Red Hat Enterprise Linux 7 SELinux User's and Administrator's Guide

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

Barbora Ančincová Tomáš Čapek

Red Hat Enterprise Linux 7 SELinux User's and Administrator's Guide

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

Barbora Ančincová
Red Hat Engineering Content Services
bancinco@redhat.com

Tomáš Čapek
Red Hat Engineering Content Services
tcapek@redhat.com

Legal Notice

Copyright © 2013 Red Hat, Inc.

This document is licensed by Red Hat under the Creative Commons Attribution-ShareAlike 3.0 Unported License. If you distribute this document, or a modified version of it, you must provide attribution to Red Hat, Inc. and provide a link to the original. If the document is modified, all Red Hat trademarks must be removed.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, MetaMatrix, Fedora, the Infinity Logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack Logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

This books consists of two parts: SELinux and Managing Confined Services. The former describes the basics and principles upon which SELinux functions, the latter is more focused on practical tasks to set up and configure various services.

Table of Contents

Part I. SELinux

Chapter 1. Introduction

Security-Enhanced Linux (SELinux) is an implementation of a *mandatory access control* mechanism in the Linux kernel, checking for allowed operations after standard *discretionary access controls* are checked. It was created by the National Security Agency and can enforce rules on files and processes in a Linux system, and on their actions, based on defined policies.

When using SELinux, files, including directories and devices, are referred to as objects. Processes, such as a user running a command or the Mozilla Firefox application, are referred to as subjects. Most operating systems use a Discretionary Access Control (DAC) system that controls how subjects interact with objects, and how subjects interact with each other. On operating systems using DAC, users control the permissions of files (objects) that they own. For example, on Linux operating systems, users could make their home directories world-readable, giving users and processes (subjects) access to potentially sensitive information, with no further protection over this unwanted action.

Relying on DAC mechanisms alone is fundamentally inadequate for strong system security. DAC access decisions are only based on user identity and ownership, ignoring other security-relevant information such as the role of the user, the function and trustworthiness of the program, and the sensitivity and integrity of the data. Each user typically has complete discretion over their files, making it difficult to enforce a system-wide security policy. Furthermore, every program run by a user inherits all of the permissions granted to the user and is free to change access to the user's files, so minimal protection is provided against malicious software. Many system services and privileged programs run with coarse-grained privileges that far exceed their requirements, so that a flaw in any one of these programs could be exploited to obtain further system access. [1]

The following is an example of permissions used on Linux operating systems that do not run Security-Enhanced Linux (SELinux). The permissions and output in these examples may differ slightly from your system. Use the following command to view file permissions:

```
~]$ ls -l file1
-rwxrw-r-- 1 user1 group1 0 2009-08-30 11:03 file1
```

In this example, the first three permission bits, **rwx**, control the access the Linux **user1** user (in this case, the owner) has to **file1**. The next three permission bits, **rw-**, control the access the Linux **group1** group has to **file1**. The last three permission bits, **r--**, control the access everyone else has to **file1**, which includes all users and processes.

Security-Enhanced Linux (SELinux) adds Mandatory Access Control (MAC) to the Linux kernel, and is enabled by default in Red Hat Enterprise Linux. A general purpose MAC architecture needs the ability to enforce an administratively-set security policy over all processes and files in the system, basing decisions on labels containing a variety of security-relevant information. When properly implemented, it enables a system to adequately defend itself and offers critical support for application security by protecting against the tampering with, and bypassing of, secured applications. MAC provides strong separation of applications that permits the safe execution of untrustworthy applications. Its ability to limit the privileges associated with executing processes limits the scope of potential damage that can result from the exploitation of vulnerabilities in applications and system services. MAC enables information to be protected from legitimate users with limited authorization as well as from authorized users who have unwittingly executed malicious applications. [2]

The following is an example of the labels containing security-relevant information that are used on processes, Linux users, and files, on Linux operating systems that run SELinux. This information is called the SELinux *context*, and is viewed using the following command:

```
~]$ ls -Z file1
-rwxrw-r--  user1 group1 unconfined_u:object_r:user_home_t:s0      file1
```

In this example, SELinux provides a user (**`unconfined_u`**), a role (**`object_r`**), a type (**`user_home_t`**), and a level (**`s0`**). This information is used to make access control decisions. With DAC, access is controlled based only on Linux user and group IDs. It is important to remember that SELinux policy rules are checked *after* DAC rules. SELinux policy rules are not used if DAC rules deny access first.

Linux and SELinux Users

On Linux operating systems that run SELinux, there are Linux users as well as SELinux users. SELinux users are part of SELinux policy. Linux users are mapped to SELinux users. To avoid confusion, this guide uses *Linux user* and *SELinux user* to differentiate between the two.

1.1. Benefits of running SELinux

- All processes and files are labeled with a type. A type defines a domain for processes, and a type for files. Processes are separated from each other by running in their own domains, and 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 level.
- SELinux policy is administratively-defined, enforced system-wide, and is not set at user discretion.
- Reduced vulnerability to 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,
- a replacement for passwords, firewalls, or other security systems,
- an 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, firewalls, and so on.

1.2. 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 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 files and applications in their home directory. If configured, this prevents users from executing malicious files from their home directories.
- Process separation is used. Processes run in their own domains, preventing processes from accessing files used by other processes, as well as preventing processes from accessing other processes. 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 databases used by MariaDB.
- SELinux helps limit 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 [3] from being updated via zone transfers, by the BIND **named** daemon itself, and by other processes.
- Refer to the NetworkWorld.com article, A seatbelt for server software: SELinux blocks real-world exploits [4], for background information about SELinux, and information about various exploits that SELinux has prevented.

1.3. SELinux Architecture

SELinux is a Linux security module that is built into the Linux kernel. SELinux is driven by loadable policy rules. When security-relevant access is taking place, such as when a process attempts to open a file, the operation is intercepted in the kernel by SELinux. If an SELinux policy rule 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.

1.4. SELinux Modes

SELinux has three modes:

- Enforcing: SELinux policy is enforced. SELinux denies access based on SELinux policy rules.
- Permissive: SELinux policy is not enforced. SELinux does not deny access, but denials are logged for actions that would have been denied if running in enforcing mode.
- Disabled: SELinux is disabled. Only DAC rules are used.

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, as the Linux root user, run the **`setenforce 1`** command. To change to permissive mode, run the **`setenforce 0`** command. Use the **`getenforce`** utility to view the current SELinux mode.

Persistent mode changes are covered in Section 4.4, “Enabling and Disabling SELinux”.

1.5. What Is New in Red Hat Enterprise Linux 7

This section provides a brief overview of new SELinux features that have been introduced in Red Hat Enterprise Linux 7. For each feature described below, a link is provided to full documentation in the book.

File Name Transition

Previously, creating a specified object class within a directory with a different label than the directory's one could cause problems related to incorrect labeling. With the *file name transition* feature, policy writers can now specify the file name when writing rules. Now, it is possible to write a rule that states, if a process labeled **`A_t`** creates a specified object class in a directory labeled **`B_t`** and the specified object class is named **`objectname`**, it gets the label **`C_t`**. This mechanism decreases problems with incorrect labeling and provides more fine-grained control over system processes.

See the Section 4.12, “File Name Transition” for more information about **`File Name Transition`**.

OpenShift

The OpenShift Platform as a Service (PaaS) has been added to the list of SELinux confined services. OpenShift provides a platform in the cloud where developers and teams can build, test, deploy, and run their applications. See Chapter 25, *OpenShift* for more information about SELinux configuration of OpenShift.

Identity Management

Identity Management (IdM) has been added to the list of SELinux confined services. IdM provides centralized authentication, authorization, and account information by storing data about user, groups, hosts, and other objects necessary to manage the security aspects of a network of computers. See Chapter 26, *Identity Management* for more information about SELinux configuration of IdM.

Disable ptrace()

The new **`deny_ptrace`** Boolean has been added to the list of SELinux Booleans. This Boolean allows the user to disable the **`ptrace()`** system call. See Section 4.13, “Disable ptrace()” for more information about **`deny_ptrace`**.

New Confined Domains

With Red Hat Enterprise Linux 7, many products and services use their separate confined domains. For example, SELinux now supports:

OpenStack

OpenStack is a cloud computing platform which consists of various components, such as Swift, Nova, or Glance. The **`swift_t`**, **`nova_*`**, and **`glance_*`** domains have been added to the SELinux policy.

OpenShift

OpenShift is a cloud computing platform. The **openshift_*** domains have been added to the SELinux policy.

realmd

The **realmd** utility is a DBus service that manages discovery and enrollment in realms and domains such as Active Directory or Identity Management. The **realmd_t** domain has been added to the SELinux policy.

glusterd

The **glusterd** utility is a volume management daemon. The **glusterd_t** domain has been added to the SELinux policy.

stapserver

The **stapserver** utility provides an instrumentation system server. The **stapserver_t** domain has been added to the SELinux policy.

OpenLMI

OpenLMI (Open Linux Management Infrastructure) provides a common infrastructure for the management of Linux systems. The **pegasus_openlmi_*** domains have been added to the SELinux policy.

To learn more about the confined domains, see Chapter 3, *Targeted Policy*. See Section 10.3.3, “Manual Pages for Services” for more information about manual pages for services.

Shrinking Policy

Previously, M4 macros were used to specify policy rules, the SELinux policy was consuming a large amount of kernel memory, and it took a long time to load the policy during boot. In Red Hat Enterprise Linux 7, attributes that allow grouping types together are used instead of macros to simplify the SELinux policy and to shrink its size. For example, some SELinux domains, mainly for cluster and antivirus services, have been merged together:

- The **amavis.pp** and **clamav.pp** modules have been consolidated into the **antivirus.pp** module and aliased to the **antivirus_t** type:

```
typealias antivirus_t alias { amavis_t clamd_t clamscan_t freshclam_t }
```

- The **pacemaker.pp**, **corosync.pp**, and **aisexec.pp** modules have been consolidated into the **rhcs.pp** module and aliased to the **cluster_t** type:

```
typealias cluster_t alias { aisexec_t corosync_t pacemaker_t rgmanager_t }
```

Size of the SELinux policy has been reduced by 80 percent, its loading time is now significantly shorter, and it consumes less kernel memory during boot.

Pre-built Policy

The *selinux-policy* package now contains a pre-built policy; the SELinux policy is recompiled after installation only if the user made any local customizations to the policy. This change speeds up the package installation process and lowers the memory usage.

Secure Linux Containers

Linux Containers is a low-level virtualization feature that allows you to run multiple copies of the same service at the same time on a system. When creating a secure Linux container, the **`virt-sandbox-service`** utility uses a systemd unit file to properly set up a service within the container. The **`libvirt`** library then launches the container with the SELinux context specified in the SELinux policy.

To learn more about secure Linux containers, see Chapter 8, *Secure Linux Containers*.

The sepolicy Suite

The **`sepolicy`** utility provides a suite of features to query the installed SELinux policy. The suite allows you to generate transition reports, man pages, or even new policy modules, thus giving users easier access and better understanding of the SELinux policy.

To learn more about the **`sepolicy`** inspection suite, see Chapter 5, *The **`sepolicy`** Suite*.

Thumbnail Protection

Previously, the thumbnail drivers were not locked when the screen was locked. Consequently, it was possible to use the thumbnail driver code to bypass the lock screen without entering a password. A new SELinux policy has been introduced to prevent this type of attack, improving system security. See Section 4.14, “Thumbnail Protection” for more information about thumbnail protection.

Disabling Permissive Domains

All permissive domain declarations have been re-factored into the new **`permissivedomains.pp`** module. Consequently, it is now possible to disable all permissive domains using the **`semodule`** utility. See Section 10.3.4.2, “Disabling Permissive Domains” for more information about this module.

Labeled NFS

Passing SELinux labels between an NFS client and server is now supported. The new labeled NFS feature allows users to differentiate various SELinux labels on a single NFS volume, which provides better security control of confined domains accessing NFS volumes.

See Section 16.1, “NFS and SELinux” for more information about labeled NFS.

SELinux Systemd Access Control

In Red Hat Enterprise Linux 7, the **`systemd`** daemon manages all calls to start and stop system services. The **`systemd`** method calls are mapped to SELinux access checks, which provides consistent SELinux labels when the services are started either automatically or manually.

To learn more about SELinux and **`systemd`** integration, see Chapter 9, *SELinux **`systemd`** Access Control*.

[1] "Integrating Flexible Support for Security Policies into the Linux Operating System", by Peter Loscocco and Stephen Smalley. This paper was originally prepared for the National Security Agency and is, consequently, in the public domain. Refer to the original paper for details and the document as it was first released. Any edits and changes were done by Murray McAllister.

[2] "Meeting Critical Security Objectives with Security-Enhanced Linux", by Peter Loscocco and Stephen Smalley. This paper was originally prepared for the National Security Agency and is, consequently, in the public domain. Refer to the original paper for details and the document as it was first released. Any edits and changes were done by Murray McAllister.

[3] Text files that include information, such as host name to IP address mappings, that are used by DNS servers.

[4] Marti, Don. "A seatbelt for server software: SELinux blocks real-world exploits". Published 24 February 2008. Accessed 27 August 2009: http://www.networkworld.com/news/2008/022408-selinux.html.

Chapter 2. SELinux Contexts

Processes and files are labeled with an SELinux context that contains additional information, such as an SELinux user, role, type, and, optionally, a level. When running SELinux, all of this information is used to make access control decisions. In Red Hat Enterprise Linux, SELinux provides a combination of Role-Based Access Control (RBAC), Type Enforcement (TE), and, optionally, Multi-Level Security (MLS).

The following is an example showing SELinux context. SELinux contexts are used on processes, Linux users, and files, on Linux operating systems that run SELinux. Use the following command to view the SELinux context of files and directories:

```
~]$ ls -Z file1
-rwxrw-r--  user1 group1 unconfined_u:object_r:user_home_t:s0      file1
```

SELinux contexts follow the *SELinux user:role:type:level* syntax. The fields are as follows:

SELinux user

The SELinux user identity is an identity known to the policy that is authorized for a specific set of roles, and for a specific MLS/MCS range. Each Linux user is mapped to an SELinux user via SELinux policy. This allows Linux users to inherit the restrictions placed on SELinux users. The mapped SELinux user identity is used in the SELinux context for processes in that session, in order to define what roles and levels they can enter. Run the following command as root to view a list of mappings between SELinux and Linux user accounts (you need to have the *policycoreutils-python* package installed):

```
~]# 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       *
system_u             system_u             s0-s0:c0.c1023       *
```

Output may differ slightly from system to system:

- The **`Login Name`** column lists Linux users.
- The **`SELinux User`** column lists which SELinux user the Linux user is mapped to. For processes, the SELinux user limits which roles and levels are accessible.
- The **`MLS/MCS Range`** column, is the level used by Multi-Level Security (MLS) and Multi-Category Security (MCS).
- The **`Service`** column determines the correct SELinux context, in which the Linux user is supposed to be logged in to the system. By default, the asterisk (**`*`**) character is used, which stands for any service.

role

Part of SELinux is the Role-Based Access Control (RBAC) security model. The role is an attribute of RBAC. SELinux users are authorized for roles, and roles are authorized for domains. The role serves as an intermediary between domains and SELinux users. The roles that can be entered determine which domains can be entered; ultimately, this controls which object types can be accessed. This helps reduce vulnerability to privilege escalation attacks.

type

The type is an attribute of Type Enforcement. The type defines a domain for processes, and a type for files. SELinux policy rules define how types can access each other, whether it be a domain accessing a type, or a domain accessing another domain. Access is only allowed if a specific SELinux policy rule exists that allows it.

level

The level is an attribute of MLS and MCS. An MLS range is a pair of levels, written as *lowlevel-highlevel* if the levels differ, or *lowlevel* if the levels are identical (**s0-s0** is the same as **s0**). Each level is a sensitivity-category pair, with categories being optional. If there are categories, the level is written as *sensitivity:category-set*. If there are no categories, it is written as *sensitivity*.

If the category set is a contiguous series, it can be abbreviated. For example, **c0.c3** is the same as **c0,c1,c2,c3**. The **/etc/selinux/targeted/setrans.conf** file maps levels (**s0:c0**) to human-readable form (that is **CompanyConfidential**). In Red Hat Enterprise Linux, targeted policy enforces MCS, and in MCS, there is just one sensitivity, **s0**. MCS in Red Hat Enterprise Linux supports 1024 different categories: **c0** through to **c1023**. **s0-s0:c0.c1023** is sensitivity **s0** and authorized for all categories.

MLS enforces the Bell-La Padula Mandatory Access Model, and is used in Labeled Security Protection Profile (LSPP) environments. To use MLS restrictions, install the *selinux-policy-mls* package, and configure MLS to be the default SELinux policy. The MLS policy shipped with Red Hat Enterprise Linux omits many program domains that were not part of the evaluated configuration, and therefore, MLS on a desktop workstation is unusable (no support for the X Window System); however, an MLS policy from the upstream SELinux Reference Policy can be built that includes all program domains. For more information on MLS configuration, refer to Section 4.11, “Multi-Level Security (MLS)”.

2.1. Domain Transitions

A process in one domain transitions to another domain by executing an application that has the **entrypoint** type for the new domain. The **entrypoint** permission is used in SELinux policy and controls which applications can be used to enter a domain. The following example demonstrates a domain transition:

Procedure 2.1. An Example of a Domain Transition

1. A user wants to change their password. To do this, they run the **passwd** utility. The **/usr/bin/passwd** executable is labeled with the **passwd_exec_t** type:

   ```
   ~]$ ls -Z /usr/bin/passwd
   -rwsr-xr-x  root root system_u:object_r:passwd_exec_t:s0 /usr/bin/passwd
   ```

   The **passwd** utility accesses **/etc/shadow**, which is labeled with the **shadow_t** type:

   ```
   ~]$ ls -Z /etc/shadow
   -r--------. root root system_u:object_r:shadow_t:s0    /etc/shadow
   ```

2. An SELinux policy rule states that processes running in the **passwd_t** domain are allowed to read and write to files labeled with the **shadow_t** type. The **shadow_t** type is only applied to files that are required for a password change. This includes **/etc/gshadow**, **/etc/shadow**, and their backup files.

3. An SELinux policy rule states that the **passwd_t** domain has **entrypoint** permission to the **passwd_exec_t** type.

4. When a user runs the **passwd** utility, the user's shell process transitions to the **passwd_t** domain. With SELinux, since the default action is to deny, and a rule exists that allows (among other things) applications running in the **passwd_t** domain to access files labeled with the **shadow_t** type, the **passwd** application is allowed to access **/etc/shadow**, and update the user's password.

This example is not exhaustive, and is used as a basic example to explain domain transition. Although there is an actual rule that allows subjects running in the **passwd_t** domain to access objects labeled with the **shadow_t** file type, other SELinux policy rules must be met before the subject can transition to a new domain. In this example, Type Enforcement ensures:

- The **passwd_t** domain can only be entered by executing an application labeled with the **passwd_exec_t** type; can only execute from authorized shared libraries, such as the **lib_t** type; and cannot execute any other applications.

- Only authorized domains, such as **passwd_t**, can write to files labeled with the **shadow_t** type. Even if other processes are running with superuser privileges, those processes cannot write to files labeled with the **shadow_t** type, as they are not running in the **passwd_t** domain.

- Only authorized domains can transition to the **passwd_t** domain. For example, the **sendmail** process running in the **sendmail_t** domain does not have a legitimate reason to execute **passwd**; therefore, it can never transition to the **passwd_t** domain.

- Processes running in the **passwd_t** domain can only read and write to authorized types, such as files labeled with the **etc_t** or **shadow_t** types. This prevents the **passwd** application from being tricked into reading or writing arbitrary files.

2.2. SELinux Contexts for Processes

Use the **ps -eZ** command to view the SELinux context for processes. For example:

Procedure 2.2. Viewing the SELinux Context for the passwd Utility

1. Open a terminal, such as **Applications** → **System Tools** → **Terminal**.

2. Run the **passwd** utility. Do not enter a new password:

```
~]$ passwd
Changing password for user user_name.
Changing password for user_name.
(current) UNIX password:
```

3. Open a new tab, or another terminal, and run the following command. The output is similar to the following:

```
~]$ ps -eZ | grep passwd
unconfined_u:unconfined_r:passwd_t:s0-s0:c0.c1023 13212 pts/1 00:00:00 passwd
```

4. In the first tab/terminal, press **Ctrl+C** to cancel the **passwd** utility.

In this example, when the **passwd** utility (labeled with the **passwd_exec_t** type) is executed, the user's shell process transitions to the **passwd_t** domain. Remember that the type defines a domain for processes, and a type for files.

To view the SELinux contexts for all running processes, run the **ps** utility again. Note that below is a truncated example of the output, and may differ on your system:

```
]$ ps -eZ
system_u:system_r:dhcpc_t:s0             1869 ?  00:00:00 dhclient
system_u:system_r:sshd_t:s0-s0:c0.c1023  1882 ?  00:00:00 sshd
system_u:system_r:gpm_t:s0               1964 ?  00:00:00 gpm
system_u:system_r:crond_t:s0-s0:c0.c1023 1973 ?  00:00:00 crond
system_u:system_r:kerneloops_t:s0        1983 ?  00:00:05 kerneloops
system_u:system_r:crond_t:s0-s0:c0.c1023 1991 ?  00:00:00 atd
```

The **system_r** role is used for system processes, such as daemons. Type Enforcement then separates each domain.

2.3. SELinux Contexts for Users

Use the following command to view the SELinux context associated with your Linux user:

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

In Red Hat Enterprise Linux, Linux users run unconfined by default. This SELinux context shows that the Linux user is mapped to the SELinux **unconfined_u** user, running as the **unconfined_r** role, and is running in the **unconfined_t** domain. **s0-s0** is an MLS range, which in this case, is the same as just **s0**. The categories the user has access to is defined by **c0.c1023**, which is all categories (**c0** through to **c1023**).

