY GENERATOR

To simplify creating new SELinux policies for custom containers, RHEL 8 provides the **udica** utility. You can use this tool to create a policy based on an inspection of the container JavaScript Object Notation (JSON) file, which contains Linux-capabilities, mount-points, and ports definitions. The tool consequently combines rules generated using the results of the inspection with rules inherited from a specified SELinux Common Intermediate Language (CIL) block.

The process of generating SELinux policy for a container using **udica** has three main parts:

1. Parsing the container spec file in the JSON format
2. Finding suitable allow rules based on the results of the first part
3. Generating final SELinux policy

During the parsing phase, **udica** looks for Linux capabilities, network ports, and mount points.

Based on the results, **udica** detects which Linux capabilities are required by the container and creates an SELinux rule allowing all these capabilities. If the container binds to a specific port, **udica** uses SELinux user-space libraries to get the correct SELinux label of a port that is used by the inspected container.

Afterward, **udica** detects which directories are mounted to the container file-system name space from the host.

The CIL's block inheritance feature allows **udica** to create templates of SELinux*allow rules* focusing on a specific action, for example:

- *allow accessing home directories*
- *allow accessing log files*
- *allow accessing communication with Xserver.*

These templates are called blocks and the final SELinux policy is created by merging the blocks.

Additional resources

- Generate SELinux policies for containers with udicaRed Hat Blog article

9.2. CREATING AND USING AN SELINUX POLICY FOR A CUSTOM CONTAINER

To generate an SELinux security policy for a custom container, follow the steps in this procedure.

Prerequisites

- The **podman** tool for managing containers is installed. If it is not, use the **yum install podman** command.
- A custom Linux container - *ubi8* in this example.

Procedure

1. Install the **udica** package:

```
# yum install -y udica
```

Alternatively, install the **container-tools** module, which provides a set of container software packages, including **udica**:

```
# yum module install -y container-tools
```

2. Start the *ubi8* container that mounts the/**home** directory with read-only permissions and the /**var**/**spool** directory with permissions to read and write. The container exposes the port *21*.

```
# podman run --env container=podman -v /home:/home:ro -v /var/spool:/var/spool:rw -p
21:21 -it ubi8 bash
```

Note that now the container runs with the **container_t** SELinux type. This type is a generic domain for all containers in the SELinux policy and it might be either too strict or too loose for your scenario.

3. Open a new terminal, and enter the **podman ps** command to obtain the ID of the container:

```
# podman ps
CONTAINER ID  IMAGE                                 COMMAND  CREATED         STATUS
PORTS  NAMES
37a3635afb8f  registry.access.redhat.com/ubi8:latest  bash     15 minutes ago  Up 15
minutes ago          heuristic_lewin
```

4. Create a container JSON file, and use **udica** for creating a policy module based on the information in the JSON file:

```
# podman inspect 37a3635afb8f > container.json
# udica -j container.json my_container
Policy my_container with container id 37a3635afb8f created!
[...]
```

Alternatively:

```
# podman inspect 37a3635afb8f | udica my_container
Policy my_container with container id 37a3635afb8f created!

Please load these modules using:
# semodule -i my_container.cil
```

```
/usr/share/udica/templates/{base_container.cil,net_container.cil,home_container.cil}

Restart the container with: "--security-opt label=type:my_container.process" parameter
```

5. **As suggested by the output of udica in the previous step, load the policy module:**

```
# semodule -i my_container.cil
/usr/share/udica/templates/{base_container.cil,net_container.cil,home_container.cil}
```

6. **Stop the container and start it again with the --security-opt label=type:my_container.process option:**

```
# podman stop 37a3635afb8f
# podman run --security-opt label=type:my_container.process -v /home:/home:ro -v
/var/spool:/var/spool:rw -p 21:21 -it ubi8 bash
```

Verification

1. **Check that the container runs with the my_container.process type:**

```
# ps -efZ | grep my_container.process
unconfined_u:system_r:container_runtime_t:s0-s0:c0.c1023 root 2275 434  1 13:49 pts/1
00:00:00 podman run --security-opt label=type:my_container.process -v /home:/home:ro -v
/var/spool:/var/spool:rw -p 21:21 -it ubi8 bash
system_u:system_r:my_container.process:s0:c270,c963 root 2317 2305  0 13:49 pts/0
00:00:00 bash
```

2. **Verify that SELinux now allows access the /home and /var/spool mount points:**

```
[root@37a3635afb8f /]# cd /home
[root@37a3635afb8f home]# ls
username
[root@37a3635afb8f ~]# cd /var/spool/
[root@37a3635afb8f spool]# touch test
[root@37a3635afb8f spool]#
```

3. **Check that SELinux allows binding only to the port 21:**

```
[root@37a3635afb8f /]# yum install nmap-ncat
[root@37a3635afb8f /]# nc -lvp 21
...
Ncat: Listening on :::21
Ncat: Listening on 0.0.0.0:21
^C
[root@37a3635afb8f /]# nc -lvp 80
...
Ncat: bind to :::80: Permission denied. QUITTING.
```

Additional resources

- **udica(8)** and **podman(1)** man pages
- Building, running, and managing containers

9.3. ADDITIONAL RESOURCES

- udica - Generate SELinux policies for containers

CHAPTER 10. DEPLOYING THE SAME SELINUX CONFIGURATION ON MULTIPLE SYSTEMS

This section provides two recommended ways for deploying your verified SELinux configuration on multiple systems:

- Using RHEL System Roles and Ansible
- Using **semanage** export and import commands in your scripts

10.1. INTRODUCTION TO THE SELINUX SYSTEM ROLE

RHEL System Roles is a collection of Ansible roles and modules that provide a consistent configuration interface to remotely manage multiple RHEL systems. The **selinux** System Role enables the following actions:

- Cleaning local policy modifications related to SELinux booleans, file contexts, ports, and logins.
- Setting SELinux policy booleans, file contexts, ports, and logins.
- Restoring file contexts on specified files or directories.
- Managing SELinux modules.

