Red Hat Enterprise Linux 8

Using SELinux

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

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION ........................................ 3

CHAPTER 1. GETTING STARTED WITH SELINUX .............................................. 4
   1.1. INTRODUCTION TO SELINUX .................................................. 4
   1.2. BENEFITS OF RUNNING SELINUX ............................................. 5
   1.3. SELINUX EXAMPLES ............................................................. 6
   1.4. SELINUX ARCHITECTURE AND PACKAGES ............................... 6
   1.5. SELINUX STATES AND MODES ................................................. 7

CHAPTER 2. CHANGING SELINUX STATES AND MODES .................................. 9
   2.1. PERMANENT CHANGES IN SELINUX STATES AND MODES ............. 9
   2.2. ENABLING SELINUX ............................................................ 9
      2.2.1. Changing to permissive mode ....................................... 10
      2.2.2. Changing to enforcing mode ....................................... 10
   2.3. DISABLING SELINUX ......................................................... 11
   2.4. CHANGING SELINUX MODES AT BOOT TIME ............................ 12

CHAPTER 3. CONFIGURING SELINUX FOR APPLICATIONS AND SERVICES WITH NON-STANDARD CONFIGURATIONS .............................................. 13
   3.1. CUSTOMIZING THE SELINUX POLICY FOR THE APACHE HTTP SERVER IN A NON-STANDARD CONFIGURATION .......................... 13
   3.2. ADJUSTING THE POLICY FOR SHARING NFS AND CIFS VOLUMES USING SELINUX BOOLEANS ............................. 15
   3.3. ADDITIONAL RESOURCES .................................................... 16

CHAPTER 4. TROUBLESHOOTING PROBLEMS RELATED TO SELINUX ............ 17
   4.1. IDENTIFYING SELINUX DENIALS .......................................... 17
   4.2. ANALYZING SELINUX DENIAL MESSAGES ................................ 18
   4.3. FIXING ANALYZED SELINUX DENIALS ................................... 19
   4.4. SELINUX DENIALS IN THE AUDIT LOG ................................... 22
   4.5. RELATED INFORMATION ...................................................... 23

CHAPTER 5. CREATING SELINUX POLICIES FOR CONTAINERS ....................... 24
   5.1. INTRODUCTION TO THE UDICA SELINUX POLICY GENERATOR ...... 24
   5.2. CREATING AND USING AN SELINUX POLICY FOR A CUSTOM CONTAINER ......................................................... 24
   5.3. ADDITIONAL RESOURCES .................................................... 26
We appreciate your input on our documentation. Please let us know how we could make it better. To do so:

- For simple comments on specific passages, make sure you are viewing the documentation in the Multi-page HTML format. Highlight the part of text that you want to comment on. Then, click the Add Feedback pop-up that appears below the highlighted text, and follow the displayed instructions.

- For submitting more complex feedback, create a Bugzilla ticket:
  1. Go to the Bugzilla website.
  2. As the Component, use Documentation.
  3. Fill in the Description field with your suggestion for improvement. Include a link to the relevant part(s) of documentation.
  4. Click Submit Bug.
CHAPTER 1. GETTING STARTED WITH SELINUX

1.1. INTRODUCTION TO 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?

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.

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.
As the previous scheme shows, SELinux allows the Apache process running as \texttt{httpd\_t} to access the \texttt{/var/www/html/} directory and it denies the same process to access the \texttt{/data/mysql/} directory because there is no allow rule for the \texttt{httpd\_t} and \texttt{mysqld\_db\_t} type contexts). On the other hand, the MariaDB process running as \texttt{mysqld\_t} is able to access the \texttt{/data/mysql/} directory and SELinux also correctly denies the process with the \texttt{mysqld\_t} type to access the \texttt{/var/www/html/} directory labeled as \texttt{httpd\_sys\_content\_t}.

### Additional resources

To better understand SELinux basic concepts, see the following documentation:

- The \texttt{selinux(8)} man page and the \texttt{man -k selinux} command.
- The SELinux Coloring Book
- 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, as well as preventing them from executing potentially malicious files and applications in their home directory.

- 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 what is known as 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 Red Hat Enterprise Linux, even if an administrator forgets to limit which servers can perform a zone transfer, the default SELinux policy prevents zone files from being updated using zone transfers, by the BIND named daemon itself, and by other processes.

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 Red Hat Enterprise Linux 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.

**IMPORTANT**

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

Red Hat Enterprise Linux 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
```
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:

```bash
# 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 information, such as host name to IP address mappings, that are used by DNS servers.
CHAPTER 2. CHANGING SELINUX STATES AND 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
```

**NOTE**

When systems run SELinux in permissive mode, users and processes can 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, file systems are automatically relabeled when changing from the disabled state to permissive or enforcing mode. In permissive 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.

2.2. ENABLING SELINUX

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

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

1. Enable SELinux in permissive mode. For more information, see Changing to permissive mode.
2. Reboot your system.
3. Check for SELinux denial messages. For more information, see Identifying SELinux denials.
4. If there are no denials, switch to enforcing mode. For more information, see Changing to enforcing mode.

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 Custom SELinux Policy Knowledgebase article for more information.

Temporary changes in modes are covered in SE Linux states and modes.

2.2.1. Changing to permissive mode

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

To permanently change mode to permissive, follow the procedure below:

1. Edit the `/etc/selinux/config` file as follows:
   ```
   # 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
   ```

2. Reboot the system:
   ```
   # reboot
   ```

2.2.2. Changing to enforcing mode

When SE Linux is running in enforcing mode, it enforces the SE Linux policy and denies access based on SE Linux policy rules. In Red Hat Enterprise Linux, enforcing mode is enabled by default when the system was initially installed with SE Linux.

Prerequisites

This procedure assumes that the `selinux-policy-targeted`, `libselinux-utils`, and `policycoreutils` packages are installed.

Procedure

If SE Linux was disabled, follow the procedure below to change mode to enforcing again:

1. Edit the `/etc/selinux/config` file as follows:
   ```
   # 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