Chapter 3. Targeted Policy

Targeted policy is the default SELinux policy used in Red Hat Enterprise Linux. When using targeted policy, processes that are targeted run in a confined domain, and processes that are not targeted run in an unconfined domain. For example, by default, logged-in users run in the **`unconfined_t`** domain, and system processes started by init run in the **`initrc_t`** domain; both of these domains are unconfined.

Unconfined domains (as well as confined domains) are subject to executable and writeable memory checks. By default, subjects running in an unconfined domain cannot allocate writeable memory and execute it. This reduces vulnerability to buffer overflow attacks. These memory checks are disabled by setting Booleans, which allow the SELinux policy to be modified at runtime. Boolean configuration is discussed later.

3.1. Confined Processes

Almost every service that listens on a network, such as **`sshd`** or **`httpd`**, is confined in Red Hat Enterprise Linux. Also, most processes that run as the root user and perform tasks for users, such as the **`passwd`** utility, are confined. When a process is confined, it runs in its own domain, such as the **`httpd`** process running in the **`httpd_t`** domain. If a confined process is compromised by an attacker, depending on SELinux policy configuration, an attacker's access to resources and the possible damage they can do is limited.

Complete this procedure to ensure that SELinux is enabled and the system is prepared to perform the following example:

Procedure 3.1. How to Verify SELinux Status

1. Confirm that SELinux is enabled, is running in enforcing mode, and that targeted policy is being used. The correct output should look similar to the output below:

```
~]$ sestatus
SELinux status:                 enabled
SELinuxfs mount:                /selinux
Current mode:                   enforcing
Mode from config file:          enforcing
Policy version:                 24
Policy from config file:        targeted
```

   See Section 4.4, “Enabling and Disabling SELinux” for detailed information about enabling and disabling SELinux.

2. As root, create a file in the **`/var/www/html/`** directory:

```
~]# touch /var/www/html/testfile
```

3. Run the following command to view the SELinux context of the newly created file:

```
~]$ ls -Z /var/www/html/testfile
-rw-r--r--  root root unconfined_u:object_r:httpd_sys_content_t:s0
/var/www/html/testfile
```

   By default, Linux users run unconfined in Red Hat Enterprise Linux, which is why the **`testfile`** file is labeled with the SELinux **`unconfined_u`** user. RBAC is used for processes, not files. Roles do not have a meaning for files; the **`object_r`** role is a generic role used for files (on persistent storage and network file systems). Under the **/proc/** directory, files related to processes may use the **system_r** role. [5] The **httpd_sys_content_t** type allows the **httpd** process to access this file.

The following example demonstrates how SELinux prevents the Apache HTTP Server (**httpd**) from reading files that are not correctly labeled, such as files intended for use by Samba. This is an example, and should not be used in production. It assumes that the *httpd* and *wget* packages are installed, the SELinux targeted policy is used, and that SELinux is running in enforcing mode.

Procedure 3.2. An Example of Confined Process

1. As root, start the **httpd** daemon:

   ```
   ~]# systemctl start httpd.service
   ```

   Confirm that the service is running. The output should include the information below (only the time stamp will differ):

   ```
   ~]$ systemctl status httpd.service
   httpd.service - The Apache HTTP Server
      Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled)
      Active: active (running) since Mon 2013-08-05 14:00:55 CEST; 8s ago
   ```

2. Change into a directory where your Linux user has write access to, and run the following command. Unless there are changes to the default configuration, this command succeeds:

   ```
   ~]$ wget http://localhost/testfile
   --2009-11-06 17:43:01--  http://localhost/testfile
   Resolving localhost... 127.0.0.1
   Connecting to localhost|127.0.0.1|:80... connected.
   HTTP request sent, awaiting response... 200 OK
   Length: 0 [text/plain]
   Saving to: `testfile'

   [ <=>                              ] 0      --.-K/s   in 0s

   2009-11-06 17:43:01 (0.00 B/s) - `testfile' saved [0/0]
   ```

3. The **chcon** command relabels files; however, such label changes do not survive when the file system is relabeled. For permanent changes that survive a file system relabel, use the **semanage** utility, which is discussed later. As root, run the following command to change the type to a type used by Samba:

   ```
   ~]# chcon -t samba_share_t /var/www/html/testfile
   ```

   Run the following command to view the changes:

   ```
   ~]$ ls -Z /var/www/html/testfile
   -rw-r--r--  root root unconfined_u:object_r:samba_share_t:s0
   /var/www/html/testfile
   ```

4. Note that the current DAC permissions allow the **httpd** process access to **testfile**. Change into a directory where your user has write access to, and run the following command. Unless there are changes to the default configuration, this command fails:

   ```
   ~]$ wget http://localhost/testfile
   --2009-11-06 14:11:23--  http://localhost/testfile
   ```

```
Resolving localhost... 127.0.0.1
Connecting to localhost|127.0.0.1|:80... connected.
HTTP request sent, awaiting response... 403 Forbidden
2009-11-06 14:11:23 ERROR 403: Forbidden.
```

5. As root, remove **testfile**:

   ```
   ~]# rm -i /var/www/html/testfile
   ```

6. If you do not require **httpd** to be running, as root, run the following command to stop it:

   ```
   ~]# systemctl stop httpd.service
   ```

This example demonstrates the additional security added by SELinux. Although DAC rules allowed the **httpd** process access to **testfile** in step 2, because the file was labeled with a type that the **httpd** process does not have access to, SELinux denied access.

If the **auditd** daemon is running, an error similar to the following is logged to **/var/log/audit/audit.log**:

```
type=AVC msg=audit(1220706212.937:70): avc:  denied  { getattr } for  pid=1904
comm="httpd" path="/var/www/html/testfile" dev=sda5 ino=247576
scontext=unconfined_u:system_r:httpd_t:s0
tcontext=unconfined_u:object_r:samba_share_t:s0  tclass=file

type=SYSCALL msg=audit(1220706212.937:70): arch=40000003 syscall=196 success=no
exit=-13 a0=b9e21da0 a1=bf9581dc a2=555ff4 a3=2008171 items=0 ppid=1902 pid=1904
auid=500 uid=48 gid=48 euid=48 suid=48 fsuid=48 egid=48 sgid=48 fsgid=48 tty=(none)
ses=1 comm="httpd" exe="/usr/sbin/httpd" subj=unconfined_u:system_r:httpd_t:s0 key=
(null)
```

Also, an error similar to the following is logged to **/var/log/httpd/error_log**:

```
[Wed May 06 23:00:54 2009] [error] [client 127.0.0.1] (13)Permission denied: access
to /testfile denied
```

3.2. Unconfined Processes

Unconfined processes run in unconfined domains, for example, init programs run in the unconfined **initrc_t** domain, unconfined kernel processes run in the **kernel_t** domain, and unconfined Linux users run in the **unconfined_t** domain. For unconfined processes, SELinux policy rules are applied, but policy rules exist that allow processes running in unconfined domains almost all access. Processes running in unconfined domains fall back to using DAC rules exclusively. If an unconfined process is compromised, SELinux does not prevent an attacker from gaining access to system resources and data, but of course, DAC rules are still used. SELinux is a security enhancement on top of DAC rules – it does not replace them.

To ensure that SELinux is enabled and the system is prepared to perform the following example, complete the Procedure 3.1, “How to Verify SELinux Status” described in Section 3.1, “Confined Processes”.

The following example demonstrates how the Apache HTTP Server (**httpd**) can access data intended for use by Samba, when running unconfined. Note that in Red Hat Enterprise Linux, the **httpd** process runs in the confined **httpd_t** domain by default. This is an example, and should not be used in production. It assumes that the *httpd*, *wget*, *dbus* and *audit* packages are installed, that the SELinux targeted policy is used, and that SELinux is running in enforcing mode.

Procedure 3.3. An Example of Unconfined Process

1. The **chcon** command relabels files; however, such label changes do not survive when the file system is relabeled. For permanent changes that survive a file system relabel, use the **semanage** utility, which is discussed later. As the root user, run the following command to change the type to a type used by Samba:

   ```
   ~]# chcon -t samba_share_t /var/www/html/testfile
   ```

   View the changes:

   ```
   ~]$ ls -Z /var/www/html/testfile
   -rw-r--r--  root root unconfined_u:object_r:samba_share_t:s0
   /var/www/html/testfile
   ```

2. Run the following command to confirm that the **httpd** process is not running:

   ```
   ~]$ systemctl status httpd.service
   httpd.service - The Apache HTTP Server
      Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled)
      Active: inactive (dead)
   ```

   If the output differs, run the following command as root to stop the **httpd** process:

   ```
   ~]# systemctl stop httpd.service
   ```

3. To make the **httpd** process run unconfined, run the following command as root to change the type of the **/usr/sbin/httpd** file, to a type that does not transition to a confined domain:

   ```
   ~]# chcon -t unconfined_exec_t /usr/sbin/httpd
   ```

4. Confirm that **/usr/sbin/httpd** is labeled with the **unconfined_exec_t** type:

   ```
   ~]$ ls -Z /usr/sbin/httpd
   -rwxr-xr-x  root root system_u:object_r:unconfined_exec_t:s0 /usr/sbin/httpd
   ```

5. As root, start the **httpd** process and confirm, that it started successfully:

   ```
   ~]# systemctl start httpd.service
   ```

   ```
   ~]# systemctl status httpd.service
   httpd.service - The Apache HTTP Server
      Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled)
      Active: active (running) since Thu 2013-08-15 11:17:01 CEST; 5s ago
   ```

6. Run the following command to view **httpd** running in the **unconfined_t** domain:

   ```
   ~]$ ps -eZ | grep httpd
   unconfined_u:unconfined_r:unconfined_t:s0 7721 ?      00:00:00 httpd
   unconfined_u:unconfined_r:unconfined_t:s0 7723 ?      00:00:00 httpd
   unconfined_u:unconfined_r:unconfined_t:s0 7724 ?      00:00:00 httpd
   unconfined_u:unconfined_r:unconfined_t:s0 7725 ?      00:00:00 httpd
   unconfined_u:unconfined_r:unconfined_t:s0 7726 ?      00:00:00 httpd
   ```

```
unconfined_u:unconfined_r:unconfined_t:s0 7727 ?      00:00:00 httpd
unconfined_u:unconfined_r:unconfined_t:s0 7728 ?      00:00:00 httpd
unconfined_u:unconfined_r:unconfined_t:s0 7729 ?      00:00:00 httpd
unconfined_u:unconfined_r:unconfined_t:s0 7730 ?      00:00:00 httpd
```

7. Change into a directory where your Linux user has write access to, and run the following command. Unless there are changes to the default configuration, this command succeeds:

```
~]$ wget http://localhost/testfile
--2009-05-07 01:41:10--  http://localhost/testfile
Resolving localhost... 127.0.0.1
Connecting to localhost|127.0.0.1|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 0 [text/plain]
Saving to: `testfile.1'

[ <=>                            ]--.-K/s   in 0s

2009-05-07 01:41:10 (0.00 B/s) - `testfile.1' saved [0/0]
```

   Although the **httpd** process does not have access to files labeled with the **samba_share_t** type, **httpd** is running in the unconfined **unconfined_t** domain, and falls back to using DAC rules, and as such, the **wget** command succeeds. Had **httpd** been running in the confined **httpd_t** domain, the **wget** command would have failed.

8. The **restorecon** utility restores the default SELinux context for files. As root, run the following command to restore the default SELinux context for **/usr/sbin/httpd**:

```
~]# restorecon -v /usr/sbin/httpd
restorecon reset /usr/sbin/httpd context
system_u:object_r:unconfined_exec_t:s0->system_u:object_r:httpd_exec_t:s0
```

   Confirm that **/usr/sbin/httpd** is labeled with the **httpd_exec_t** type:

```
~]$ ls -Z /usr/sbin/httpd
-rwxr-xr-x  root root system_u:object_r:httpd_exec_t:s0 /usr/sbin/httpd
```

9. As root, run the following command to restart **httpd**. After restarting, confirm that **httpd** is running in the confined **httpd_t** domain:

```
~]# systemctl restart httpd.service
```

```
~]$ ps -eZ | grep httpd
system_u:system_r:httpd_t:s0    8883 ?        00:00:00 httpd
system_u:system_r:httpd_t:s0    8884 ?        00:00:00 httpd
system_u:system_r:httpd_t:s0    8885 ?        00:00:00 httpd
system_u:system_r:httpd_t:s0    8886 ?        00:00:00 httpd
system_u:system_r:httpd_t:s0    8887 ?        00:00:00 httpd
system_u:system_r:httpd_t:s0    8888 ?        00:00:00 httpd
system_u:system_r:httpd_t:s0    8889 ?        00:00:00 httpd
```

10. As root, remove **testfile**:

```
~]# rm -i /var/www/html/testfile
rm: remove regular empty file `/var/www/html/testfile'? y
```

11. If you do not require **httpd** to be running, as root, run the following command to stop **httpd**:

```
~]# systemctl stop httpd.service
```

The examples in these sections demonstrate how data can be protected from a compromised confined-process (protected by SELinux), as well as how data is more accessible to an attacker from a compromised unconfined-process (not protected by SELinux).

3.3. 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. This Linux user mapping is seen by running 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       *
system_u             system_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
```

The following procedure demonstrates how to add a new Linux user to the system and how to map that user to the SELinux **unconfined_u** user. It assumes that the root user is running unconfined, as it does by default in Red Hat Enterprise Linux:

Procedure 3.4. Mapping a New Linux User to the SELinux unconfined_u User

1. As root, run the following command to create a new Linux user named **newuser**:

   ```
   ~]# useradd newuser
   ```

2. To assign a password to the Linux **newuser** user. Run the following command as root:

   ```
   ~]# passwd newuser
   Changing password for user newuser.
   New UNIX password: Enter a password
   Retype new UNIX password: Enter the same password again
   passwd: all authentication tokens updated successfully.
   ```

3. Log out of your current session, and log in as the Linux **newuser** 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. Run the following command to view the context of a Linux user:

   ```
   [newuser@localhost ~]$ id -Z
   unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
   ```

Note

If you no longer need the **`newuser`** user on your system, log out of the Linux **`newuser`**'s session, log in with your account, and run the **`userdel -r newuser`** command as root. It will remove **`newuser`** along with their home directory.

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

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. However, each confined Linux user is restricted by a confined user domain against the **`unconfined_t`** 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 Linux 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. In the table below, you can see examples of basic confined domains for Linux users in Red Hat Enterprise Linux:

Table 3.1. SELinux User Capabilities

User	Domain	X Window System	su or sudo	Execute in home directory and /tmp/ (default)	Networking
sysadm_u	sysadm_t	yes	**su** and **sudo**	yes	yes
staff_u	staff_t	yes	only **sudo**	yes	yes
user_u	user_t	yes	no	yes	yes
guest_u	guest_t	no	no	no	yes
xguest_u	xguest_t	yes	no	no	Firefox only

- Linux users in the **`user_t`**, **`guest_t`**, **`xguest_t`**, and **`git_shell_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 via the X Window System and a terminal.

- By default, Linux users in the **`guest_t`** and **`xguest_t`** domains cannot execute applications in their home directories or the **`/tmp/`** directory, preventing them from executing applications, which inherit users' permissions, in directories they have write access to. This helps prevent flawed or malicious applications from modifying users' files.

- By default, Linux users in the **`staff_t`** and **`user_t`** domains can execute applications in their home directories and **`/tmp/`**. See Section 6.6, “Booleans for Users Executing Applications” for information about allowing and preventing users from executing applications in their home directories and **`/tmp/`**.

- The only network access Linux users in the **`xguest_t`** domain have is **`Firefox`** connecting to web pages.

---

[5] When using other policies, such as MLS, other roles may be used, for example, **`secadm_r`**.

Chapter 4. Working with SELinux

The following sections give a brief overview of the main SELinux packages in Red Hat Enterprise Linux; installing and updating packages; which log files are used; the main SELinux configuration file; enabling and disabling SELinux; SELinux modes; configuring Booleans; temporarily and persistently changing file and directory labels; overriding file system labels with the **`mount`** command; mounting NFS volumes; and how to preserve SELinux contexts when copying and archiving files and directories.

4.1. SELinux Packages

In Red Hat Enterprise Linux full installation, the SELinux packages are installed by default unless they are manually excluded during installation. If performing a minimal installation in text mode, the *policycoreutils-python* and the *policycoreutils-gui* package are not installed by default. Also, by default, SELinux runs in enforcing mode and the SELinux targeted policy is used. The following SELinux packages are installed on your system by default:

- *policycoreutils* provides utilities such as **`restorecon`**, **`secon`**, **`setfiles`**, **`semodule`**, **`load_policy`**, and **`setsebool`**, for operating and managing SELinux.
- *selinux-policy* provides configuration for the SELinux Reference policy. The SELinux Reference Policy is a complete SELinux policy, and is used as a basis for other policies, such as the SELinux targeted policy; refer to the Tresys Technology SELinux Reference Policy page for further information. This package contains the **`selinux-policy.conf`** file and RPM macros.
- *selinux-policy-targeted* provides the SELinux targeted policy.
- *libselinux* – provides an API for SELinux applications.
- *libselinux-utils* provides the **`avcstat`**, **`getenforce`**, **`getsebool`**, **`matchpathcon`**, **`selinuxconlist`**, **`selinuxdefcon`**, **`selinuxenabled`**, and **`setenforce`** utilities.
- *libselinux-python* provides Python bindings for developing SELinux applications.

The following packages are not installed by default but can be optionally installed by running the **`yum install <package-name>`** command:

- *selinux-policy-devel* provides utilities for creating a custom SELinux policy and policy modules. It also contains manual pages that describe how to configure SELinux altogether with various services.
- *selinux-policy-mls* provides the MLS (Multi-Level Security) SELinux policy.
- *setroubleshoot-server* translates denial messages, produced when access is denied by SELinux, into detailed descriptions that can be viewed with the **`sealert`** utility, also provided in this package.
- *setools-console* provides the Tresys Technology SETools distribution, a number of utilities and libraries for analyzing and querying policy, audit log monitoring and reporting, and file context management. The *setools* package is a meta-package for SETools. The *setools-gui* package provides the **`apol`** and **`seaudit`** utilities. The *setools-console* package provides the **`sechecker`**, **`sediff`**, **`seinfo`**, **`sesearch`**, and **`findcon`** command-line utilities. Refer to the Tresys Technology SETools page for information about these utilities. Note that *setools* and *setools-gui* packages are available only when the Red Hat Network Optional channel is enabled. For further information, see Scope of Coverage Details.

- *mcstrans* translates levels, such as **`s0-s0:c0.c1023`**, to a form that is easier to read, such as **`SystemLow-SystemHigh`**.
- *policycoreutils-python* provides utilities such as **`semanage`**, **`audit2allow`**, **`audit2why`**, and **`chcat`**, for operating and managing SELinux.
- *policycoreutils-gui* provides **`system-config-selinux`**, a graphical utility for managing SELinux.

4.2. Which Log File is Used

In Red Hat Enterprise Linux, the *dbus* and *audit* packages are installed by default, unless they are removed from the default package selection. The *setroubleshoot-server* must be installed via Yum (use the **`yum install setroubleshoot`** command).

If the **`auditd`** daemon is running, an SELinux denial message, such as the following, is written to **`/var/log/audit/audit.log`** by default:

```
type=AVC msg=audit(1223024155.684:49): avc:  denied  { getattr } for  pid=2000
comm="httpd" path="/var/www/html/file1" dev=dm-0 ino=399185
scontext=unconfined_u:system_r:httpd_t:s0 tcontext=system_u:object_r:samba_share_t:s0
tclass=file
```

In addition, a message similar to the one below is written to the **`/var/log/message`** file:

```
May 7 18:55:56 localhost setroubleshoot: SELinux is preventing httpd (httpd_t)
"getattr" to /var/www/html/file1 (samba_share_t). For complete SELinux messages. run
sealert -l de7e30d6-5488-466d-a606-92c9f40d316d
```

In Red Hat Enterprise Linux 7, **`setroubleshootd`** no longer constantly runs as a service. However, it is still used to analyze the AVC messages. Two new programs act as a method to start **`setroubleshoot`** when needed:

- The **`sedispatch`** utility runs as a part of the **`audit`** subsystem. When an AVC denial message is returned, **`sedispatch`** sends a message using **`dbus`**. These messages go straight to **`setroubleshootd`** if it is already running. If it is not running, **`sedispatch`** starts it automatically.
- The **`seapplet`** utility runs in the system toolbar, waiting for dbus messages in **`setroubleshootd`**. It launches the notification bubble, allowing the user to review AVC messages.

Procedure 4.1. Starting Daemons Automatically

1. To configure the **`auditd`** and **`rsyslog`** daemons to automatically start at boot, run the following commands as the root user:

   ```
   ~]# systemctl enable auditd.service
   ```

   ```
   ~]# systemctl enable rsyslog.service
   ```

2. To ensure that the daemons are enabled, type the following commands at the shell prompt:

   ```
   ~]$ systemctl is-enabled auditd
   enabled
   ```

```
~]$ systemctl is-enabled rsyslog
enabled
```

Alternatively, use the **systemctl status *service-name*.service** command and search for the keyword **enabled** in the command output, for example:

```
~]$ systemctl status auditd.service | grep enabled
auditd.service - Security Auditing Service
   Loaded: loaded (/usr/lib/systemd/system/auditd.service; enabled)
```

To learn more on how the **systemd** daemon manages system services, see the Managing System Services chapter in the System Administrator's Guide.

4.3. Main Configuration File

The **/etc/selinux/config** file is the main SELinux configuration file. It controls the SELinux mode and the SELinux policy to use:

```
# 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
```

SELINUX=enforcing

The **SELINUX** option sets the mode SELinux runs in. SELinux has three modes: enforcing, permissive, and disabled. When using enforcing mode, SELinux policy is enforced, and SELinux denies access based on SELinux policy rules. Denial messages are logged. When using permissive mode, SELinux policy is not enforced. SELinux does not deny access, but denials are logged for actions that would have been denied if running SELinux in enforcing mode. When using disabled mode, SELinux is disabled (the SELinux module is not registered with the Linux kernel), and only DAC rules are used.

SELINUXTYPE=targeted

The **SELINUXTYPE** option sets the SELinux policy to use. Targeted policy is the default policy. Only change this option if you want to use the MLS policy. For information on how to enable the MLS policy, refer to Section 4.11.2, “Enabling MLS in SELinux”.

Important

When systems run with SELinux in permissive or disabled mode, users have permission to label fies incorrectly. Also, files created while SELinux is disabled are not labeled. This causes problems when changing to enforcing mode. To prevent incorrectly labeled and unlabeled files from causing problems, file systems are automatically relabeled when changing from disabled mode to permissive or enforcing mode.

4.4. Enabling and Disabling SELinux

Use the **getenforce** or **sestatus** commands to check the status of SELinux. 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:                /selinux
Current mode:                   enforcing
Mode from config file:          enforcing
Policy version:                 24
Policy from config file:        targeted
```

4.4.1. Enabling SELinux

Important

If the system was initially installed without SELinux, particularly the *selinux-policy* package, which was added to the system later, one additional step is necessary to enable SELinux. To make sure SELinux is initialized during system startup, the **dracut** utility has to be run to put SELinux awareness into the initramfs file system. Failing to do so causes SELinux not to start during system startup.