The following table provides an overview of input variables available in the **selinux** System Role.

Table 10.1. **selinux** System Role variables

Role variable	Description	CLI alternative
selinux_policy	Chooses a policy protecting targeted processes or Multi Level Security protection.	**SELINUXTYPE** in **/etc/selinux/config**
selinux_state	Switches SELinux modes.	**setenforce** and **SELINUX** in **/etc/selinux/config**.
selinux_booleans	Enables and disables SELinux booleans.	**setsebool**
selinux_fcontexts	Adds or removes a SELinux file context mapping.	**semanage fcontext**
selinux_restore_dirs	Restores SELinux labels in the file-system tree.	**restorecon -R**
selinux_ports	Sets SELinux labels on ports.	**semanage port**
selinux_logins	Sets users to SELinux user mapping.	**semanage login**

Role variable	Description	CLI alternative
selinux_modules	Installs, enables, disables, or removes SELinux modules.	**semodule**

The **/usr/share/doc/rhel-system-roles/selinux/example-selinux-playbook.yml** example playbook installed by the **rhel-system-roles** package demonstrates how to set the targeted policy in enforcing mode. The playbook also applies several local policy modifications and restores file contexts in the **/tmp/test_dir/** directory.

For a detailed reference on **selinux** role variables, install the **rhel-system-roles** package, and see the **README.md** or **README.html** files in the **/usr/share/doc/rhel-system-roles/selinux/** directory.

Additional resources

- Introduction to RHEL System Roles.

10.2. USING THE SELINUX SYSTEM ROLE TO APPLY SELINUX SETTINGS ON MULTIPLE SYSTEMS

Follow the steps to prepare and apply an Ansible playbook with your verified SELinux settings.

Prerequisites

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the **selinux** System Role.
- Access and permissions to a *control node*, which is a system from which Red Hat Ansible Core configures other systems.
  On the control node:
  - The **ansible-core** and **rhel-system-roles** packages are installed.
  - An inventory file which lists the managed nodes.

IMPORTANT

RHEL 8.0-8.5 provided access to a separate Ansible repository that contains Ansible Engine 2.9 for automation based on Ansible. Ansible Engine contains command-line utilities such as **ansible**, **ansible-playbook**, connectors such as **docker** and **podman**, and many plugins and modules. For information on how to obtain and install Ansible Engine, see the How to download and install Red Hat Ansible EngineKnowledgebase article.

RHEL 8.6 and 9.0 have introduced Ansible Core (provided as the **ansible-core** package), which contains the Ansible command-line utilities, commands, and a small set of built-in Ansible plugins. RHEL provides this package through the AppStream repository, and it has a limited scope of support. For more information, see the Scope of support for the Ansible Core package included in the RHEL 9 and RHEL 8.6 and later AppStream repositories Knowledgebase article.

- An inventory file which lists the managed nodes.

Procedure

1. Prepare your playbook. You can either start from the scratch or modify the example playbook installed as a part of the **rhel-system-roles** package:

```
# cp /usr/share/doc/rhel-system-roles/selinux/example-selinux-playbook.yml my-selinux-playbook.yml
# vi my-selinux-playbook.yml
```

2. Change the content of the playbook to fit your scenario. For example, the following part ensures that the system installs and enables the **selinux-local-1.pp** SELinux module:

```
selinux_modules:
- { path: "selinux-local-1.pp", priority: "400" }
```

3. Save the changes, and exit the text editor.

4. Run your playbook on the *host1*, *host2*, and *host3* systems:

```
# ansible-playbook -i host1,host2,host3 my-selinux-playbook.yml
```

Additional resources

- For more information, install the **rhel-system-roles** package, and see the **/usr/share/doc/rhel-system-roles/selinux/** and **/usr/share/ansible/roles/rhel-system-roles.selinux/** directories.

10.3. TRANSFERRING SELINUX SETTINGS TO ANOTHER SYSTEM WITH SEMANAGE

Use the following steps for transferring your custom and verified SELinux settings between RHEL 8-based systems.

Prerequisites

- The **policycoreutils-python-utils** package is installed on your system.

Procedure

1. Export your verified SELinux settings:

```
# semanage export -f ./my-selinux-settings.mod
```

2. Copy the file with the settings to the new system:

```
# scp ./my-selinux-settings.mod new-system-hostname:
```

3. Log in on the new system:

```
$ ssh root@new-system-hostname
```

4. Import the settings on the new system:

```
new-system-hostname# semanage import -f ./my-selinux-settings.mod
```

Additional resources

- **semanage-export(8)** and **semanage-import(8)** man pages