2. Reboot 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.

**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
```

### 2.3. DISABLING SELINUX

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.

**IMPORTANT**

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

To permanently disable SELinux, follow the procedure below:

1. Configure `SELINUX=disabled` in the `/etc/selinux/config` file:

   ```
   # 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

2. Reboot your system. After reboot, confirm that the `getenforce` command returns **Disabled**:

```
$ getenforce
Disabled
```

### 2.4. 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 machine to boot 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 is reported. However, in enforcing mode you might get a denial on 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, you might need to boot in permissive mode in order to make the autorelabel process successful.
CHAPTER 3. 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.

In the following procedures, 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.

3.1. 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 package is installed.

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.
   [trimmed for clarity]
   ```

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:
5. However, the content remains inaccessible:

```
# wget localhost:3131/index.html
```

HTTP request sent, awaiting response... 403 Forbidden
[trimmed for clarity]

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.
[trimmed for clarity]

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

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

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
[trimmed for clarity]

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

```
# wget localhost:3131/index.html
```

...
Additional resources

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

### 3.2. 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:
   ```bash
   # 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:
   ```bash
   $ getsebool -a | grep 'nfs\|cifs' | grep httpd
   httpd_use_cifs --> off
   httpd_use_nfs --> off
   ```

3. Enable the identified booleans:
   ```bash
   # 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 steps

1. Check that the booleans are on:

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

Additional resources

- The `semanage-boolean(8)`, `sepolicy-BOOLEANS(8)`, `getsebool(8)`, `setsebool(8)`, `BOOLEANS(5)`, and `BOOLEANS(8)` man pages.

3.3. ADDITIONAL RESOURCES

- See `Troubleshooting problems related to SELinux` for more details on identifying and analyzing SELinux denials.
CHAPTER 4. 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.

4.1. 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:

   1. Switch to permissive mode:
2. Repeat your scenario.

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

4.2. 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 ****************************

[trimmed for clarity]

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

[trimmed for clarity]

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.

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 SETLinux denials in the Audit log.

Additional resources
- The `sealert(8)` man page.

4.3. 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 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/
```

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
ftp_anon_write --> off
ftp_full_access --> off
ftp_use_cifs --> off
ftp_use_nfs --> off
ftp_connect_db --> off
http_enable_ftpserver --> 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`.

### 4.4. 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
```

**SELinux denied the httpd process with PID 2465 and the httpd_t type to read from a directory with the samba_share_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:
• \{ \texttt{getattr} \} - the \texttt{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 \texttt{getattr}, \texttt{read}, and \texttt{write}.

• \texttt{path=\"/var/www/html/file1\"} - the path to the object (target) the process attempted to access.

• \texttt{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 \texttt{httpd\_t} type.

• \texttt{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 \texttt{file1}.

This SELinux denial can be translated to:

\emph{SELinux denied the \texttt{httpd} process with PID 2465 to access the \texttt{/var/www/html/file1} file with the \texttt{samba\_share\_t} type, which is not accessible to processes running in the \texttt{httpd\_t} domain unless configured otherwise.}

Additional resources

• For more information, see the \texttt{auditd(8)} and \texttt{ausearch(8)} man pages.

4.5. RELATED INFORMATION

• The Basic SELinux Troubleshooting in CLI article on the Customer Portal.

• The What is SELinux trying to tell me? The 4 key causes of SELinux errors presentation on Fedora People
CHAPTER 5. CREATING SELINUX POLICIES FOR CONTAINERS

RHEL 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.

5.1. 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

- For more details on the process of generating an SELinux policy with udica, see the Generate SELinux policies for containers with udica Red Hat Blog article.

5.2. 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
   ```

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. 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
   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:

```bash
# podman stop 37a3635af8b8f
# 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 steps

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

```bash
# 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:

```bash
[root@37a3635af8b8f ~]# cd /home
[root@37a3635af8b8f home]# ls
username
[root@37a3635af8b8f ~]# cd /var/spool/
[root@37a3635af8b8f spool]# touch test
[root@37a3635af8b8f spool]#
```

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

```bash
[root@37a3635af8b8f ~]# yum install nmap-ncat
[root@37a3635af8b8f ~]# nc -lvp 21
Ncat: Version 7.60 (https://nmap.org/ncat)
Ncat: Generating a temporary 1024-bit RSA key. Use --ssl-key and --ssl-cert to use a permanent one.
Ncat: SHA-1 fingerprint: 6EEC 102E 6666 5F96 CC4F E5FA A1BE 4A5E 6C76 B6DC
Ncat: Listening on :::21
Ncat: Listening on 0.0.0.0:21

[root@37a3635af8b8f ~]# nc -lvp 80
Ncat: Version 7.60 (https://nmap.org/ncat)
Ncat: Generating a temporary 1024-bit RSA key. Use --ssl-key and --ssl-cert to use a permanent one.
Ncat: SHA-1 fingerprint: 6EEC 102E 6666 5F96 CC4F E5FA A1BE 4A5E 6C76 B6DC
Ncat: bind to :::80: Permission denied. QUITTING.
```

### Additional resources

- For more information, see the `udica(8)` and `podman(1)` man pages.

### 5.3. ADDITIONAL RESOURCES

- For more details on creating policies with `udica`, see the `udica - Generate SELinux policies for containers` page.