On systems with SELinux disabled, the **SELINUX=disabled** option is configured in **/etc/selinux/config**:

```
# 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
```

Also, the **getenforce** command returns **Disabled**:

```
~]$ getenforce
Disabled
```

Following procedure shows how to enable SELinux:

Procedure 4.2. Enabling SELinux

1. This guide assumes that the following packages are installed:

   - *selinux-policy-targeted*
   - *selinux-policy*
   - *libselinux*
   - *libselinux-python*

- *libselinux-utils*
- *policycoreutils*
- *policycoreutils-python*
- *setroubleshoot*
- *setroubleshoot-server*
- *setroubleshoot-plugins*

To confirm that the aforementioned packages are installed, use the **rpm** utility:

```
~]$ rpm -qa | grep selinux
selinux-policy-3.12.1-136.el7.noarch
libselinux-2.2.2-4.el7.x86_64
selinux-policy-targeted-3.12.1-136.el7.noarch
libselinux-utils-2.2.2-4.el7.x86_64
libselinux-python-2.2.2-4.el7.x86_64
```

```
~]$ rpm -qa | grep policycoreutils
policycoreutils-2.2.5-6.el7.x86_64
policycoreutils-python-2.2.5-6.el7.x86_64
```

```
~]$ rpm -qa | grep setroubleshoot
setroubleshoot-server-3.2.17-2.el7.x86_64
setroubleshoot-3.2.17-2.el7.x86_64
setroubleshoot-plugins-3.0.58-2.el7.noarch
```

If they are not installed, use the **yum** utility as root to install them:

```
~]# yum install package_name
```

The following packages are optional:

- *policycoreutils-gui*
- *setroubleshoot*
- *mcstrans*

2. Before SELinux is enabled, each file on the file system must be labeled with an SELinux context. Before this happens, confined domains may be denied access, preventing your system from booting correctly. To prevent this, configure **SELINUX=permissive** 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=permissive
# SELINUXTYPE= can take one of these two values:
#       targeted - Targeted processes are protected,
#       mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

3. As root, restart the system. During the next boot, file systems are labeled. The label process labels all files with an SELinux context:

```
~]# reboot
```

```
*** Warning -- SELinux targeted policy relabel is required.
*** Relabeling could take a very long time, depending on file
*** system size and speed of hard drives.
****
```

Each `*` (asterisk) character on the bottom line represents 1000 files that have been labeled. In the above example, four `*` characters represent 4000 files have been labeled. The time it takes to label all files depends upon the number of files on the system, and the speed of the hard disk drives. On modern systems, this process can take as little as 10 minutes.

4. In permissive mode, SELinux policy is not enforced, but denials are still logged for actions that would have been denied if running in enforcing mode. Before changing to enforcing mode, as root, run the following command to confirm that SELinux did not deny actions during the last boot. If SELinux did not deny actions during the last boot, this command does not return any output. Refer to Chapter 10, *Troubleshooting* for troubleshooting information if SELinux denied access during boot.

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

5. If there were no denial messages in the **`/var/log/messages`** file, configure **`SELINUX=enforcing`** in **`/etc/selinux/config`**:

```
# 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
```

6. Reboot your system. After reboot, confirm that **`getenforce`** returns **`Enforcing`**:

```
~]$ getenforce
Enforcing
```

7. As root, run the following command to view the mapping between SELinux and Linux users. The output should be as follows:

```
~]# 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      *
system_u            system_u            s0-s0:c0.c1023      *
```

If this is not the case, run the following commands as root to fix the user mappings. It is safe to ignore the **`SELinux-user `*`username`*` is already defined`** warnings if they occur, where *username* can be **`unconfined_u`**, **`guest_u`**, or **`xguest_u`**:

Procedure 4.3. Fixing User Mappings

1.
   ```
   ~]# semanage user -a -S targeted -P user -R "unconfined_r system_r" -r s0-s0:c0.c1023 unconfined_u
   ```

2.
   ```
   ~]# semanage login -m -S targeted -s "unconfined_u" -r s0-s0:c0.c1023 __default__
   ```

3.
   ```
   ~]# semanage login -m -S targeted -s "unconfined_u" -r s0-s0:c0.c1023 root
   ```

4.
   ```
   ~]# semanage user -a -S targeted -P user -R guest_r guest_u
   ```

5.
   ```
   ~]# semanage user -a -S targeted -P user -R xguest_r xguest_u
   ```

Important

When systems run with SELinux in permissive or disabled mode, users have permission to label files incorrectly. Also, files created while SELinux is disabled are not labeled. This causes problems when changing to enforcing mode. To prevent incorrectly labeled and unlabeled files from causing problems, file systems are automatically relabeled when changing from disabled mode to permissive or enforcing mode.

4.4.2. Disabling SELinux

To disable SELinux, 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
```

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

```
~]$ getenforce
Disabled
```

4.5. Booleans

Booleans allow parts of SELinux policy to be changed at runtime, without any knowledge of SELinux policy writing. This allows changes, such as allowing services access to NFS volumes, without reloading or recompiling SELinux policy.

4.5.1. Listing Booleans

For a list of Booleans, an explanation of what each one is, and whether they are on or off, run the **semanage boolean -l** command as the Linux root user. The following example does not list all Booleans and the output is shortened for brevity:

```
~]# semanage boolean -l
SELinux boolean                State  Default Description

ftp_home_dir                   (off  ,  off)  Determine whether ftpd can read...
smartmon_3ware                 (off  ,  off)  Determine whether smartmon can...
mpd_enable_homedirs            (off  ,  off)  Determine whether mpd can traverse...
```

The **SELinux boolean** column lists Boolean names. The **Description** column lists whether the Booleans are on or off, and what they do.

In the following example, the **ftp_home_dir** Boolean is off, preventing the FTP daemon (**vsftpd**) from reading and writing to files in user home directories:

```
ftp_home_dir                   (off  ,  off)  Determine whether ftpd can read...
```

The **getsebool -a** command lists Booleans, whether they are on or off, but does not give a description of each one. The following example does not list all Booleans:

```
~]$ getsebool -a
cvs_read_shadow --> off
daemons_dump_core --> on
ftp_home_dir --> off
```

Run the **getsebool *boolean-name*** command to only list the status of the *boolean-name* Boolean:

```
~]$ getsebool cvs_read_shadow
cvs_read_shadow --> off
```

Use a space-separated list to list multiple Booleans:

```
~]$ getsebool cvs_read_shadow daemons_dump_core ftp_home_dir
cvs_read_shadow --> off
daemons_dump_core --> on
ftp_home_dir --> off
```

4.5.2. Configuring Booleans

Run the **setsebool** utility in the **setsebool *boolean_name* on/off** form to enable or disable Booleans.

The following example demonstrates configuring the **httpd_can_network_connect_db** Boolean:

Procedure 4.4. Configuring Booleans

1. By default, the **httpd_can_network_connect_db** Boolean is off, preventing Apache HTTP Server scripts and modules from connecting to database servers:

   ```
   ~]$ getsebool httpd_can_network_connect_db
   httpd_can_network_connect_db --> off
   ```

2. To temporarily enable Apache HTTP Server scripts and modules to connect to database servers, run the following command as root:

```
~]# setsebool httpd_can_network_connect_db on
```

3. Use the **getsebool** utility to verify the Boolean has been enabled:

   ```
   ~]$ getsebool httpd_can_network_connect_db
   httpd_can_network_connect_db --> on
   ```

   This allows Apache HTTP Server scripts and modules to connect to database servers.

4. This change is not persistent across reboots. To make changes persistent across reboots, run the **setsebool -P *boolean-name* on** command as root: [6]

   ```
   ~]# setsebool -P httpd_can_network_connect_db on
   ```

4.6. SELinux Contexts – Labeling Files

On systems running SELinux, all processes and files are labeled in a way that represents security-relevant information. This information is called the SELinux context. For files, this is viewed using the **ls -Z** command:

```
~]$ ls -Z file1
-rw-rw-r--  user1 group1 unconfined_u:object_r:user_home_t:s0 file1
```

In this example, SELinux provides a user (**unconfined_u**), a role (**object_r**), a type (**user_home_t**), and a level (**s0**). This information is used to make access control decisions. On DAC systems, access is controlled based on Linux user and group IDs. SELinux policy rules are checked after DAC rules. SELinux policy rules are not used if DAC rules deny access first.

There are multiple commands for managing the SELinux context for files, such as **chcon**, **semanage fcontext**, and **restorecon**.

4.6.1. Temporary Changes: chcon

The **chcon** command changes the SELinux context for files. However, changes made with the **chcon** command do not survive a file system relabel, or the execution of the **restorecon** command. SELinux policy controls whether users are able to modify the SELinux context for any given file. When using **chcon**, users provide all or part of the SELinux context to change. An incorrect file type is a common cause of SELinux denying access.

Quick Reference

- Run the **chcon -t *type* *file-name*** command to change the file type, where *type* is an SELinux type, such as **httpd_sys_content_t**, and *file-name* is a file or directory name:

  ```
  ~]$ chcon -t httpd_sys_content_t file-name
  ```

- Run the **chcon -R -t *type* *directory-name*** command to change the type of the directory and its contents, where *type* is an SELinux type, such as **httpd_sys_content_t**, and *directory-name* is a directory name:

  ```
  ~]$ chcon -R -t httpd_sys_content_t directory-name
  ```

Procedure 4.5. Changing a File's or Directory's Type

The following procedure demonstrates changing the type, and no other attributes of the SELinux context. The example in this section works the same for directories, for example, if **`file1`** was a directory.

1. Change into your home directory.

2. Create a new file and view its SELinux context:

   ```
   ~]$ touch file1
   ```

   ```
   ~]$ ls -Z file1
   -rw-rw-r--  user1 group1 unconfined_u:object_r:user_home_t:s0 file1
   ```

   In this example, the SELinux context for **`file1`** includes the SELinux **`unconfined_u`** user, **`object_r`** role, **`user_home_t`** type, and the **`s0`** level. For a description of each part of the SELinux context, see Chapter 2, *SELinux Contexts*.

3. Run the following command to change the type to **`samba_share_t`**. The **`-t`** option only changes the type. Then view the change:

   ```
   ~]$ chcon -t samba_share_t file1
   ```

   ```
   ~]$ ls -Z file1
   -rw-rw-r--  user1 group1 unconfined_u:object_r:samba_share_t:s0 file1
   ```

4. Use the following command to restore the SELinux context for the **`file1`** file. Use the **`-v`** option to view what changes:

   ```
   ~]$ restorecon -v file1
   restorecon reset file1 context unconfined_u:object_r:samba_share_t:s0-
   >system_u:object_r:user_home_t:s0
   ```

   In this example, the previous type, **`samba_share_t`**, is restored to the correct, **`user_home_t`** type. When using targeted policy (the default SELinux policy in Red Hat Enterprise Linux), the **`restorecon`** command reads the files in the **`/etc/selinux/targeted/contexts/files/`** directory, to see which SELinux context files should have.

Procedure 4.6. Changing a Directory and its Contents Types

The following example demonstrates creating a new directory, and changing the directory's file type (along with its contents) to a type used by the Apache HTTP Server. The configuration in this example is used if you want Apache HTTP Server to use a different document root (instead of **`/var/www/html/`**):

1. As the root user, create a new **`/mkdir/`**directory and then 3 empty files (**`file1`**, **`file2`**, and **`file3`**) within this directory. The **`/web/`** directory and files in it are labeled with the **`default_t`** type:

   ```
   ~]# mkdir /web
   ```

   ```
   ~]# touch /web/file{1,2,3}
   ```

   ```
   ~]# ls -dZ /web
   drwxr-xr-x  root root unconfined_u:object_r:default_t:s0 /web
   ```

```
~]# ls -lZ /web
-rw-r--r--  root root unconfined_u:object_r:default_t:s0 file1
-rw-r--r--  root root unconfined_u:object_r:default_t:s0 file2
-rw-r--r--  root root unconfined_u:object_r:default_t:s0 file3
```

2. As root, run the following command to change the type of the **/web/** directory (and its contents) to **httpd_sys_content_t**:

```
~]# chcon -R -t httpd_sys_content_t /web/
```

```
~]# ls -dZ /web/
drwxr-xr-x  root root unconfined_u:object_r:httpd_sys_content_t:s0 /web/
```

```
~]# ls -lZ /web/
-rw-r--r--  root root unconfined_u:object_r:httpd_sys_content_t:s0 file1
-rw-r--r--  root root unconfined_u:object_r:httpd_sys_content_t:s0 file2
-rw-r--r--  root root unconfined_u:object_r:httpd_sys_content_t:s0 file3
```

3. To restore the default SELinux contexts, use the **restorecon** utility as root:

```
~]# restorecon -R -v /web/
restorecon reset /web context unconfined_u:object_r:httpd_sys_content_t:s0-
>system_u:object_r:default_t:s0
restorecon reset /web/file2 context
unconfined_u:object_r:httpd_sys_content_t:s0->system_u:object_r:default_t:s0
restorecon reset /web/file3 context
unconfined_u:object_r:httpd_sys_content_t:s0->system_u:object_r:default_t:s0
restorecon reset /web/file1 context
unconfined_u:object_r:httpd_sys_content_t:s0->system_u:object_r:default_t:s0
```

Refer to the chcon(1) manual page for further information about **chcon**.

Note

Type Enforcement is the main permission control used in SELinux targeted policy. For the most part, SELinux users and roles can be ignored.

4.6.2. Persistent Changes: semanage fcontext

The **semanage fcontext** command is used to change the SELinux context of files. When using targeted policy, changes are written to files located in the **/etc/selinux/targeted/contexts/files/** directory:

- The **file_contexts** file specifies default contexts for many files, as well as contexts updated via **semanage fcontext**.
- The **file_contexts.local** file stores contexts to newly created files and directories not found in **file_contexts**.

Two utilities read these files. The **setfiles** utility is used when a file system is relabeled and the **restorecon** utility restores the default SELinux contexts. This means that changes made by **semanage fcontext** are persistent, even if the file system is relabeled. SELinux policy controls whether users are able to modify the SELinux context for any given file.

Quick Reference

To make SELinux context changes that survive a file system relabel:

1. Run the following command, remembering to use the full path to the file or directory:

   ```
   ~]# semanage fcontext -a options file-name|directory-name
   ```

2. Use the **restorecon** utility to apply the context changes:

   ```
   ~]# restorecon -v file-name|directory-name
   ```

Procedure 4.7. Changing a File's or Directory's Type

The following example demonstrates changing a file's type, and no other attributes of the SELinux context. This example works the same for directories, for instance if **file1** was a directory.

1. As the root user, create a new file in the **/etc/** directory. By default, newly-created files in **/etc/** are labeled with the **etc_t** type:

   ```
   ~]# touch /etc/file1
   ```

   ```
   ~]$ ls -Z /etc/file1
   -rw-r--r--  root root unconfined_u:object_r:etc_t:s0       /etc/file1
   ```

   To list information about a directory, use the following command:

   ```
   ~]$ ls -dZ directory_name
   ```

2. As root, run the following command to change the **file1** type to **samba_share_t**. The **-a** option adds a new record, and the **-t** option defines a type (**samba_share_t**). Note that running this command does not directly change the type; **file1** is still labeled with the **etc_t** type:

   ```
   ~]# semanage fcontext -a -t samba_share_t /etc/file1
   ```

   ```
   ~]# ls -Z /etc/file1
   -rw-r--r--  root root unconfined_u:object_r:etc_t:s0       /etc/file1
   ```

   The **semanage fcontext -a -t samba_share_t /etc/file1** command adds the following entry to **/etc/selinux/targeted/contexts/files/file_contexts.local**:

   ```
   /etc/file1    unconfined_u:object_r:samba_share_t:s0
   ```

3. As root, use the **restorecon** utility to change the type. Because **semanage** added an entry to **file.contexts.local** for **/etc/file1**, **restorecon** changes the type to **samba_share_t**:

   ```
   ~]# restorecon -v /etc/file1
   restorecon reset /etc/file1 context unconfined_u:object_r:etc_t:s0-
   >system_u:object_r:samba_share_t:s0
   ```

Procedure 4.8. Changing a Directory and its Contents Types

The following example demonstrates creating a new directory, and changing the directory's file type (along with its contents) to a type used by Apache HTTP Server. The configuration in this example is used if you want Apache HTTP Server to use a different document root (instead of **/var/www/html/**):

1. As the root user, create a new **/mkdir/**directory and then 3 empty files (**file1**, **file2**, and **file3**) within this directory. The **/web/** directory and files in it are labeled with the **default_t** type:

   ```
   ~]# mkdir /web
   ```

   ```
   ~]# touch /web/file{1,2,3}
   ```

   ```
   ~]# ls -dZ /web
   drwxr-xr-x  root root unconfined_u:object_r:default_t:s0 /web
   ```

   ```
   ~]# ls -lZ /web
   -rw-r--r--  root root unconfined_u:object_r:default_t:s0 file1
   -rw-r--r--  root root unconfined_u:object_r:default_t:s0 file2
   -rw-r--r--  root root unconfined_u:object_r:default_t:s0 file3
   ```

2. As root, run the following command to change the type of the **/web/** directory and the files in it, to **httpd_sys_content_t**. The **-a** option adds a new record, and the **-t** option defines a type (httpd_sys_content_t). The **"/web(/.*)?"** regular expression causes **semanage** to apply changes to **/web/**, as well as the files in it. Note that running this command does not directly change the type; **/web/** and files in it are still labeled with the **default_t** type:

   ```
   ~]# semanage fcontext -a -t httpd_sys_content_t "/web(/.*)?"
   ```

   ```
   ~]$ ls -dZ /web
   drwxr-xr-x  root root unconfined_u:object_r:default_t:s0 /web
   ```

   ```
   ~]$ ls -lZ /web
   -rw-r--r--  root root unconfined_u:object_r:default_t:s0 file1
   -rw-r--r--  root root unconfined_u:object_r:default_t:s0 file2
   -rw-r--r--  root root unconfined_u:object_r:default_t:s0 file3
   ```

   The **semanage fcontext -a -t httpd_sys_content_t "/web(/.*)?"** command adds the following entry to **/etc/selinux/targeted/contexts/files/file_contexts.local**:

   ```
   /web(/.*)?    system_u:object_r:httpd_sys_content_t:s0
   ```

3. As root, use the **restorecon** utility to change the type of **/web/**, as well as all files in it. The **-R** is for recursive, which means all files and directories under **/web/** are labeled with the **httpd_sys_content_t** type. Since **semanage** added an entry to **file.contexts.local** for **/web(/.*)?**, **restorecon** changes the types to **httpd_sys_content_t**:

   ```
   ~]# restorecon -R -v /web
   restorecon reset /web context unconfined_u:object_r:default_t:s0-
   >system_u:object_r:httpd_sys_content_t:s0
   restorecon reset /web/file2 context unconfined_u:object_r:default_t:s0-
   >system_u:object_r:httpd_sys_content_t:s0
   restorecon reset /web/file3 context unconfined_u:object_r:default_t:s0-
   >system_u:object_r:httpd_sys_content_t:s0
   restorecon reset /web/file1 context unconfined_u:object_r:default_t:s0-
   >system_u:object_r:httpd_sys_content_t:s0
   ```

By default, newly-created files and directories inherit the SELinux type of their parents. In this example, files and directories created in the **/web/** directory will be labeled with the **httpd_sys_content_t** type.

Procedure 4.9. Deleting an added Context

The following example demonstrates adding and removing an SELinux context. If the context is part of a regular expression, for example, **/web(/.*)?**, use quotation marks around the regular expression:

```
~]# semanage fcontext -d "/web(/.*)?"
```

1. To remove the context, as root, run the following command, where *file-name*|*directory-name* is the first part in **file_contexts.local**:

   ```
   ~]# semanage fcontext -d file-name|directory-name
   ```

   The following is an example of a context in **file_contexts.local**:

   ```
   /test    system_u:object_r:httpd_sys_content_t:s0
   ```

   With the first part being **/test**. To prevent the **/test/** directory from being labeled with the **httpd_sys_content_t** after running **restorecon**, or after a file system relabel, run the following command as root to delete the context from **file_contexts.local**:

   ```
   ~]# semanage fcontext -d /test
   ```

2. As root, use the **restorecon** utility to restore the default SELinux context.

See the semanage(8) manual page for further information about **semanage**.

Important

When changing the SELinux context with **semanage fcontext -a**, use the full path to the file or directory to avoid files being mislabeled after a file system relabel, or after the **restorecon** command is run.

4.7. The file_t and default_t Types

When using a file system that supports extended attributes (EA), the **file_t** type is the default type of a file that has not yet been assigned EA value. This type is only used for this purpose and does not exist on correctly-labeled file systems, because all files on a system running SELinux should have a proper SELinux context, and the **file_t** type is never used in file-context configuration [7].

The **default_t** type is used on files that do not match any pattern in file-context configuration, so that such files can be distinguished from files that do not have a context on disk, and generally are kept inaccessible to confined domains. For example, if you create a new top-level directory, such as **/mydirectory/**, this directory may be labeled with the default_t type. If services need access to this directory, you need to update the file-contexts configuration for this location. See Section 4.6.2, "Persistent Changes: semanage fcontext" for details on adding a context to the file-context configuration.

4.8. Mounting File Systems

By default, when a file system that supports extended attributes is mounted, the security context for each file is obtained from the *security.selinux* extended attribute of the file. Files in file systems that do not support extended attributes are assigned a single, default security context from the policy configuration, based on file system type.

Use the **`mount -o context`** command to override existing extended attributes, or to specify a different, default context for file systems that do not support extended attributes. This is useful if you do not trust a file system to supply the correct attributes, for example, removable media used in multiple systems. The **`mount -o context`** command can also be used to support labeling for file systems that do not support extended attributes, such as File Allocation Table (FAT) or NFS volumes. The context specified with the **`context`** option is not written to disk: the original contexts are preserved, and are seen when mounting without **`context`** (if the file system had extended attributes in the first place).

For further information about file system labeling, refer to James Morris's "Filesystem Labeling in SELinux" article: http://www.linuxjournal.com/article/7426.

4.8.1. Context Mounts

To mount a file system with the specified context, overriding existing contexts if they exist, or to specify a different, default context for a file system that does not support extended attributes, as the root user, use the **`mount -o context=`*`SELinux_user:role:type:level`*** command when mounting the desired file system. Context changes are not written to disk. By default, NFS mounts on the client side are labeled with a default context defined by policy for NFS volumes. In common policies, this default context uses the **`nfs_t`** type. Without additional mount options, this may prevent sharing NFS volumes using other services, such as the Apache HTTP Server. The following example mounts an NFS volume so that it can be shared via the Apache HTTP Server:

```
~]# mount server:/export /local/mount/point -o \
context="system_u:object_r:httpd_sys_content_t:s0"
```

Newly-created files and directories on this file system appear to have the SELinux context specified with **`-o context`**. However, since these changes are not written to disk, the context specified with this option does not persist between mounts. Therefore, this option must be used with the same context specified during every mount to retain the desired context. For information about making context mount persistent, refer to Section 4.8.5, “Making Context Mounts Persistent”.

Type Enforcement is the main permission control used in SELinux targeted policy. For the most part, SELinux users and roles can be ignored, so, when overriding the SELinux context with **`-o context`**, use the SELinux **`system_u`** user and **`object_r`** role, and concentrate on the type. If you are not using the MLS policy or multi-category security, use the **`s0`** level.

Note

When a file system is mounted with a **`context`** option, context changes (by users and processes) are prohibited. For example, running the **`chcon`** command on a file system mounted with a **`context`** option results in a **`Operation not supported`** error.

4.8.2. Changing the Default Context

As mentioned in Section 4.7, “The file_t and default_t Types”, on file systems that support extended attributes, when a file that lacks an SELinux context on disk is accessed, it is treated as if it had a default context as defined by SELinux policy. In common policies, this default context uses the **file_t** type. If it is desirable to use a different default context, mount the file system with the **defcontext** option.

The following example mounts a newly-created file system (on **/dev/sda2**) to the newly-created **/test/** directory. It assumes that there are no rules in **/etc/selinux/targeted/contexts/files/** that define a context for the **/test/** directory:

```
~]# mount /dev/sda2 /test/ -o defcontext="system_u:object_r:samba_share_t:s0"
```

In this example:

- the **defcontext** option defines that **system_u:object_r:samba_share_t:s0** is "the default security context for unlabeled files" [8].
- when mounted, the root directory (**/test/**) of the file system is treated as if it is labeled with the context specified by **defcontext** (this label is not stored on disk). This affects the labeling for files created under **/test/**: new files inherit the **samba_share_t** type, and these labels are stored on disk.
- files created under **/test/** while the file system was mounted with a **defcontext** option retain their labels.

4.8.3. Mounting an NFS Volume

By default, NFS mounts on the client side are labeled with a default context defined by policy for NFS volumes. In common policies, this default context uses the **nfs_t** type. Depending on policy configuration, services, such as Apache HTTP Server and MariaDB, may not be able to read files labeled with the **nfs_t** type. This may prevent file systems labeled with this type from being mounted and then read or exported by other services.

If you would like to mount an NFS volume and read or export that file system with another service, use the **context** option when mounting to override the **nfs_t** type. Use the following context option to mount NFS volumes so that they can be shared via the Apache HTTP Server:

```
~]# mount server:/export /local/mount/point -o
context="system_u:object_r:httpd_sys_content_t:s0"
```

Since these changes are not written to disk, the context specified with this option does not persist between mounts. Therefore, this option must be used with the same context specified during every mount to retain the desired context. For information about making context mount persistent, refer to Section 4.8.5, “Making Context Mounts Persistent”.

As an alternative to mounting file systems with **context** options, Booleans can be enabled to allow services access to file systems labeled with the **nfs_t** type. Refer to Part II, “Managing Confined Services” for instructions on configuring Booleans to allow services access to the **nfs_t** type.

4.8.4. Multiple NFS Mounts

When mounting multiple mounts from the same NFS export, attempting to override the SELinux context of each mount with a different context, results in subsequent mount commands failing. In the following example, the NFS server has a single export, **/export/**, which has two subdirectories, **/web/** and **/database/**. The following commands attempt two mounts from a single NFS export, and try to override the context for each one:

```
~]# mount server:/export/web /local/web -o
context="system_u:object_r:httpd_sys_content_t:s0"
```

```
~]# mount server:/export/database /local/database -o
context="system_u:object_r:mysqld_db_t:s0"
```

The second mount command fails, and the following is logged to **/var/log/messages**:

```
kernel: SELinux: mount invalid.  Same superblock, different security settings for
(dev 0:15, type nfs)
```

To mount multiple mounts from a single NFS export, with each mount having a different context, use the **-o nosharecache,context** options. The following example mounts multiple mounts from a single NFS export, with a different context for each mount (allowing a single service access to each one):

```
~]# mount server:/export/web /local/web -o
nosharecache,context="system_u:object_r:httpd_sys_content_t:s0"
```

```
~]# mount server:/export/database /local/database -o \
nosharecache,context="system_u:object_r:mysqld_db_t:s0"
```

In this example, **server:/export/web** is mounted locally to the **/local/web/** directory, with all files being labeled with the **httpd_sys_content_t** type, allowing Apache HTTP Server access. **server:/export/database** is mounted locally to **/local/database/**, with all files being labeled with the **mysqld_db_t** type, allowing MariaDB access. These type changes are not written to disk.

Important

The **nosharecache** options allows you to mount the same subdirectory of an export multiple times with different contexts (for example, mounting **/export/web/** multiple times). Do not mount the same subdirectory from an export multiple times with different contexts, as this creates an overlapping mount, where files are accessible under two different contexts.

4.8.5. Making Context Mounts Persistent

To make context mounts persistent across remounting and reboots, add entries for the file systems in the **/etc/fstab** file or an automounter map, and use the desired context as a mount option. The following example adds an entry to **/etc/fstab** for an NFS context mount:

```
server:/export /local/mount/ nfs context="system_u:object_r:httpd_sys_content_t:s0" 0
0
```

4.9. Maintaining SELinux Labels

These sections describe what happens to SELinux contexts when copying, moving, and archiving files and directories. Also, it explains how to preserve contexts when copying and archiving.

4.9.1. Copying Files and Directories

When a file or directory is copied, a new file or directory is created if it does not exist. That new file or directory's context is based on default-labeling rules, not the original file or directory's context (unless options were used to preserve the original context). For example, files created in user home directories are labeled with the **user_home_t** type:

```
~]$ touch file1
```

```
~]$ ls -Z file1
-rw-rw-r--  user1 group1 unconfined_u:object_r:user_home_t:s0 file1
```

If such a file is copied to another directory, such as **/etc/**, the new file is created in accordance to default-labeling rules for **/etc/**. Copying a file (without additional options) may not preserve the original context:

```
~]$ ls -Z file1
-rw-rw-r--  user1 group1 unconfined_u:object_r:user_home_t:s0 file1
```

```
~]# cp file1 /etc/
```

```
~]$ ls -Z /etc/file1
-rw-r--r--  root root unconfined_u:object_r:etc_t:s0   /etc/file1
```

When **file1** is copied to **/etc/**, if **/etc/file1** does not exist, **/etc/file1** is created as a new file. As shown in the example above, **/etc/file1** is labeled with the **etc_t** type, in accordance to default-labeling rules.

When a file is copied over an existing file, the existing file's context is preserved, unless the user specified **cp** options to preserve the context of the original file, such as **--preserve=context**. SELinux policy may prevent contexts from being preserved during copies.

Procedure 4.10. Copying Without Preserving SELinux Contexts

This procedure shows that when copying a file with the **cp** command, if no options are given, the type is inherited from the targeted, parent directory.

1. Create a file in a user's home directory. The file is labeled with the **user_home_t** type:

   ```
   ~]$ touch file1
   ```

   ```
   ~]$ ls -Z file1
   -rw-rw-r--  user1 group1 unconfined_u:object_r:user_home_t:s0 file1
   ```

2. The **/var/www/html/** directory is labeled with the **httpd_sys_content_t** type, as shown with the following command:

   ```
   ~]$ ls -dZ /var/www/html/
   drwxr-xr-x  root root system_u:object_r:httpd_sys_content_t:s0 /var/www/html/
   ```

3. When **file1** is copied to **/var/www/html/**, it inherits the **httpd_sys_content_t** type:

   ```
   ~]# cp file1 /var/www/html/
   ```

   ```
   ~]$ ls -Z /var/www/html/file1
   -rw-r--r--  root root unconfined_u:object_r:httpd_sys_content_t:s0
   /var/www/html/file1
   ```

Procedure 4.11. Preserving SELinux Contexts When Copying

This procedure shows how to use the **--preserve=context** option to preserve contexts when copying.

1. Create a file in a user's home directory. The file is labeled with the **user_home_t** type:

   ```
   ~]$ touch file1
   ```

   ```
   ~]$ ls -Z file1
   -rw-rw-r--  user1 group1 unconfined_u:object_r:user_home_t:s0 file1
   ```

2. The **/var/www/html/** directory is labeled with the **httpd_sys_content_t** type, as shown with the following command:

   ```
   ~]$ ls -dZ /var/www/html/
   drwxr-xr-x  root root system_u:object_r:httpd_sys_content_t:s0 /var/www/html/
   ```

3. Using the **--preserve=context** option preserves SELinux contexts during copy operations. As shown below, the **user_home_t** type of **file1** was preserved when the file was copied to **/var/www/html/**:

   ```
   ~]# cp --preserve=context file1 /var/www/html/
   ```

   ```
   ~]$ ls -Z /var/www/html/file1
   -rw-r--r--  root root unconfined_u:object_r:user_home_t:s0 /var/www/html/file1
   ```

Procedure 4.12. Copying and Changing the Context

This procedure show how to use the **--context** option to change the destination copy's context. The following example is performed in the user's home directory:

1. Create a file in a user's home directory. The file is labeled with the **user_home_t** type:

   ```
   ~]$ touch file1
   ```

   ```
   ~]$ ls -Z file1
   -rw-rw-r--  user1 group1 unconfined_u:object_r:user_home_t:s0 file1
   ```

2. Use the **--context** option to define the SELinux context:

   ```
   ~]$ cp --context=system_u:object_r:samba_share_t:s0 file1 file2
   ```

3. Without **--context**, **file2** would be labeled with the **unconfined_u:object_r:user_home_t** context:

   ```
   ~]$ ls -Z file1 file2
   -rw-rw-r--  user1 group1 unconfined_u:object_r:user_home_t:s0 file1
   -rw-rw-r--  user1 group1 system_u:object_r:samba_share_t:s0 file2
   ```

Procedure 4.13. Copying a File Over an Existing File

This procedure shows that when a file is copied over an existing file, the existing file's context is preserved (unless an option is used to preserve contexts).

1. As root, create a new file, **file1** in the **/etc/** directory. As shown below, the file is labeled with the **etc_t** type:

   ```
   ~]# touch /etc/file1
   ```

   ```
   ~]$ ls -Z /etc/file1
   -rw-r--r--  root root unconfined_u:object_r:etc_t:s0   /etc/file1
   ```

2. Create another file, **file2**, in the **/tmp/** directory. As shown below, the file is labeled with the **user_tmp_t** type:

   ```
   ~]$ touch /tmp/file2
   ```

   ```
   ~$ ls -Z /tmp/file2
   -rw-r--r--  root root unconfined_u:object_r:user_tmp_t:s0 /tmp/file2
   ```

3. Overwrite **file1** with **file2**:

   ```
   ~]# cp /tmp/file2 /etc/file1
   ```

4. After copying, the following command shows **file1** labeled with the **etc_t** type, not the **user_tmp_t** type from **/tmp/file2** that replaced **/etc/file1**:

   ```
   ~]$ ls -Z /etc/file1
   -rw-r--r--  root root unconfined_u:object_r:etc_t:s0   /etc/file1
   ```

Important

Copy files and directories, rather than moving them. This helps ensure they are labeled with the correct SELinux contexts. Incorrect SELinux contexts can prevent processes from accessing such files and directories.

4.9.2. Moving Files and Directories

Files and directories keep their current SELinux context when they are moved. In many cases, this is incorrect for the location they are being moved to. The following example demonstrates moving a file from a user's home directory to the **/var/www/html/** directory, which is used by the Apache HTTP Server. Since the file is moved, it does not inherit the correct SELinux context:

Procedure 4.14. Moving Files and Directories

1. Change into your home directory and create file in it. The file is labeled with the **user_home_t** type:

   ```
   ~]$ touch file1
   ```

   ```
   ~]$ ls -Z file1
   -rw-rw-r--  user1 group1 unconfined_u:object_r:user_home_t:s0 file1
   ```

2. Run the following command to view the SELinux context of the **/var/www/html/** directory:

```
~]$ ls -dZ /var/www/html/
drwxr-xr-x  root root system_u:object_r:httpd_sys_content_t:s0 /var/www/html/
```

By default, **/var/www/html/** is labeled with the **httpd_sys_content_t** type. Files and directories created under **/var/www/html/** inherit this type, and as such, they are labeled with this type.

3. As root, move **file1** to **/var/www/html/**. Since this file is moved, it keeps its current **user_home_t** type:

```
~]# mv file1 /var/www/html/
```

```
~]# ls -Z /var/www/html/file1
-rw-rw-r--  user1 group1 unconfined_u:object_r:user_home_t:s0
/var/www/html/file1
```

By default, the Apache HTTP Server cannot read files that are labeled with the **user_home_t** type. If all files comprising a web page are labeled with the **user_home_t** type, or another type that the Apache HTTP Server cannot read, permission is denied when attempting to access them via web browsers, such as **Mozilla Firefox**.

Important

Moving files and directories with the **mv** command may result in the incorrect SELinux context, preventing processes, such as the Apache HTTP Server and Samba, from accessing such files and directories.

4.9.3. Checking the Default SELinux Context

Use the **matchpathcon** utility to check if files and directories have the correct SELinux context. This utility queries the system policy and then provides the default security context associated with the file path. [9] The following example demonstrates using **matchpathcon** to verify that files in **/var/www/html/** directory are labeled correctly:

Procedure 4.15. Checking the Default SELinux Conxtext with matchpathcon

1. As the root user, create three files (**file1**, **file2**, and **file3**) in the **/var/www/html/** directory. These files inherit the **httpd_sys_content_t** type from **/var/www/html/**:

```
~]# touch /var/www/html/file{1,2,3}
```

```
~]# ls -Z /var/www/html/
-rw-r--r--  root root unconfined_u:object_r:httpd_sys_content_t:s0 file1
-rw-r--r--  root root unconfined_u:object_r:httpd_sys_content_t:s0 file2
-rw-r--r--  root root unconfined_u:object_r:httpd_sys_content_t:s0 file3
```

2. As root, change the **file1** type to **samba_share_t**. Note that the Apache HTTP Server cannot read files or directories labeled with the **samba_share_t** type.

```
~]# chcon -t samba_share_t /var/www/html/file1
```

3. The **matchpathcon -V** option compares the current SELinux context to the correct, default context in SELinux policy. Run the following command to check all files in the **/var/www/html/** directory:

   ```
   ~]$ matchpathcon -V /var/www/html/*
   /var/www/html/file1 has context unconfined_u:object_r:samba_share_t:s0, should
   be system_u:object_r:httpd_sys_content_t:s0
   /var/www/html/file2 verified.
   /var/www/html/file3 verified.
   ```

The following output from the **matchpathcon** command explains that **file1** is labeled with the **samba_share_t** type, but should be labeled with the **httpd_sys_content_t** type:

```
/var/www/html/file1 has context unconfined_u:object_r:samba_share_t:s0, should be
system_u:object_r:httpd_sys_content_t:s0
```

To resolve the label problem and allow the Apache HTTP Server access to **file1**, as root, use the **restorecon** utility:

```
~]# restorecon -v /var/www/html/file1
restorecon reset /var/www/html/file1 context unconfined_u:object_r:samba_share_t:s0-
>system_u:object_r:httpd_sys_content_t:s0
```

4.9.4. Archiving Files with tar

The **tar** utility does not retain extended attributes by default. Since SELinux contexts are stored in extended attributes, contexts can be lost when archiving files. Use the **tar --selinux** command to create archives that retain contexts. If a **tar** archive contains files without extended attributes, or if you want the extended attributes to match the system defaults, use the **restorecon** utility:

```
~]$ tar -xvf archive.tar | restorecon -f -
```

Note that depending on the directory, you may need to be the root user to run the **restorecon**.

The following example demonstrates creating a **tar** archive that retains SELinux contexts:

Procedure 4.16. Creating a tar Archive

1. As root, create three files (**file1**, **file2**, and **file3**) in the **/var/www/html/** directory. These files inherit the **httpd_sys_content_t** type from **/var/www/html/**:

   ```
   ~]# touch /var/www/html/file{1,2,3}
   ```

   ```
   ~]# ls -Z /var/www/html/
   -rw-r--r--  root root unconfined_u:object_r:httpd_sys_content_t:s0 file1
   -rw-r--r--  root root unconfined_u:object_r:httpd_sys_content_t:s0 file2
   -rw-r--r--  root root unconfined_u:object_r:httpd_sys_content_t:s0 file3
   ```

2. Change into **/var/www/html/**. Once in this directory, as root, run the following command to create a **tar** archive named **test.tar**:

   ```
   ~]$ cd /var/www/html/
   ```

   ```
   html]# tar --selinux -cf test.tar file{1,2,3}
   ```

3. As root, create a new directory named **/test/**, and then allow all users full access to it:

   ```
   ~]# mkdir /test
   ```

   ```
   ~]# chmod 777 /test/
   ```

4. Copy the **test.tar** file into **/test/**:

   ```
   ~]$ cp /var/www/html/test.tar /test/
   ```

5. Change into **/test/** directory. Once in this directory, run the following command to extract the **tar** archive:

   ```
   test]$ tar -xvf test.tar
   ```

6. View the SELinux contexts. The **httpd_sys_content_t** type has been retained, rather than being changed to **default_t**, which would have happened had the **--selinux** not been used:

   ```
   ~]$ ls -lZ /test/
   -rw-r--r--  user1 group1 unconfined_u:object_r:httpd_sys_content_t:s0 file1
   -rw-r--r--  user1 group1 unconfined_u:object_r:httpd_sys_content_t:s0 file2
   -rw-r--r--  user1 group1 unconfined_u:object_r:httpd_sys_content_t:s0 file3
   -rw-r--r--  user1 group1 unconfined_u:object_r:default_t:s0 test.tar
   ```

7. If the **/test/** directory is no longer required, as root, run the following command to remove it, as well as all files in it:

   ```
   ~]# rm -ri /test/
   ```

See the tar(1) manual page for further information about **tar**, such as the **--xattrs** option that retains all extended attributes.

4.9.5. Archiving Files with star

The **star** utility does not retain extended attributes by default. Since SELinux contexts are stored in extended attributes, contexts can be lost when archiving files. Use the **star -xattr -H=exustar** command to create archives that retain contexts. The *star* package is not installed by default. To install **star**, run the **yum install star** command as the root user.

The following example demonstrates creating a **star** archive that retains SELinux contexts:

Procedure 4.17. Creating a star Archive

1. As root, create three files (**file1**, **file2**, and **file3**) in the **/var/www/html/**. These files inherit the **httpd_sys_content_t** type from **/var/www/html/**:

   ```
   ~]# touch /var/www/html/file{1,2,3}
   ```

   ```
   ~]# ls -Z /var/www/html/
   -rw-r--r--  root root unconfined_u:object_r:httpd_sys_content_t:s0 file1
   -rw-r--r--  root root unconfined_u:object_r:httpd_sys_content_t:s0 file2
   -rw-r--r--  root root unconfined_u:object_r:httpd_sys_content_t:s0 file3
   ```

2. Change into **/var/www/html/** directory. Once in this directory, as root, run the following command to create a **star** archive named **test.star**:

   ```
   ~]$ cd /var/www/html
   ```

   ```
   html]# star -xattr -H=exustar -c -f=test.star file{1,2,3}
   star: 1 blocks + 0 bytes (total of 10240 bytes = 10.00k).
   ```

3. As root, create a new directory named **/test/**, and then allow all users full access to it:

   ```
   ~]# mkdir /test
   ```

   ```
   ~]# chmod 777 /test/
   ```

4. Run the following command to copy the **test.star** file into **/test/**:

   ```
   ~]$ cp /var/www/html/test.star /test/
   ```

5. Change into **/test/**. Once in this directory, run the following command to extract the **star** archive:

   ```
   ~]$ cd /test/
   ```

   ```
   test]$ star -x -f=test.star
   star: 1 blocks + 0 bytes (total of 10240 bytes = 10.00k).
   ```

6. View the SELinux contexts. The **httpd_sys_content_t** type has been retained, rather than being changed to **default_t**, which would have happened had the **-xattr -H=exustar** option not been used:

   ```
   ~]$ ls -lZ /test/
   -rw-r--r--  user1 group1 unconfined_u:object_r:httpd_sys_content_t:s0 file1
   -rw-r--r--  user1 group1 unconfined_u:object_r:httpd_sys_content_t:s0 file2
   -rw-r--r--  user1 group1 unconfined_u:object_r:httpd_sys_content_t:s0 file3
   -rw-r--r--  user1 group1 unconfined_u:object_r:default_t:s0 test.star
   ```

7. If the **/test/** directory is no longer required, as root, run the following command to remove it, as well as all files in it:

   ```
   ~]#  rm -ri /test/
   ```

8. If **star** is no longer required, as root, remove the package:

   ```
   ~]# yum remove star
   ```

See the star(1) manual page for further information about **star**.

4.10. Information Gathering Tools

The utilities listed below are command-line tools that provide well-formatted information, such as access vector cache statistics or the number of classes, types, or Booleans.

avcstat

This command provides a short output of the access vector cache statistics since boot. You can watch the statistics in real time by specifying a time interval in seconds. This provides updated statistics since the initial output. The statistics file used is **/selinux/avc/cache_stats**, and you can specify a different cache file with the **-f /path/to/file** option.

```
~]# avcstat
   lookups       hits     misses     allocs   reclaims      frees
  47517410   47504630      12780      12780      12176      12275
```

seinfo

This utility is useful in describing the break-down of a policy, such as the number of classes, types, Booleans, allow rules, and others. **seinfo** is a command-line utility that uses a policy.conf file (a single text file containing policy source for versions 12 through 21), a binary policy file, a modular list of policy packages, or a policy list file as input. You must have the *setools-console* package installed to use the **seinfo** utility.

The output of **seinfo** will vary between binary and source files. For example, the policy source file uses the **{ }** brackets to group multiple rule elements onto a single line. A similar effect happens with attributes, where a single attribute expands into one or many types. Because these are expanded and no longer relevant in the binary policy file, they have a return value of zero in the search results. However, the number of rules greatly increases as each formerly one line rule using brackets is now a number of individual lines.

Some items are not present in the binary policy. For example, neverallow rules are only checked during policy compile, not during runtime, and initial Security Identifiers (SIDs) are not part of the binary policy since they are required prior to the policy being loaded by the kernel during boot.

```
~]# seinfo

Statistics for policy file: /etc/selinux/targeted/policy/policy.24
Policy Version  & Type: v.24 (binary, mls)

   Classes:            77    Permissions:       229
   Sensitivities:       1    Categories:       1024
   Types:            3001    Attributes:        244
   Users:               9    Roles:              13
   Booleans:          158    Cond. Expr.:       193
   Allow:          262796    Neverallow:          0
   Auditallow:         44    Dontaudit:      156710
   Type_trans:      10760    Type_change:        38
   Type_member:        44    Role allow:         20
   Role_trans:        237    Range_trans:      2546
   Constraints:        62    Validatetrans:       0
   Initial SIDs:       27    Fs_use:             22
   Genfscon:           82    Portcon:           373
   Netifcon:            0    Nodecon:             0
   Permissives:        22    Polcap:              2
```

The **seinfo** utility can also list the number of types with the domain attribute, giving an estimate of the number of different confined processes:

```
~]# seinfo -adomain -x | wc -l
550
```

Not all domain types are confined. To look at the number of unconfined domains, use the **unconfined_domain** attribute:

```
~]# seinfo -aunconfined_domain_type -x | wc -l
52
```

Permissive domains can be counted with the **--permissive** option:

```
~]# seinfo --permissive -x | wc -l
31
```

Remove the additional **| wc -l** command in the above commands to see the full lists.

sesearch

You can use the **sesearch** utility to search for a particular rule in the policy. It is possible to search either policy source files or the binary file. For example:

```
~]$ sesearch --role_allow -t httpd_sys_content_t
/etc/selinux/targeted/policy/policy.24
Found 20 role allow rules:
   allow system_r sysadm_r;
   allow sysadm_r system_r;
   allow sysadm_r staff_r;
   allow sysadm_r user_r;
   allow system_r git_shell_r;
   allow system_r guest_r;
   allow logadm_r system_r;
   allow system_r logadm_r;
   allow system_r nx_server_r;
   allow system_r staff_r;
   allow staff_r logadm_r;
   allow staff_r sysadm_r;
   allow staff_r unconfined_r;
   allow staff_r webadm_r;
   allow unconfined_r system_r;
   allow system_r unconfined_r;
   allow system_r user_r;
   allow webadm_r system_r;
   allow system_r webadm_r;
   allow system_r xguest_r;
```

The **sesearch** utility can provide the number of *allow* rules:

```
~]# sesearch --allow | wc -l
262798
```

And the number of *dontaudit* rules:

```
~]# sesearch --dontaudit | wc -l
156712
```

4.11. Multi-Level Security (MLS)

The Multi-Level Security technology refers to a security scheme that enforces the Bell-La Padula Mandatory Access Model. Under MLS, users and processes are called *subjects*, and files, devices, and other passive components of the system are called *objects*. Both subjects and objects are labeled with a security level, which entails a subject's clearance or an object's classification. Each security level is composed of a *sensitivity* and a *category*, for example, an internal release schedule is filed under the internal documents category with a confidential sensitivity.

Figure 4.1, “Levels of clearance” shows levels of clearance as originally designed by the US defense community. Relating to our internal schedule example above, only users that have gained the confidential clearance are allowed to view documents in the confidential category. However, users who only have the confidential clearance are not allowed to view documents that require higher levels or clearance; they are allowed read access only to documents with lower levels of clearance, and write access to documents with higher levels of clearance.

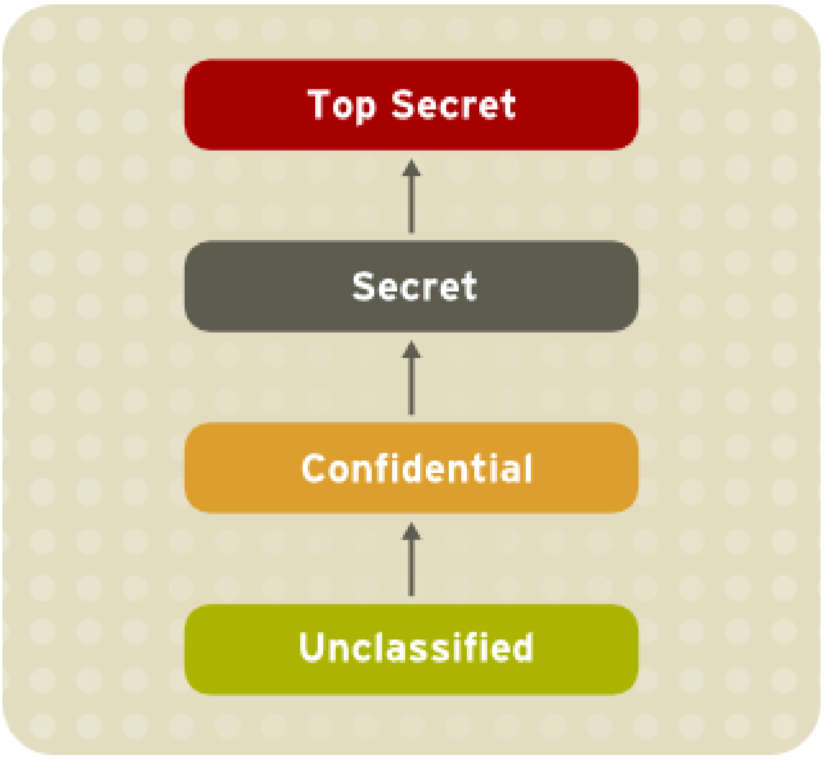


Figure 4.1. Levels of clearance

Figure 4.2, “Allowed data flows using MLS” shows all allowed data flows between a subject running under the "Secret" security level and various objects with different security levels. In simple terms, the Bell-LaPadula model enforces two properties: *no read up* and *no write down*.

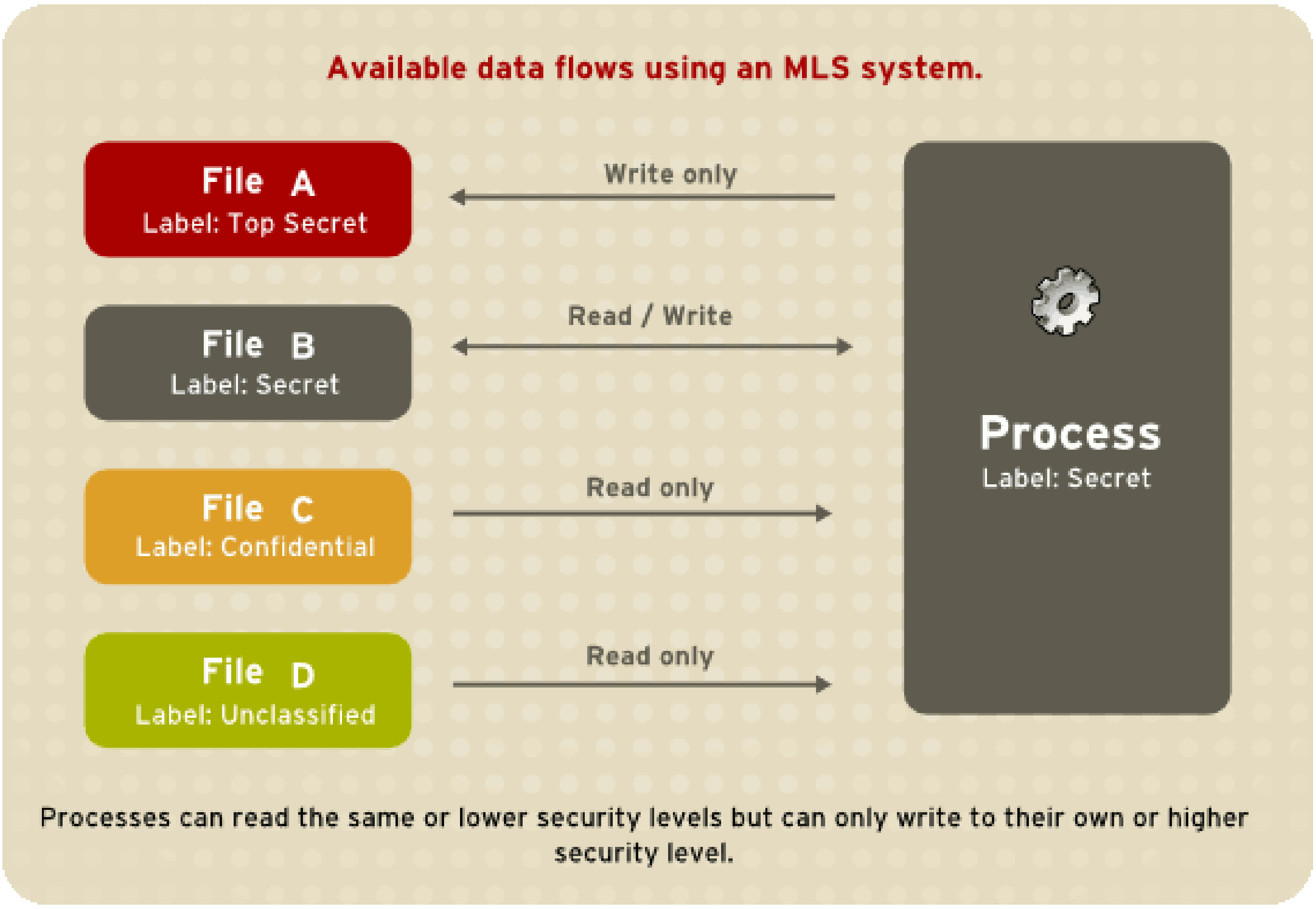


Figure 4.2. Allowed data flows using MLS

4.11.1. MLS and System Privileges

MLS access rules are always combined with conventional access permissions (file 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, this also blocks access by users with a security level of "Top Secret". It is important to remember that SELinux MLS policy rules are checked *after* DAC rules. A higher security clearance does not automatically give permission to arbitrarily browse a file system.

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

4.11.2. Enabling MLS in SELinux

Note

It is not recommended to use the MLS policy on a system that is running the X Window System.

Follow these steps to enable the SELinux MLS policy on your system.

Procedure 4.18. Enabling SELinux MLS Policy

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

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

2. Before the MLS policy is enabled, each file on the file system must be relabeled with an MLS label. When the file system is relabeled, confined domains may be denied access, which may prevent your system from booting correctly. To prevent this from happening, configure **`SELINUX=permissive`** in the **`/etc/selinux/config`** file. Also, enable the MLS policy by configuring **`SELINUXTYPE=mls`**. Your configuration file should look like this:

```
# 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=mls
```

3. Make sure SELinux is running in the permissive mode:

```
~]# setenforce 0
```

```
~]$ getenforce
Permissive
```

4. Create the **`.autorelabel`** file in root's home directory to ensure that files are relabeled upon next reboot:

```
~]# touch /.autorelabel
```

Note that it is necessary to add the **`-F`** option to this file. This can be done by executing the following command:

```
~]# echo "-F" >> /.autorelabel
```

5. Reboot your system. During the next boot, all file systems will be relabeled according to the MLS policy. The label process labels all files with an appropriate SELinux context:

```
*** Warning -- SELinux mls policy relabel is required.
*** Relabeling could take a very long time, depending on file
*** system size and speed of hard drives.
***********
```

Each **`*`** (asterisk) character on the bottom line represents 1000 files that have been labeled. In the above example, eleven **`*`** characters represent 11000 files which have been labeled. The time it takes to label all files depends upon the number of files on the system, and the speed of the hard disk drives. On modern systems, this process can take as little as 10 minutes. Once the labeling process finishes, the system will automatically reboot.

6. In permissive mode, SELinux policy is not enforced, but denials are still logged for actions that would have been denied if running in enforcing mode. Before changing to enforcing mode, as root, run the following command to confirm that SELinux did not deny actions during the last boot. If SELinux did not deny actions during the last boot, this command does not return any output. Refer to Chapter 10, *Troubleshooting* for troubleshooting information if SELinux denied access during boot.

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

7. If there were no denial messages in the **/var/log/messages** file, or you have resolved all existing denials, configure **SELINUX=enforcing** 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=enforcing
# SELINUXTYPE= can take one of these two values:
#       targeted - Targeted processes are protected,
#       mls - Multi Level Security protection.
SELINUXTYPE=mls
```

8. Reboot your system and make sure SELinux is running in enforcing mode:

```
~]$ getenforce
Enforcing
```

and the MLS policy is enabled:

```
~]# sestatus |grep mls
Policy from config file:        mls
```

4.11.3. Creating a User With a Specific MLS Range

Follow these steps to create a new Linux user with a specific MLS range:

Procedure 4.19. Creating a User With a Specific MLS Range

1. Add a new Linux user using the **useradd** command and map the new Linux user to an existing SELinux user (in this case, **user_u**):

```
~]# useradd -Z user_u john
```

2. Assign the newly-created Linux user a password:

```
prompt~]# passwd john
```

3. Run the following command as root to view the mapping between SELinux and Linux users. The output should be as follows:

```
~]# semanage login -l

Login Name          SELinux User         MLS/MCS Range        Service

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

4. Define a specific range for user **john**:

```
~]# semanage login --modify --seuser user_u --range s2:c100 john
```

5. View the mapping between SELinux and Linux users again. Note that the user **john** now has a specific MLS range defined:

```
~]# semanage login -l

Login Name           SELinux User         MLS/MCS Range        Service

__default__          unconfined_u         s0-s0:c0.c1023       *
john                 user_u               s2:c100              *
root                 unconfined_u         s0-s0:c0.c1023       *
system_u             system_u             s0-s0:c0.c1023       *
```

6. To correct the label on john's home directory (if needed), run the following command:

```
~]# chcon -R -l s2:c100 /home/john
```

4.11.4. Setting Up Polyinstantiated Directories

The **/tmp/** and **/var/tmp/** directories are normally used for temporary storage by all programs, services, and users. Such setup, however, makes these directories vulnerable to race condition attacks, or an information leak based on file names. SELinux offers a solution in the form of *polyinstantiated* directories. This effectively means that both **/tmp/** and **/var/tmp/** are instantiated, making them appear private for each user. When instantiation of directories is enabled, each user's **/tmp/** and **/var/tmp/** directory is automatically mounted under **/tmp-inst** and **/var/tmp/tmp-inst**.

Follow these steps to enable polyinstantiation of directories:

Procedure 4.20. Enabling Polyinstantiation Directories

1. Uncomment the last three lines in the **/etc/security/namespace.conf** file to enable instantiation of the **/tmp/**, **/var/tmp/**, and users' home directories:

```
~]$ tail -n 3 /etc/security/namespace.conf
/tmp     /tmp-inst/           level      root,adm
/var/tmp /var/tmp/tmp-inst/   level      root,adm
$HOME    $HOME/$USER.inst/    level
```

2. Ensure that in the **/etc/pam.d/login** file, the **pam_namespace.so** module is configured for session:

```
~]$ grep namespace /etc/pam.d/login
session    required     pam_namespace.so
```

3. Reboot your system.

4.12. File Name Transition

The *file name transition* feature allows policy writers to specify the file name when writing policy transition rules. It is possible to write a rule that states: If a process labeled **A_t** creates a specified object class in a directory labeled **B_t** and the specified object class is named **objectname**, it gets the label **C_t**. This mechanism provides more fine-grained control over processes on the system.

Without file name transition, there are three possible ways how to label an object:

- By default, objects inherit labels from parent directories. For example, if the user creates a file in a directory labeled **etc_t**, then the file is labeled also **etc_t**. However, this method is useless when it is desirable to have multiple files within a directory with different labels.

- Policy writers can write a rule in policy that states: If a process with type **A_t** creates a specified object class in a directory labeled **B_t**, the object gets the new **C_t** label. This practice is problematic if a single program creates multiple objects in the same directory where each object requires a separate label. Moreover, these rules provide only partial control, because names of the created objects are not specified.

- Certain applications have SELinux awareness that allow such an application to ask the system what the label of a certain path should be. These applications then request the kernel to create the object with the required label. Examples of applications with SELinux awareness are the **rpm** package manager, the **restorecon** utility, or the **udev** device manager. However, it is not possible to instruct every application that creates files or directories with SELinux awareness. It is often necessary to relabel objects with the correct label after creating. Otherwise, when a confined domain attempts to use the object, AVC messages are returned.

The file name transition feature decreases problems related to incorrect labeling and improves the system to be more secure. Policy writers are able to state properly that a certain application can only create a file with a specified name in a specified directory. The rules take into account the file name, not the file path. This is the basename of the file path. Note that file name transition uses an exact match done by the **strcmp()** function. Use of regular expressions or wildcard characters is not considered.

Note

File paths can vary in the kernel and file name transition does not use the paths to determine labels. Consequently, this feature only affects initial file creation and does not fix incorrect labels of already created objects.

Example 4.1. Examples of Policy Rules Written with File Name Transition

The example below shows a policy rule with file name transition:

```
filetrans_pattern(unconfined_t, admin_home_t, ssh_home_t, dir, ".ssh")
```

This rule states that if a process with the **unconfined_t** type creates the **~/.ssh/** directory in a directory labeled **admin_home_t**, the **~/.ssh/** directory gets the label **ssh_home_t**.

Similar examples of policy rules written with file name transition are presented below:

```
filetrans_pattern(staff_t, user_home_dir_t, httpd_user_content_t, dir,
"public_html")
filetrans_pattern(thumb_t, user_home_dir_t, thumb_home_t, file, "missfont.log")
filetrans_pattern(kernel_t, device_t, xserver_misc_device_t, chr_file, "nvidia0")
filetrans_pattern(puppet_t, etc_t, krb5_conf_t, file, "krb5.conf")
```

Note

The file name transition feature affects mainly policy writers, but users can notice that instead of file objects almost always created with the default label of the containing directory, some file objects have a different label as specified in policy.

4.13. Disable ptrace()

The **`ptrace()`** system call allows one process to observe and control the execution of another process and change its memory and registers. This call is used primarily by developers during debugging. When **`ptrace()`** is not needed, it can be disabled to improve system security. This can be done by enabling the **`deny_ptrace`** Boolean, which denies all processes, even those that are running in **`unconfined_t`** domains, from being able to use **`ptrace()`** on other processes.

The **`deny_ptrace`** Boolean is disabled by default. To enable it, run the **`setsebool -P deny_ptrace on`** command as the root user:

```
~]# setsebool -P deny_ptrace on
```

To verify if this Boolean is enabled, use the **`getsetbool`**:

```
~]$ getsebool deny_ptrace
deny_ptrace --> on
```

To disable this Boolean, run the **`setsebool -P deny_ptrace off`** command as root:

```
~]# setsebool -P deny_ptrace off
```

Note

The **`setsebool -P`** command makes persistent changes. Do not use the **`-P`** option if you do not want changes to persist across reboots.

This Boolean influences only packages that are part of Red Hat Enterprise Linux. Consequently, third-party packages could still use the **`ptrace()`** system call. To list all domains that are allowed to use **`ptrace()`**, run the following command:

```
~]# sesearch -A -p ptrace,sys_ptrace -C | grep -v deny_ptrace | cut -d ' ' -f 5
```

4.14. Thumbnail Protection

The thumbnail icons can potentially allow an attacker to break into a locked machine using removable media, such as USB devices or CDs. When the system detects a removable media, the Nautilus file manager executes the thumbnail driver code to display thumbnail icons in an appropriate file browser even if the machine is locked. This behavior is unsafe because if the thumbnail executables were vulnerable, the attacker could use the thumbnail driver code to bypass the lock screen without entering the password.

Therefore, a new SELinux policy is used to prevent such attacks. This policy ensures that all thumbnail drivers are locked when the screen is locked. The thumbnail protection is enabled for both confined users and unconfined users. This policy affects the following applications:

- /usr/bin/evince-thumbnailer
- /usr/bin/ffmpegthumbnailer
- /usr/bin/gnome-exe-thumbnailer.sh
- /usr/bin/gnome-nds-thumbnailer
- /usr/bin/gnome-xcf-thumbnailer
- /usr/bin/gsf-office-thumbnailer
- /usr/bin/raw-thumbnailer
- /usr/bin/shotwell-video-thumbnailer
- /usr/bin/totem-video-thumbnailer
- /usr/bin/whaaw-thumbnailer
- /usr/lib/tumbler-1/tumblerd
- /usr/lib64/tumbler-1/tumblerd

---

[6] To temporarily revert to the default behavior, as the Linux root user, run the **`setsebool httpd_can_network_connect_db off`** command. For changes that persist across reboots, run the **`setsebool -P httpd_can_network_connect_db off`** command.

[7] Files in the **`/etc/selinux/targeted/contexts/files/`** directory define contexts for files and directories. Files in this directory are read by the **`restorecon`** and **`setfiles`** utilities to restore files and directories to their default contexts.

[8] Morris, James. "Filesystem Labeling in SELinux". Published 1 October 2004. Accessed 14 October 2008: http://www.linuxjournal.com/article/7426.

[9] Refer to the matchpathcon(8) manual page for further information about **`matchpathcon`**.

Chapter 5. The sepolicy Suite

The **sepolicy** utility provides a suite of features to query the installed SELinux policy. These features are either new or were previously provided by separate utilities, such as **sepolgen** or **setrans**. The suite allows you to generate transition reports, man pages, or even new policy modules, thus giving users easier access and better understanding of the SELinux policy.

The *policycoreutils-devel* package provides **sepolicy**. Run the following command as the root user to install **sepolicy**:

```
~]# yum install policycoreutils-devel
```

The **sepolicy** suite provides the following features that are invoked as command-line parameters:

Table 5.1. The sepolicy Features

Feature	Description
booleans	Query the SELinux Policy to see description of Booleans
communicate	Query the SELinux policy to see if domains can communicate with each other
generate	Generate an SELinux policy module template
gui	Graphical User Interface for SELinux Policy
interface	List SELinux Policy interfaces
manpage	Generate SELinux man pages
network	Query SELinux policy network information
transition	Query SELinux policy and generate a process transition report

5.1. The sepolicy Python Bindings

In previous versions of Red Hat Enterprise Linux, the *setools* package included the **sesearch** and **seinfo** utilities. The **sesearch** utility is used for searching rules in a SELinux policy while the **seinfo** utility allows you to query various other components in the policy.

In Red Hat Enterprise Linux 7, Python bindings for **sesearch** and **seinfo** have been added so that you can use the functionality of these utilities via the **sepolicy** suite. See the example below:

```
> python
>>> import sepolicy
>>> sepolicy.info(sepolicy.ATTRIBUTE)
Returns a dictionary of all information about SELinux Attributes
>>>sepolicy.search([sepolicy.ALLOW])
Returns a dictionary of all allow rules in the policy.
```

5.2. Generating SELinux Policy Modules: sepolicy generate

In previous versions of Red Hat Enterprise Linux, the **sepolgen** or **selinux-polgengui** utilities were used for generating a SELinux policy. These tools have been merged to the **sepolicy** suite. In Red Hat Enterprise Linux 7, the **sepolicy generate** command is used to generate an initial SELinux policy module template.

Unlike **sepolgen**, it is not necessary to run **sepolicy generate** as the root user. This utility also creates an RPM spec file, which can be used to build an RPM package that installs the policy package file (***NAME*.pp**) and the interface file (***NAME*.if**) to the correct location, provides installation of the SELinux policy into the kernel, and fixes the labeling. The setup script continues to install SELinux policy and sets up the labeling. In addition, a manual page based on the installed policy is generated using the **sepolicy manpage** command. [10] Finally, **sepolicy generate** builds and compiles the SELinux policy and the manual page into an RPM package, ready to be installed on other systems.

When **sepolicy generate** is executed, the following files are produced:

***NAME*.te – type enforcing file**

This file defines all the types and rules for a particular domain.

***NAME*.if – interface file**

This file defines the default file context for the system. It takes the file types created in the **NAME.te** file and associates file paths to the types. Utilities, such as **restorecon** and **rpm**, use these paths to write labels.

***NAME*_selinux.spec – RPM spec file**

This file is an RPM spec file that installs SELinux policy and sets up the labeling. This file also installs the interface file and a man page describing the policy. You can use the **sepolicy manpage -d *NAME*** command to generate the man page.

***NAME*.sh – helper shell script**

This script helps to compile, install, and fix the labeling on the system. It also generates a man page based on the installed policy, compiles, and builds an RPM package suitable to be installed on other systems.

If it is possible to generate an SELinux policy module, **sepolicy generate** prints out all generated paths from the source domain to the target domain. See the sepolicy-generate(8) manual page for further information about **sepolicy generate**.

5.3. Understanding Domain Transitions: sepolicy transition

Previously, the **setrans** utility was used to examine if transition between two domain or process types is possible and printed out all intermediary types that are used to transition between these domains or processes. In Red Hat Enterprise Linux 7, **setrans** is provided as part of the **sepolicy** suite and the **sepolicy transition** command is now used instead.

The **sepolicy transition** command queries a SELinux policy and creates a process transition report. The **sepolicy transition** command requires two command-line arguments – a source domain (specified by the **-s** option) and a target domain (specified by the **-t** option). If only the source domain is entered, **sepolicy transition** lists all possible domains that the source domain can transition to. The following output does not contain all entries. The “@” character means “execute”:

```
~]$ sepolicy transition -s httpd_t
httpd_t @ httpd_suexec_exec_t --> httpd_suexec_t
httpd_t @ mailman_cgi_exec_t --> mailman_cgi_t
httpd_t @ abrt_retrace_worker_exec_t --> abrt_retrace_worker_t
httpd_t @ dirsrvadmin_unconfined_script_exec_t --> dirsrvadmin_unconfined_script_t
httpd_t @ httpd_unconfined_script_exec_t --> httpd_unconfined_script_t
```

If the target domain is specified, **sepolicy transition** examines SELinux policy for all transition paths from the source domain to the target domain and lists these paths. The output below is not complete:

```
~]$ sepolicy transition -s httpd_t -t system_mail_t
httpd_t @ exim_exec_t --> system_mail_t
httpd_t @ courier_exec_t --> system_mail_t
httpd_t @ sendmail_exec_t --> system_mail_t
httpd_t ... httpd_suexec_t @ sendmail_exec_t --> system_mail_t
httpd_t ... httpd_suexec_t @ exim_exec_t --> system_mail_t
httpd_t ... httpd_suexec_t @ courier_exec_t --> system_mail_t
httpd_t ... httpd_suexec_t ... httpd_mojomojo_script_t @ sendmail_exec_t -->
system_mail_t
```

See the sepolicy-transition(8) manual page for further information about **`sepolicy transition`**.

5.4. Generating Manual Pages: `sepolicy manpage`

The **`sepolicy manpage`** command generates manual pages based on the SELinux policy that document process domains. As a result, such documentation is always up-to-date. Each name of automatically generated manual pages consists of the process domain name and the **`_selinux`** suffix, for example **`httpd_selinux`**.

The manual pages include several sections that provide information about various parts of the SELinux policy for confined domains:

- The **`Entrypoints`** section contains all executable files that need to be executed during a domain transition.
- The **`Process Types`** section lists all process types that begin with the same prefix as the target domain.
- The **`Booleans`** section lists Booleans associated with the domain.
- The **`Port Types`** section contains the port types matching the same prefix as the domain and describes the default port numbers assigned to these port types.
- The **`Managed Files`** section describes the types that the domain is allowed to write to and the default paths associated with these types.
- The **`File Contexts`** section contains all file types associated with the domain and describes how to use these file types along with the default path labeling on a system.
- The **`Sharing Files`** section explains how to use the domain sharing types, such as **`public_content_t`**.

See the sepolicy-manpage(8) manual page for further information about **`sepolicy manpage`**.

[10] See Section 5.4, “Generating Manual Pages: **`sepolicy manpage`**” for more information about **`sepolicy manpage`**.

Chapter 6. Confining Users

A number of confined SELinux users are available in Red Hat Enterprise Linux. Each Linux user is mapped to an SELinux user using SELinux policy, allowing Linux users to inherit the restrictions placed on SELinux users, for example (depending on the user), not being able to: run the X Window System; use networking; run setuid applications (unless SELinux policy permits it); or run the **su** and **sudo** commands. This helps protect the system from the user. Refer to Section 3.3, "Confined and Unconfined Users" for further information about confined users.

6.1. Linux and SELinux User Mappings

As the root user, run the following command to view the mapping between Linux users and 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       *
system_u            system_u             s0-s0:c0.c1023       *
```

In Red Hat Enterprise Linux, Linux users are mapped to the SELinux **__default__** login by default (which is in turn mapped to the SELinux **unconfined_u** user). When a Linux user is created with the **useradd** command, if no options are specified, they are mapped to the SELinux **unconfined_u** user. The following defines the default-mapping:

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

6.2. Confining New Linux Users: useradd

Linux users mapped to the SELinux **unconfined_u** user run in the **unconfined_t** domain. This is seen by running the **id -Z** command while logged-in as a Linux user mapped to **unconfined_u**:

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

When Linux users run in the **unconfined_t** domain, SELinux policy rules are applied, but policy rules exist that allow Linux users running in the **unconfined_t** domain almost all access. If unconfined Linux users execute an application that SELinux policy defines can transition from the **unconfined_t** domain to its own confined domain, unconfined Linux users are 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, and therefore, the exploitation of a flaw in the application can be limited by policy.

Note

This does not protect the system from the user. Instead, the user and the system are being protected from possible damage caused by a flaw in the application.

When creating Linux users with the **useradd** command, use the **-Z** option to specify which SELinux user they are mapped to. The following example creates a new Linux user, **useruuser**, and maps that user to the SELinux **user_u** user. Linux users mapped to the SELinux **user_u** user run in the **user_t** domain. In this domain, Linux users are unable to run setuid applications unless SELinux policy permits it (such as **passwd**), and cannot run the **su** or **sudo** command, preventing them from becoming the root user with these commands.

Procedure 6.1. Confining a New Linux User to user_u SELinux User

1. As root, create a new Linux user (**useruuser**) that is mapped to the SELinux **user_u** user.

```
~]# useradd -Z user_u useruuser
```

2. To view the mapping between **useruuser** and **user_u**, run the following 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       *
system_u             system_u             s0-s0:c0.c1023       *
useruuser            user_u               s0                   *
```

3. As root, assign a password to the Linux **useruuser** user:

```
~]# passwd useruuser
Changing password for user useruuser.
New password: Enter a password
Retype new password: Enter the same password again
passwd: all authentication tokens updated successfully.
```

4. Log out of your current session, and log in as the Linux **useruuser** user. When you log in, the **pam_selinux** module maps the Linux user to an SELinux user (in this case, **user_u**), and sets up the resulting SELinux context. The Linux user's shell is then launched with this context. Run the following command to view the context of a Linux user:

```
~]$ id -Z
user_u:user_r:user_t:s0
```

5. Log out of the Linux **useruuser**'s session, and log back in with your account. If you do not want the Linux **useruuser** user, run the following command as root to remove it, along with its home directory:

```
~]# userdel -r useruuser
```

6.3. Confining Existing Linux Users: semanage login

If a Linux user is mapped to the SELinux **unconfined_u** user (the default behavior), and you would like to change which SELinux user they are mapped to, use the **semanage login** command. The following example creates a new Linux user named **newuser**, then maps that Linux user to the SELinux **user_u** user:

Procedure 6.2. Mapping Linux Users to the SELinux Users

1. As the root user, create a new Linux user (**newuser**). Since this user uses the default mapping, it does not appear in the **semanage login -l** output:

   ```
   ~]# useradd newuser
   ```

   ```
   ~]# 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       *
   system_u             system_u             s0-s0:c0.c1023       *
   ```

2. To map the Linux **newuser** user to the SELinux **user_u** user, run the following command as root:

   ```
   ~]# semanage login -a -s user_u newuser
   ```

   The **-a** option adds a new record, and the **-s** option specifies the SELinux user to map a Linux user to. The last argument, **newuser**, is the Linux user you want mapped to the specified SELinux user.

3. To view the mapping between the Linux **newuser** user and **user_u**, use the **semanage** utility again:

   ```
   ~]# semanage login -l

   Login Name           SELinux User         MLS/MCS Range        Service

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

4. As root, assign a password to the Linux **newuser** user:

   ```
   ~]# passwd newuser
   Changing password for user newuser.
   New password: Enter a password
   Retype new password: Enter the same password again
   passwd: all authentication tokens updated successfully.
   ```

5. Log out of your current session, and log in as the Linux **newuser** user. Run the following command to view the **newuser**'s SELinux context:

   ```
   ~]$ id -Z
   user_u:user_r:user_t:s0
   ```

6. Log out of the Linux **newuser**'s session, and log back in with your account. If you do not want the Linux **newuser** user, run the following command as root to remove it, along with its home directory:

   ```
   ~]# userdel -r newuser
   ```

   As root, remove the mapping between the Linux **newuser** user and **user_u**:

   ```
   ~]# semanage login -d newuser
   ```

```
~]# 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       *
system_u             system_u             s0-s0:c0.c1023       *
```

6.4. Changing the Default Mapping

In Red Hat Enterprise Linux, Linux users are mapped to the SELinux **`__default__`** login by default (which is in turn mapped to the SELinux **`unconfined_u`** user). If you would like new Linux users, and Linux users not specifically mapped to an SELinux user to be confined by default, change the default mapping with the **`semanage login`** command.

For example, run the following command as root to change the default mapping from **`unconfined_u`** to **`user_u`**:

```
~]# semanage login -m -S targeted -s "user_u" -r s0 __default__
```

Verify the **`__default__`** login is mapped to **`user_u`**:

```
~]# semanage login -l

Login Name           SELinux User         MLS/MCS Range        Service

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

If a new Linux user is created and an SELinux user is not specified, or if an existing Linux user logs in and does not match a specific entry from the **`semanage login -l`** output, they are mapped to **`user_u`**, as per the **`__default__`** login.

To change back to the default behavior, run the following command as root to map the **`__default__`** login to the SELinux **`unconfined_u`** user:

```
~]# semanage login -m -S targeted -s "unconfined_u" -r s0-s0:c0.c1023 __default__
```

6.5. xguest: Kiosk Mode

The *xguest* package provides a kiosk user account. This account is used to secure machines that people walk up to and use, such as those at libraries, banks, airports, information kiosks, and coffee shops. The kiosk user account is very limited: essentially, it only allows users to log in and use **Firefox** to browse Internet websites. Any changes made while logged in with this account, such as creating files or changing settings, are lost when you log out.

To set up the kiosk account:

1. As the root user, install the *xguest* package. Install dependencies as required:

   ```
   ~]# yum install xguest
   ```

2. In order to allow the kiosk account to be used by a variety of people, the account is not password-protected, and as such, the account can only be protected if SELinux is running in enforcing mode. Before logging in with this account, use the **getenforce** utility to confirm that SELinux is running in enforcing mode:

   ```
   ~]$ getenforce
   Enforcing
   ```

   If this is not the case, see Section 4.4, “Enabling and Disabling SELinux” for information about changing to enforcing mode. It is not possible to log in with this account if SELinux is in permissive mode or disabled.

3. You can only log in to this account via the GNOME Display Manager (GDM). Once the *xguest* package is installed, a **Guest** account is added to the GDM login screen.

6.6. Booleans for Users Executing Applications

Not allowing Linux users to execute applications (which inherit users' permissions) in their home directories and the **/tmp/** directory, which they have write access to, helps prevent flawed or malicious applications from modifying files that users own. In Red Hat Enterprise Linux, by default, Linux users in the **guest_t** and **xguest_t** domains cannot execute applications in their home directories or **/tmp/**; however, by default, Linux users in the **user_t** and **staff_t** domains can.

Booleans are available to change this behavior, and are configured with the **setsebool** utility, which must be run as the root user. The **setsebool -P** command makes persistent changes. Do not use the **-P** option if you do not want changes to persist across reboots:

guest_t

To *allow* Linux users in the **guest_t** domain to execute applications in their home directories and **/tmp/**:

```
~]# setsebool -P guest_exec_content on
```

xguest_t

To *allow* Linux users in the **xguest_t** domain to execute applications in their home directories and **/tmp/**:

```
~]# setsebool -P xguest_exec_content on
```

user_t

To *prevent* Linux users in the **user_t** domain from executing applications in their home directories and **/tmp/**:

```
~]# setsebool -P user_exec_content off
```

staff_t

To *prevent* Linux users in the **staff_t** domain from executing applications in their home directories and **/tmp/**:

```
~]# setsebool -P staff_exec_content off
```

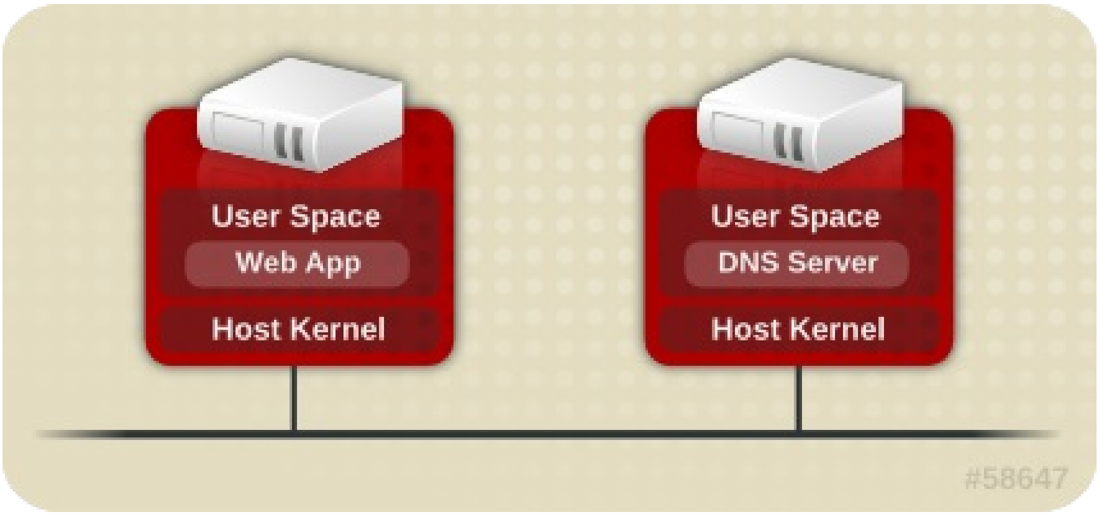
Chapter 7. sVirt

sVirt is a technology included in Red Hat Enterprise Linux that integrates SELinux and virtualization. sVirt applies Mandatory Access Control (MAC) to improve security when using virtual machines. The main reasons for integrating these technologies are to improve security and harden the system against bugs in the hypervisor that might be used as an attack vector aimed toward the host or to another virtual machine.

This chapter describes how sVirt integrates with virtualization technologies in Red Hat Enterprise Linux.

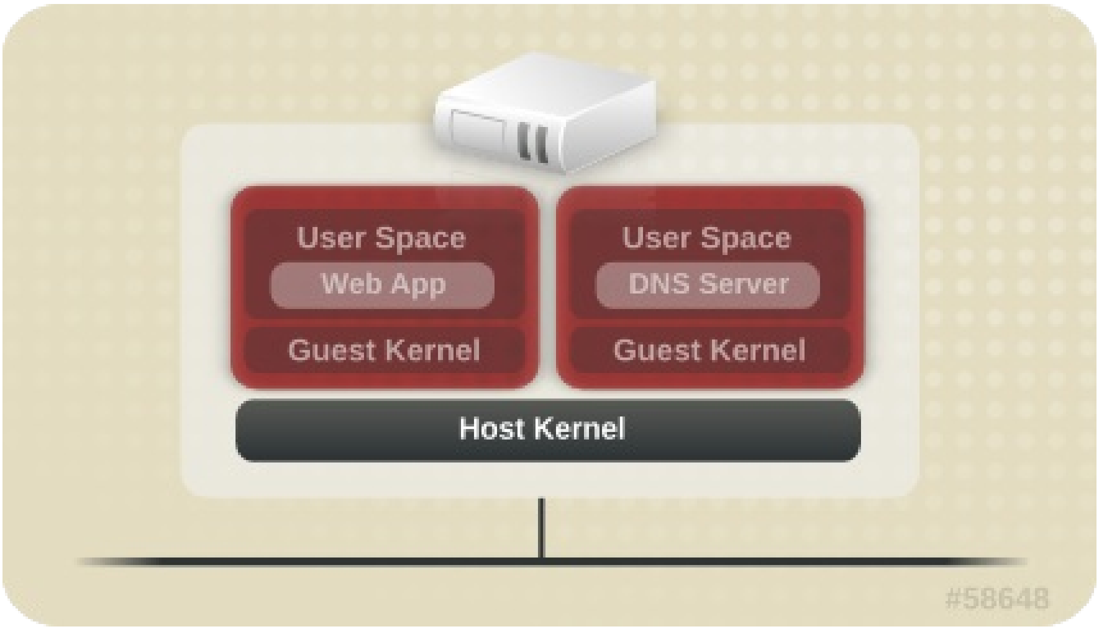
Non-Virtualized Environment

In a non-virtualized environment, hosts are separated from each other physically and each host has a self-contained environment, consisting of services such as a Web server, or a DNS server. These services communicate directly to their own user space, host kernel and physical host, offering their services directly to the network. The following image represents a non-virtualized environment:



Virtualized Environment

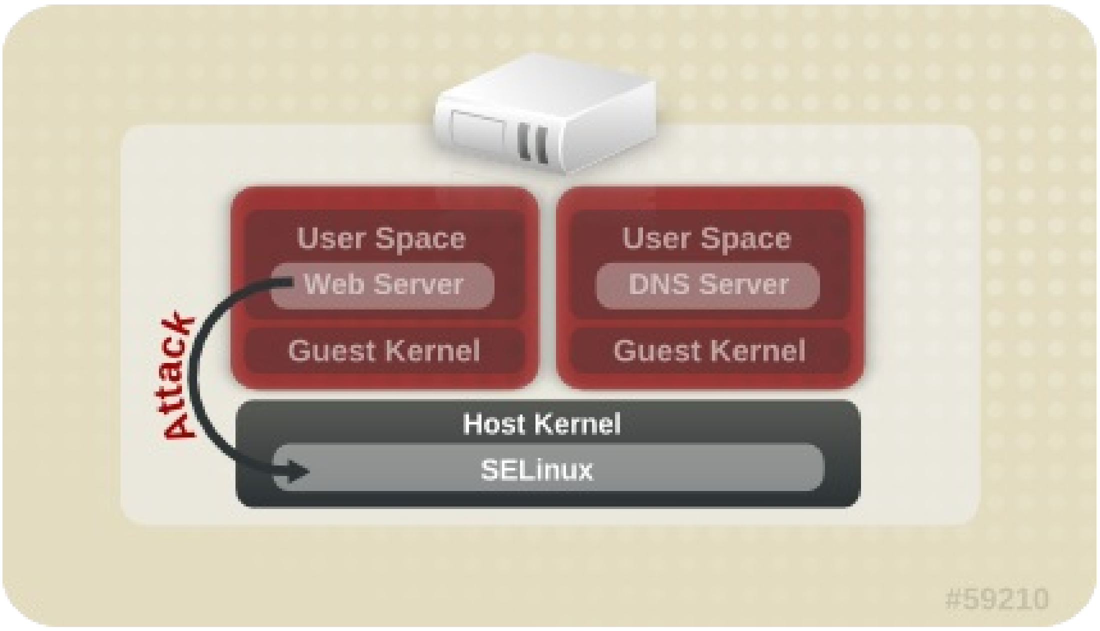
In a virtualized environment, several operating systems can be housed (as "guests") within a single host kernel and physical host. The following image represents a virtualized environment:



7.1. Security and Virtualization

When services are not virtualized, machines are physically separated. Any exploit is usually contained to the affected machine, with the obvious exception of network attacks. When services are grouped together in a virtualized environment, extra vulnerabilities emerge in the system. If there is a security flaw in the hypervisor that can be exploited by a guest instance, this guest may be able to not only attack the host, but also other guests running on that host. This is not theoretical; attacks already exist on hypervisors. These attacks can extend beyond the guest instance and could expose other guests to attack.

sVirt is an effort to isolate guests and limit their ability to launch further attacks if exploited. This is demonstrated in the following image, where an attack cannot break out of the virtual machine and extend to another host instance:



SELinux introduces a pluggable security framework for virtualized instances in its implementation of Mandatory Access Control (MAC). The sVirt framework allows guests and their resources to be uniquely labeled. Once labeled, rules can be applied which can reject access between different guests.

7.2. sVirt Labeling

Like other services under the protection of SELinux, sVirt uses process-based mechanisms and restrictions to provide an extra layer of security over guest instances. Under typical use, you should not even notice that sVirt is working in the background. This section describes the labeling features of sVirt.

As shown in the following output, when using sVirt, each Virtual Machine (VM) process is labeled and runs with a dynamically generated level. Each process is isolated from other VMs with different levels:

```
~]# ps -eZ | grep qemu

system_u:system_r:svirt_t:s0:c87,c520 27950 ?  00:00:17 qemu-kvm
system_u:system_r:svirt_t:s0:c639,c757 27989 ? 00:00:06 qemu-system-x86
```

The actual disk images are automatically labeled to match the processes, as shown in the following output:

```
~]# ls -lZ /var/lib/libvirt/images/*

system_u:object_r:svirt_image_t:s0:c87,c520   image1
```

The following table outlines the different labels that can be assigned when using sVirt:

Table 7.1. sVirt Labels

Type	SELinux Context	Description
Virtual Machine Processes	system_u:system_r:svirt_t:MCS1	MCS1 is a randomly selected MCS field. Currently approximately 500,000 labels are supported.
Virtual Machine Image	system_u:object_r:svirt_image_t:MCS1	Only processes labeled *svirt_t* with the same MCS fields are able to read/write these image files and devices.
Virtual Machine Shared Read/Write Content	system_u:object_r:svirt_image_t:s0	All processes labeled *svirt_t* are allowed to write to the svirt_image_t:s0 files and devices.
Virtual Machine Image	system_u:object_r:virt_content_t:s0	System default label used when an image exits. No *svirt_t* virtual processes are allowed to read files/devices with this label.

It is also possible to perform static labeling when using sVirt. Static labels allow the administrator to select a specific label, including the MCS/MLS field, for a virtual machine. Administrators who run statically-labeled virtual machines are responsible for setting the correct label on the image files. The virtual machine will always be started with that label, and the sVirt system will never modify the label of a statically-labeled virtual machine's content. This allows the sVirt component to run in an MLS environment. You can also run multiple virtual machines with different sensitivity levels on a system, depending on your requirements.

Chapter 8. Secure Linux Containers

Linux Containers (LXC) is a low-level virtualization feature that allows you to run multiple copies of the same service at the same time on a system. Compared to full virtualization, containers do not require an entire new system to boot, can use less memory, and can use the base operating system in a read-only manner. For example, LXC allow you to run multiple web servers simultaneously, each with their own data while sharing the system data, and even running as the root user. However, running a privileged process within a container could affect other processes running outside of the container or processes running in other containers. Secure Linux containers use the SELinux context, therefore preventing the processes running within them from interacting with each other or with the host.

The **Docker** application is the main utility for managing Linux Containers in Red Hat Enterprise Linux. As an alternative, you can also use the **virsh** command-line utility provided by the *libvirt* package.

For further details about Linux Containers see the Resource Management and Linux Containers Guide.

Chapter 9. SELinux systemd Access Control

In Red Hat Enterprise Linux 7, system services are controlled by the **systemd** daemon. In previous releases of Red Hat Enterprise Linux, daemons could be started in two ways:

- At boot time, the System V **init** daemon launched an **init.rc** script and then this script launched the desired daemon. For example, the Apache server, which was started at boot, got the following SELinux label:

  ```
  system_u:system_r:httpd_t:s0
  ```

- An administrator launched the **init.rc** script manually, causing the daemon to run. For example, when the **systemctl restart httpd.service** command was invoked on the Apache server, the resulting SELinux label looked as follows:

  ```
  unconfined_u:system_r:httpd_t:s0
  ```

When launched manually, the process adopted the user portion of the SELinux label that started it, making the labeling in the two scenarios above inconsistent. With the **systemd** daemon, the transitions are very different. As **systemd** handles all the calls to start and stop daemons on the system, using the **init_t** type, it can override the user part of the label when a daemon is restarted manually. As a result, the labels in both scenarios above are **system_u:system_r:httpd_t:s0** as expected and the SELinux policy could be improved to govern which domains are able to control which units.

9.1. SELinux Access Permissions for Services

In previous versions of Red Hat Enterprise Linux, an administrator was able to control, which users or applications were able to start or stop services based on the label of the System V Init script. Now, **systemd** starts and stops all services, and users and processes communicate with **systemd** using the **systemctl** utility. The **systemd** daemon has the ability to 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.

For example, previously, administrators had to allow NetworkManager to execute **systemctl** to send a D-Bus message to **systemd**, which would in turn start or stop whatever service NetworkManager requested. In fact, NetworkManager was allowed to do everything **systemctl** could do. It was also impossible to setup confined administrators so that they could start or stop just particular services.

To fix these issues, **systemd** also works as an SELinux Access Manager. It can retrieve 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 via SELinux. Policy writers can also use these fine-grained controls to confine administrators. Policy changes involve a new class called **service**, with the following permissions:

```
class service
{
       start
       stop
```

```
        status
        reload
        kill
        load
        enable
        disable
}
```

For example, a policy writer can now allow a domain to get the status of a service or start and stop a service, but not enable or disable a service. Access control operations in SELinux and **`systemd`** do not match in all cases. A mapping was defined to line up **`systemd`** method calls with SELinux access checks. Table 9.1, “Mapping of systemd unit file method calls on SELinux access checks” maps access checks on unit files while Table 9.2, “Mapping of systemd general system calls on SELinux access checks” covers access checks for the system in general. If no match is found in either table, then the **`undefined`** system check is called.

Table 9.1. Mapping of systemd unit file method calls on SELinux access checks

`systemd` unit file method	SELinux access check
DisableUnitFiles	disable
EnableUnitFiles	enable
GetUnit	status
GetUnitByPID	status
GetUnitFileState	status
Kill	stop
KillUnit	stop
LinkUnitFiles	enable
ListUnits	status
LoadUnit	status
MaskUnitFiles	disable
PresetUnitFiles	enable
ReenableUnitFiles	enable
Reexecute	start
Reload	reload
ReloadOrRestart	start
ReloadOrRestartUnit	start
ReloadOrTryRestart	start
ReloadOrTryRestartUnit	start
ReloadUnit	reload
ResetFailed	stop
ResetFailedUnit	stop
Restart	start
RestartUnit	start
Start	start
StartUnit	start
StartUnitReplace	start
Stop	stop
StopUnit	stop
TryRestart	start
TryRestartUnit	start
UnmaskUnitFiles	enable

Table 9.2. Mapping of systemd general system calls on SELinux access checks

`systemd` general system call	SELinux access check
ClearJobs	reboot
FlushDevices	halt
Get	status
GetAll	status
GetJob	status
GetSeat	status
GetSession	status
GetSessionByPID	status
GetUser	status
Halt	halt
Introspect	status
KExec	reboot
KillSession	halt
KillUser	halt
ListJobs	status
ListSeats	status
ListSessions	status
ListUsers	status
LockSession	halt
PowerOff	halt
Reboot	reboot
SetUserLinger	halt
TerminateSeat	halt
TerminateSession	halt
TerminateUser	halt

Example 9.1. SELinux Policy for a System Service

By using the **`sesearch`** utility, you can list policy rules for a system service. For example, calling the **`sesearch -A -s NetworkManager_t -c service`** command returns:

```
allow NetworkManager_t dnsmasq_unit_file_t : service { start stop status reload
kill load } ;
allow NetworkManager_t nscd_unit_file_t : service { start stop status reload kill
load } ;
allow NetworkManager_t ntpd_unit_file_t : service { start stop status reload kill
load } ;
allow NetworkManager_t pppd_unit_file_t : service { start stop status reload kill
load } ;
allow NetworkManager_t polipo_unit_file_t : service { start stop status reload
kill load } ;
```

9.2. SELinux and `journald`

In **`systemd`**, the **`journald`** daemon (also known as **`systemd-journal`**) is the alternative for the **`syslog`** utility, which is a system service that collects and stores logging data. It creates and maintains structured and indexed journals based on logging information that is received from the kernel, from user processes using the **`libc syslog()`** function, from standard and error output of system services, or using its native API. It implicitly collects numerous metadata fields for each log message in a secure way.

The **systemd-journal** service can be used with SELinux to increase security. SELinux controls processes by only allowing them to do what they were designed to do; sometimes even less, depending on the security goals of the policy writer. For example, SELinux prevents a compromised **ntpd** process from doing anything other than handle Network Time. However, the **ntpd** process sends **syslog** messages, so that SELinux would allow the compromised process to continue to send those messages. The compromised **ntpd** could format **syslog** messages to match other daemons and potentially mislead an administrator, or even worse, a utility that reads the **syslog** file into compromising the whole system.

The **systemd-journal** daemon verifies all log messages and, among other things, adds SELinux labels to them. It is then easy to detect inconsistencies in log messages and prevent an attack of this type before it occurs. You can use the **journalctl** utility to query logs of **systemd** journals. If no command-line arguments are specified, running this utility lists the full content of the journal, starting from the oldest entries. To see all logs generated on the system, including logs for system components, execute **journalctl** as root. If you execute it as a non-root user, the output will be limited only to logs related to the currently logged-in user.

Example 9.2. Listing Logs with journalctl

It is possible to use **journalctl** for listing all logs related to a particular SELinux label. For example, the following command lists all logs logged under the **system_u:system_r:policykit_t:s0** label:

```
~]# journalctl _SELINUX_CONTEXT=system_u:system_r:policykit_t:s0
Oct 21 10:22:42 localhost.localdomain polkitd[647]: Started polkitd version 0.112
Oct 21 10:22:44 localhost.localdomain polkitd[647]: Loading rules from directory
/etc/polkit-1/rules.d
Oct 21 10:22:44 localhost.localdomain polkitd[647]: Loading rules from directory
/usr/share/polkit-1/rules.d
Oct 21 10:22:44 localhost.localdomain polkitd[647]: Finished loading, compiling
and executing 5 rules
Oct 21 10:22:44 localhost.localdomain polkitd[647]: Acquired the name
org.freedesktop.PolicyKit1 on the system bus Oct 21 10:23:10 localhost
polkitd[647]: Registered Authentication Agent for unix-session:c1 (system bus name
:1.49, object path /org/freedesktop/PolicyKit1/AuthenticationAgent, locale
en_US.UTF-8) (disconnected from bus)
Oct 21 10:23:35 localhost polkitd[647]: Unregistered Authentication Agent for
unix-session:c1 (system bus name :1.80 [/usr/bin/gnome-shell --mode=classic],
object path /org/freedesktop/PolicyKit1/AuthenticationAgent, locale en_US.utf8)
```

For more information about **journalctl**, see the journalctl(1) manual page.

Chapter 10. Troubleshooting

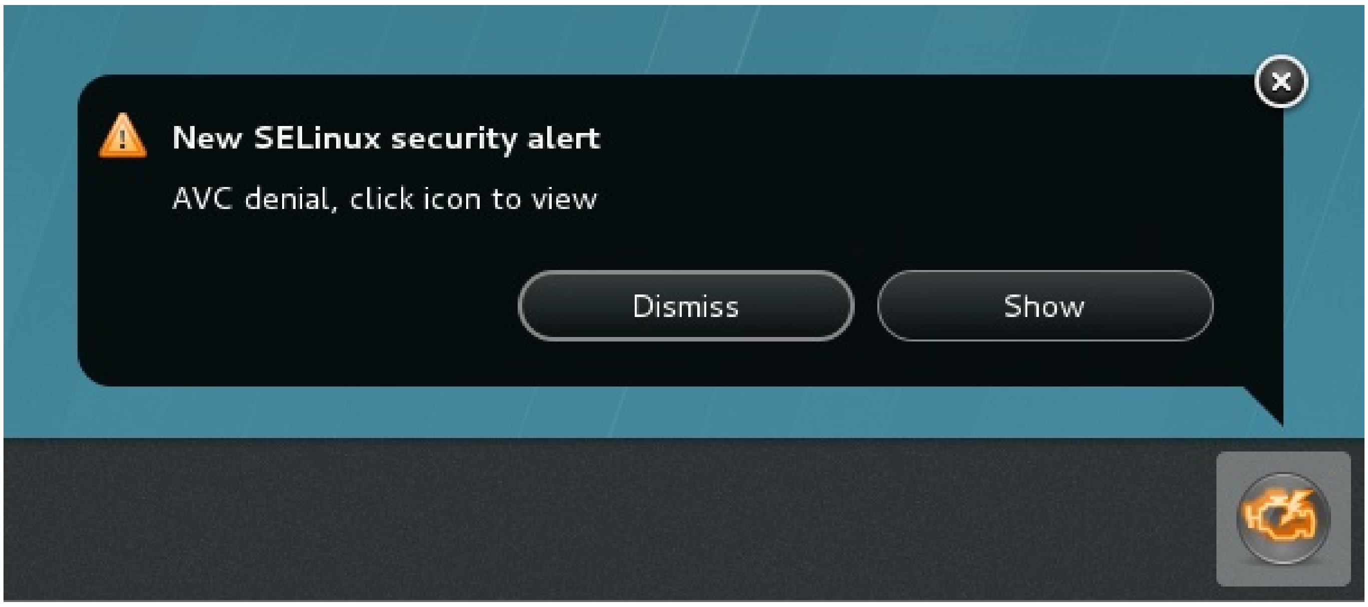
The following chapter describes what happens when SELinux denies access; the top three causes of problems; where to find information about correct labeling; analyzing SELinux denials; and creating custom policy modules with **`audit2allow`**.

10.1. What Happens when Access is Denied

SELinux decisions, such as allowing or disallowing access, are cached. This cache is known as the Access Vector Cache (AVC). Denial messages are logged when SELinux denies access. These denials are also known as "AVC denials", and are logged to a different location, depending on which daemons are running:

Daemon	Log Location
auditd on	**`/var/log/audit/audit.log`**
auditd off; rsyslogd on	**`/var/log/messages`**
setroubleshootd, rsyslogd, and auditd on	**`/var/log/audit/audit.log`**. Easier-to-read denial messages also sent to **`/var/log/messages`**

If you are running the X Window System, have the *setroubleshoot* and *setroubleshoot-server* packages installed, and the **`setroubleshootd`** and **`auditd`** daemons are running, a warning is displayed when access is denied by SELinux:



New SELinux security alert

AVC denial, click icon to view

Clicking on **`Show`** presents a detailed analysis of why SELinux denied access, and a possible solution for allowing access. If you are not running the X Window System, it is less obvious when access is denied by SELinux. For example, users browsing your website may receive an error similar to the following:

```
Forbidden

You don't have permission to access file name on this server
```

For these situations, if DAC rules (standard Linux permissions) allow access, check **/var/log/messages** and **/var/log/audit/audit.log** for **"SELinux is preventing"** and **"denied"** errors respectively. This can be done by running the following commands as the root user:

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

```
~]# grep "denied" /var/log/audit/audit.log
```

10.2. Top Three Causes of Problems

The following sections describe the top three causes of problems: labeling problems, configuring Booleans and ports for services, and evolving SELinux rules.

10.2.1. Labeling Problems

On systems running SELinux, all processes and files are labeled with a label that contains security-relevant information. This information is called the SELinux context. If these labels are wrong, access may be denied. If an application is labeled incorrectly, the process it transitions to may not have the correct label, possibly causing SELinux to deny access, and the process being able to create mislabeled files.

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 wants to use **/srv/myweb/**. On Red Hat Enterprise Linux, the **/srv/** directory is labeled with the **var_t** type. Files and directories created and **/srv/** inherit this type. Also, newly-created top-level directories (such as **/myserver/**) may 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 to **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 [11]. The **semanage** utility does not change the context. As root, run the **restorecon** utility to apply the changes:

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

Refer to Section 4.6.2, “Persistent Changes: semanage fcontext” for further information about adding contexts to the file-context configuration.

10.2.1.1. What is the Correct 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 using **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
```

Refer to Section 4.9.3, “Checking the Default SELinux Context” for a more detailed example of **matchpathcon**.

10.2.2. How are Confined Services Running?

Services can be run in a variety of ways. To cater for this, you need to specify how you run your services. This can be achieved via Booleans that allow parts of SELinux policy to be changed at runtime, without any knowledge of SELinux policy writing. This allows 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 via 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
```

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

For a list of Booleans and whether they are on or off, run the **getsebool -a** command. For a list of Booleans, an explanation of what each one is, and whether they are on or off, run the **semanage boolean -l** command as root. Refer to Section 4.5, “Booleans” for information about listing and configuring Booleans.

Port Numbers

Depending on policy configuration, services may 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, run the **semanage port** command to add a port to policy configuration [12]:

```
~]# 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.

10.2.3. Evolving Rules and Broken Applications

Applications may be broken, 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 has not seen before, 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. Refer to Section 10.3.8, “Allowing Access: audit2allow” for information about using **audit2allow**.

10.3. Fixing Problems

The following sections help troubleshoot issues. They go over: checking Linux permissions, which are checked before SELinux rules; possible causes of SELinux denying access, but no denials being logged; manual pages for services, which contain information about labeling and Booleans; permissive domains, for allowing one process to run permissive, rather than the whole system; how to search for and view denial messages; analyzing denials; and creating custom policy modules with **`audit2allow`**.

10.3.1. Linux Permissions

When access is denied, check standard Linux permissions. As mentioned in Chapter 1, *Introduction*, most operating systems use a Discretionary Access Control (DAC) system to control access, allowing users to control the permissions of files that they own. SELinux policy rules are checked after DAC rules. SELinux policy rules are not used if DAC rules deny access first.

If access is denied and no SELinux denials are logged, use the following command to view the standard Linux permissions:

```
~]$ ls -l /var/www/html/index.html
-rw-r----- 1 root root 0 2009-05-07 11:06 index.html
```

In this example, **`index.html`** is owned by the root user and group. The root user has read and write permissions (**`-rw`**), and members of the root group have read permissions (**`-r-`**). Everyone else has no access (**`---`**). By default, such permissions do not allow **`httpd`** to read this file. To resolve this issue, use the **`chown`** command to change the owner and group. This command must be run as root:

```
~]# chown apache:apache /var/www/html/index.html
```

This assumes the default configuration, in which **`httpd`** runs as the Linux Apache user. If you run **`httpd`** with a different user, replace **`apache:apache`** with that user.

Refer to the Fedora Documentation Project "Permissions" draft for information about managing Linux permissions.

10.3.2. Possible Causes of Silent Denials

In certain situations, AVC denial messages may not be logged when SELinux denies access. Applications and system library functions often probe for more access than required to perform their tasks. To maintain least privilege without filling audit logs with AVC denials for harmless application probing, the policy can silence AVC denials without allowing a permission by using **`dontaudit`** rules. These rules are common in standard policy. The downside of **`dontaudit`** is that, although SELinux denies access, denial messages are not logged, making troubleshooting more difficult.

To temporarily disable **`dontaudit`** rules, allowing all denials to be logged, run the following command as root:

```
~]# semodule -DB
```

The **`-D`** option disables **`dontaudit`** rules; the **`-B`** option rebuilds policy. After running **`semodule -DB`**, try exercising the application that was encountering permission problems, and see if SELinux denials — relevant to the application — are now being logged. Take care in deciding which denials should be allowed, as some should be ignored and handled via **`dontaudit`** rules. If in doubt, or in search of guidance, contact other SELinux users and developers on an SELinux list, such as fedora-selinux-list.

To rebuild policy and enable **dontaudit** rules, run the following command as root:

```
~]# semodule -B
```

This restores the policy to its original state. For a full list of **dontaudit** rules, run the **sesearch --dontaudit** command. Narrow down searches using the **-s *domain*** option and the **grep** command. For example:

```
~]$ sesearch --dontaudit -s smbd_t | grep squid
dontaudit smbd_t squid_port_t : tcp_socket name_bind ;
dontaudit smbd_t squid_port_t : udp_socket name_bind ;
```

Refer to Section 10.3.6, “Raw Audit Messages” and Section 10.3.7, “sealert Messages” for information about analyzing denials.

10.3.3. Manual Pages for Services

Manual pages for services contain valuable information, such as what file type to use for a given situation, and Booleans to change the access a service has (such as **httpd** accessing NFS volumes). This information may be in the standard manual page or in the manual page that can be automatically generated from SELinux policy for every service domain using the **sepolicy manpage** utility. Such manual pages are named in the ***service-name*_selinux** format.

For example, the httpd_selinux(8) manual page has information about what file type to use for a given situation, as well as Booleans to allow scripts, sharing files, accessing directories inside user home directories, and so on. Other manual pages with SELinux information for services include:

- Samba: the samba_selinux(8) manual page for example describes that enabling the **samba_enable_home_dirs** Boolean allows Samba to share users home directories.
- NFS: the nfsd_selinux(8) manual page describes SELinux nfsd policy that allows users to setup their nfsd processes in as secure a method as possible.
- Berkeley Internet Name Domain (BIND): the named(8) manual page describes what file type to use for a given situation (see the **Red Hat SELinux BIND Security Profile** section). The named_selinux(8) manual page describes that, by default, **named** cannot write to master zone files, and to allow such access, the **named_write_master_zones** Boolean must be enabled.

The information in manual pages helps you configure the correct file types and Booleans, helping to prevent SELinux from denying access.

See Section 5.4, “Generating Manual Pages: **sepolicy manpage**” for further information about **sepolicy manpage**.

10.3.4. Permissive Domains

When SELinux is running in permissive mode, SELinux does not deny access, but denials are logged for actions that would have been denied if running in enforcing mode. Previously, it was not possible to make a single domain permissive (remember: processes run in domains). In certain situations, this led to making the whole system permissive to troubleshoot issues.

Permissive domains allow an administrator to configure a single process (domain) to run permissive, rather than making the whole system permissive. SELinux checks are still performed for permissive domains; however, the kernel allows access and reports an AVC denial for situations where SELinux would have denied access.

Permissive domains have the following uses:

- They can be used for making a single process (domain) run permissive to troubleshoot an issue without putting the entire system at risk by making it permissive.
- They allow an administrator to create policies for new applications. Previously, it was recommended that a minimal policy be created, and then the entire machine put into permissive mode, so that the application could run, but SELinux denials still logged. The **audit2allow** could then be used to help write the policy. This put the whole system at risk. With permissive domains, only the domain in the new policy can be marked permissive, without putting the whole system at risk.

10.3.4.1. Making a Domain Permissive

To make a domain permissive, run the **semanage permissive -a *domain*** command, where *domain* is the domain you want to make permissive. For example, run the following command as root to make the **httpd_t** domain (the domain the Apache HTTP Server runs in) permissive:

```
~]# semanage permissive -a httpd_t
```

To view a list of domains you have made permissive, run the **semodule -l | grep permissive** command as root. For example:

```
~]# semodule -l | grep permissive
permissive_httpd_t 1.0
permissivedomains 1.0.0
```

If you no longer want a domain to be permissive, run the **semanage permissive -d *domain*** command as root. For example:

```
~]# semanage permissive -d httpd_t
```

10.3.4.2. Disabling Permissive Domains

The **permissivedomains.pp** module contains all of the permissive domain declarations that are presented on the system. To disable all permissive domains, run the following command as root:

```
~]# semodule -d permissivedomains
```

10.3.4.3. Denials for Permissive Domains

The **SYSCALL** message is different for permissive domains. The following is an example AVC denial (and the associated system call) from the Apache HTTP Server:

```
type=AVC msg=audit(1226882736.442:86): avc:  denied  { getattr } for  pid=2427
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

type=SYSCALL msg=audit(1226882736.442:86): arch=40000003 syscall=196 success=no
```

```
exit=-13 a0=b9a1e198 a1=bfc2921c a2=54dff4 a3=2008171 items=0 ppid=2425 pid=2427
auid=502 uid=48 gid=48 euid=48 suid=48 fsuid=48 egid=48 sgid=48 fsgid=48 tty=(none)
ses=4 comm="httpd" exe="/usr/sbin/httpd" subj=unconfined_u:system_r:httpd_t:s0 key=
(null)
```

By default, the **httpd_t** domain is not permissive, and as such, the action is denied, and the **SYSCALL** message contains **success=no**. The following is an example AVC denial for the same situation, except the **semanage permissive -a httpd_t** command has been run to make the **httpd_t** domain permissive:

```
type=AVC msg=audit(1226882925.714:136): avc:  denied  { read } for  pid=2512
comm="httpd" name="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

type=SYSCALL msg=audit(1226882925.714:136): arch=40000003 syscall=5 success=yes
exit=11 a0=b962a1e8 a1=8000 a2=0 a3=8000 items=0 ppid=2511 pid=2512 auid=502 uid=48
gid=48 euid=48 suid=48 fsuid=48 egid=48 sgid=48 fsgid=48 tty=(none) ses=4
comm="httpd" exe="/usr/sbin/httpd" subj=unconfined_u:system_r:httpd_t:s0 key=(null)
```

In this case, although an AVC denial was logged, access was not denied, as shown by **success=yes** in the **SYSCALL** message.

Refer to Dan Walsh's "Permissive Domains" blog entry for further information about permissive domains.

10.3.5. Searching For and Viewing Denials

This section assumes the *setroubleshoot*, *setroubleshoot-server*, *dbus* and *audit* packages are installed, and that the **auditd**, **rsyslogd**, and **setroubleshootd** daemons are running. Refer to Section 4.2, "Which Log File is Used" for information about starting these daemons. A number of utilites are available for searching for and viewing SELinux AVC messages, such as **ausearch**, **aureport**, and **sealert**.

ausearch

The *audit* package provides the **ausearch** utility that can query the **audit** daemon logs based for events based on different search criteria. [13] The **ausearch** utility accesses **/var/log/audit/audit.log**, and as such, must be run as the root user:

Searching For	Command
all denials	**ausearch -m avc**
denials for that today	**ausearch -m avc -ts today**
denials from the last 10 minutes	**ausearch -m avc -ts recent**

To search for SELinux AVC messages for a particular service, use the **-c *comm-name*** option, where *comm-name* is the executable's name, for example, **httpd** for the Apache HTTP Server, and **smbd** for Samba:

```
~]# ausearch -m avc -c httpd
```

```
~]# ausearch -m avc -c smbd
```

With each **ausearch** command, it is advised to use either the **--interpret** (**-i**) option for easier readability, or the **--raw** (**-r**) option for script processing. Refer to the ausearch(8) manual page for further **ausearch** options.

aureport

The *audit* package provides the **aureport** utility, which produces summary reports of the audit system logs. [14] The **aureport** utility accesses **/var/log/audit/audit.log**, and as such, must be run as the root user. To view a list of SELinux denial messages and how often each one occurred, run the **aureport -a** command. The following is example output that includes two denials:

```
~]# aureport -a

AVC Report
========================================================
# date time comm subj syscall class permission obj event
========================================================
1. 05/01/2009 21:41:39 httpd unconfined_u:system_r:httpd_t:s0 195 file getattr
system_u:object_r:samba_share_t:s0 denied 2
2. 05/03/2009 22:00:25 vsftpd unconfined_u:system_r:ftpd_t:s0 5 file read
unconfined_u:object_r:cifs_t:s0 denied 4
```

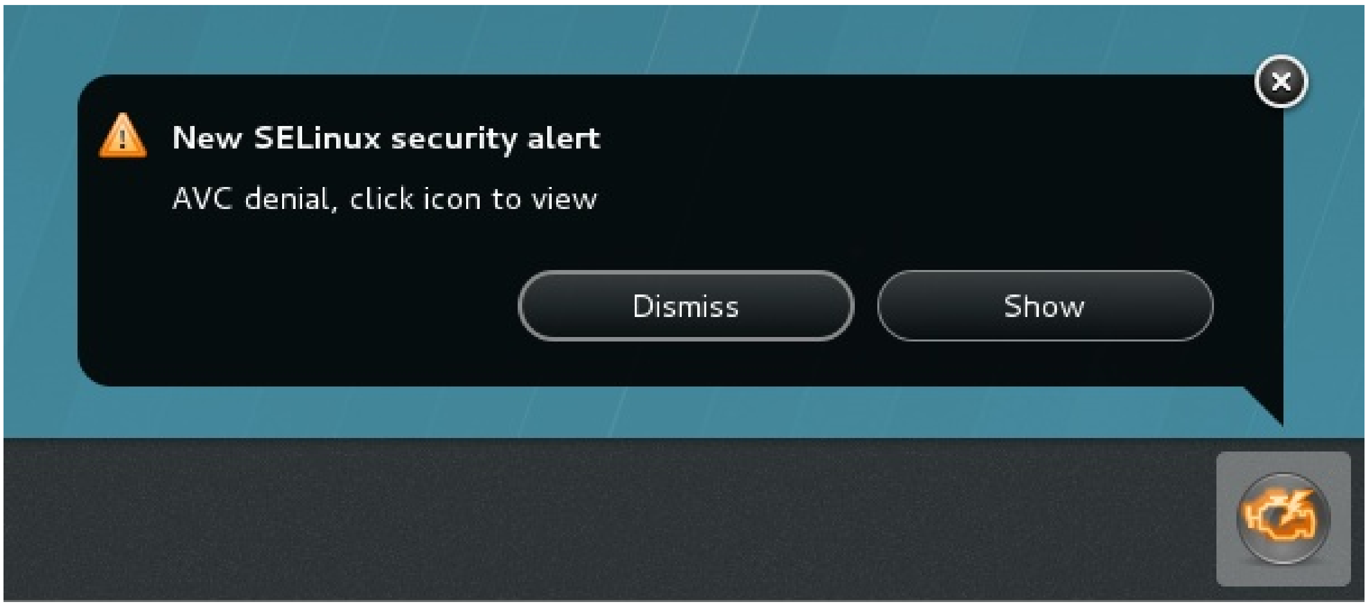
sealert

The *setroubleshoot-server* package provides the **sealert** utility, which reads denial messages translated by *setroubleshoot-server*. [15] Denials are assigned IDs, as seen in **/var/log/messages**. The following is an example denial from **messages**:

```
setroubleshoot: SELinux is preventing /usr/sbin/httpd from name_bind access on the
tcp_socket. For complete SELinux messages. run sealert -l 8c123656-5dda-4e5d-8791-
9e3bd03786b7
```

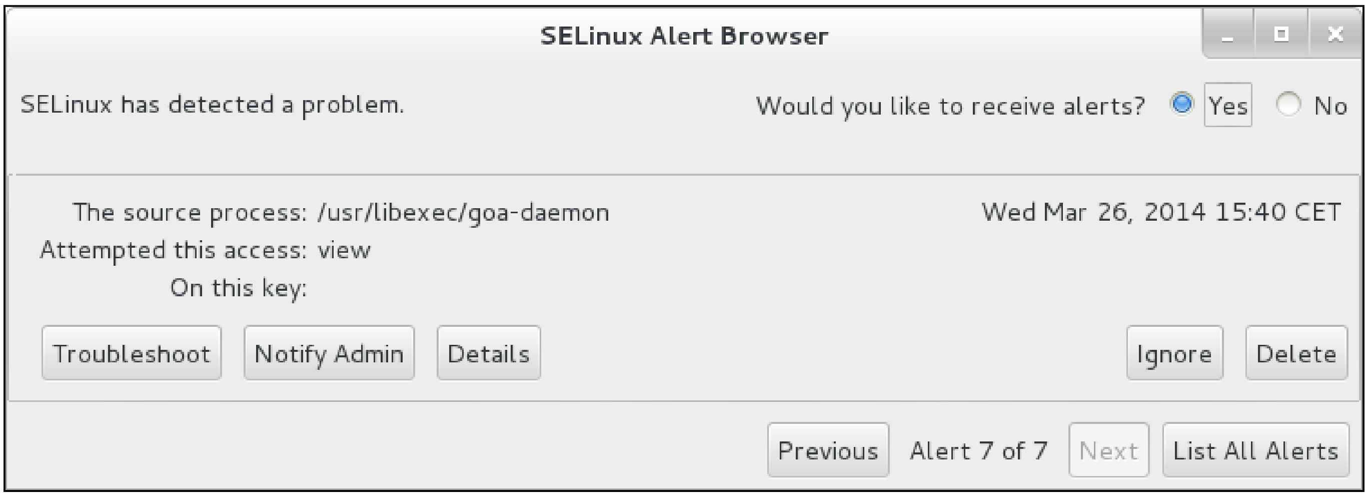
In this example, the denial ID is **8c123656-5dda-4e5d-8791-9e3bd03786b7**. The **-l** option takes an ID as an argument. Running the **sealert -l 8c123656-5dda-4e5d-8791-9e3bd03786b7** command presents a detailed analysis of why SELinux denied access, and a possible solution for allowing access.

If you are running the X Window System, have the *setroubleshoot* and *setroubleshoot-server* packages installed, and the **setroubleshootd**, **dbus** and **auditd** daemons are running, a warning is displayed when access is denied by SELinux:


New SELinux security alert

AVC denial, click icon to view

Clicking on **Show** launches the **sealert** GUI, which allows you to troubleshoot the problem:


SELinux Alert Browser

SELinux has detected a problem.

Would you like to receive alerts? [✓ Yes] No

The source process: /usr/libexec/goa-daemon

Attempted this access: view

On this key:

Wed Mar 26, 2014 15:40 CET

Alternatively, run the **sealert -b** command to launch the **sealert** GUI. To view a detailed analysis of all denial messages, run the **sealert -l *** command.

10.3.6. Raw Audit Messages

Raw audit messages are logged to **/var/log/audit/audit.log**. The following is an example AVC denial message (and the associated system call) that occurred when the Apache HTTP Server (running in the **httpd_t** domain) attempted to access the **/var/www/html/file1** file (labeled with the **samba_share_t** type):

```
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

type=SYSCALL msg=audit(1226874073.147:96): arch=40000003 syscall=196 success=no
exit=-13 a0=b98df198 a1=bfec85dc a2=54dff4 a3=2008171 items=0 ppid=2463 pid=2465
auid=502 uid=48 gid=48 euid=48 suid=48 fsuid=48 egid=48 sgid=48 fsgid=48 tty=(none)
ses=6 comm="httpd" exe="/usr/sbin/httpd" subj=unconfined_u:system_r:httpd_t:s0 key=
(null)
```

{ getattr }

The item in the curly brackets indicates the permission that was denied. The **`getattr`** entry indicates the source process was trying to read the target file's status information. This occurs before reading files. This action is denied due to the file being accessed having a wrong label. Commonly seen permissions include **`getattr`**, **`read`**, and **`write`**.

comm="*httpd*"

The executable that launched the process. The full path of the executable is found in the **`exe=`** section of the system call (**`SYSCALL`**) message, which in this case, is **`exe="/usr/sbin/httpd"`**.

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 that attempted the denied action. In this case, it is the SELinux context of the Apache HTTP Server, which is running in the **`httpd_t`** domain.

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`**. Note that the **`samba_share_t`** type is not accessible to processes running in the **`httpd_t`** domain.

In certain situations, the **`tcontext`** may match the **`scontext`**, for example, when a process attempts to execute a system service that will change characteristics of that running process, such as the user ID. Also, the **`tcontext`** may match the **`scontext`** when a process tries to use more resources (such as memory) than normal limits allow, resulting in a security check to see if that process is allowed to break those limits.

From the system call (**`SYSCALL`**) message, two items are of interest:

- **`success=`*`no`***: indicates whether the denial (AVC) was enforced or not. **`success=no`** indicates the system call was not successful (SELinux denied access). **`success=yes`** indicates the system call was successful. This can be seen for permissive domains or unconfined domains, such as **`initrc_t`** and **`kernel_t`**.
- **`exe="`*`/usr/sbin/httpd`*`"`**: the full path to the executable that launched the process, which in this case, is **`exe="/usr/sbin/httpd"`**.

An incorrect file type is a common cause for SELinux denying access. To start troubleshooting, compare the source context (**`scontext`**) with the target context (**`tcontext`**). Should the process (**`scontext`**) be accessing such an object (**`tcontext`**)? For example, the Apache HTTP Server (**`httpd_t`**) should only be accessing types specified in the httpd_selinux(8) manual page, such as **`httpd_sys_content_t`**, **`public_content_t`**, and so on, unless configured otherwise.

10.3.7. sealert Messages

Denials are assigned IDs, as seen in **`/var/log/messages`**. The following is an example AVC denial (logged to **`messages`**) that occurred when the Apache HTTP Server (running in the **`httpd_t`** domain) attempted to access the **`/var/www/html/file1`** file (labeled with the **`samba_share_t`** type):

```
hostname setroubleshoot: SELinux is preventing httpd (httpd_t) "getattr" to
/var/www/html/file1 (samba_share_t). For complete SELinux messages. run sealert -l
84e0b04d-d0ad-4347-8317-22e74f6cd020
```

As suggested, run the **`sealert -l 84e0b04d-d0ad-4347-8317-22e74f6cd020`** command to view the complete message. This command only works on the local machine, and presents the same information as the **`sealert`** GUI:

```
~]$ sealert -l 84e0b04d-d0ad-4347-8317-22e74f6cd020

Summary:

SELinux is preventing httpd (httpd_t) "getattr" to /var/www/html/file1
(samba_share_t).

Detailed Description:

SELinux denied access to /var/www/html/file1 requested by httpd.
/var/www/html/file1 has a context used for sharing by different program. If you
would like to share /var/www/html/file1 from httpd also, you need to change its
file context to public_content_t. If you did not intend to this access, this
could signal a intrusion attempt.

Allowing Access:

You can alter the file context by executing chcon -t public_content_t
'/var/www/html/file1'

Fix Command:

chcon -t public_content_t '/var/www/html/file1'

Additional Information:

Source Context                unconfined_u:system_r:httpd_t:s0
Target Context                unconfined_u:object_r:samba_share_t:s0
Target Objects                /var/www/html/file1 [ file ]
Source                        httpd
Source Path                   /usr/sbin/httpd
Port                          <Unknown>
Host                          hostname
Source RPM Packages           httpd-2.2.10-2
Target RPM Packages
Policy RPM                    selinux-policy-3.5.13-11.fc12
Selinux Enabled               True
Policy Type                   targeted
MLS Enabled                   True
Enforcing Mode                Enforcing
Plugin Name                   public_content
Host Name                     hostname
Platform                      Linux hostname 2.6.27.4-68.fc12.i686 #1 SMP Thu Oct
30 00:49:42 EDT 2008 i686 i686
Alert Count                   4
First Seen                    Wed Nov  5 18:53:05 2008
Last Seen                     Wed Nov  5 01:22:58 2008
Local ID                      84e0b04d-d0ad-4347-8317-22e74f6cd020
Line Numbers

Raw Audit Messages

node=hostname type=AVC msg=audit(1225812178.788:101): avc:  denied  { getattr } for
pid=2441 comm="httpd" path="/var/www/html/file1" dev=dm-0 ino=284916
scontext=unconfined_u:system_r:httpd_t:s0
tcontext=unconfined_u:object_r:samba_share_t:s0 tclass=file
```

```
node=hostname type=SYSCALL msg=audit(1225812178.788:101): arch=40000003 syscall=196
success=no exit=-13 a0=b8e97188 a1=bf87aaac a2=54dff4 a3=2008171 items=0 ppid=2439
pid=2441 auid=502 uid=48 gid=48 euid=48 suid=48 fsuid=48 egid=48 sgid=48 fsgid=48
tty=(none) ses=3 comm="httpd" exe="/usr/sbin/httpd"
subj=unconfined_u:system_r:httpd_t:s0 key=(null)
```

Summary

A brief summary of the denied action. This is the same as the denial in **`/var/log/messages`**. In this example, the **`httpd`** process was denied access to a file (**`file1`**), which is labeled with the **`samba_share_t`** type.

Detailed Description

A more verbose description. In this example, **`file1`** is labeled with the **`samba_share_t`** type. This type is used for files and directories that you want to export via Samba. The description suggests changing the type to a type that can be accessed by the Apache HTTP Server and Samba, if such access is desired.

Allowing Access

A suggestion for how to allow access. This may be relabeling files, enabling a Boolean, or making a local policy module. In this case, the suggestion is to label the file with a type accessible to both the Apache HTTP Server and Samba.

Fix Command

A suggested command to allow access and resolve the denial. In this example, it gives the command to change the **`file1`** type to **`public_content_t`**, which is accessible to the Apache HTTP Server and Samba.

Additional Information

Information that is useful in bug reports, such as the policy package name and version (**`selinux-policy-3.5.13-11.fc12`**), but may not help towards solving why the denial occurred.

Raw Audit Messages

The raw audit messages from **`/var/log/audit/audit.log`** that are associated with the denial. Refer to Section 10.3.6, “Raw Audit Messages” for information about each item in the AVC denial.

10.3.8. Allowing Access: audit2allow

Warning

Do not use the example in this section in production. It is used only to demonstrate the use of the **`audit2allow`** utility.

The **`audit2allow`** utility gathers information from logs of denied operations and then generates SELinux policy allow rules. [16] After analyzing denial messages as per Section 10.3.7, “sealert Messages”, and if no label changes or Booleans allowed access, use **`audit2allow`** to create a local policy module. When access is denied by SELinux, running **`audit2allow`** generates Type Enforcement rules that allow the previously denied access.

The following example demonstrates using **audit2allow** to create a policy module:

1. A denial message and the associated system call are logged to the **/var/log/audit/audit.log** file:

```
type=AVC msg=audit(1226270358.848:238): avc:  denied  { write } for  pid=13349
comm="certwatch" name="cache" dev=dm-0 ino=218171
scontext=system_u:system_r:certwatch_t:s0 tcontext=system_u:object_r:var_t:s0
tclass=dir

type=SYSCALL msg=audit(1226270358.848:238): arch=40000003 syscall=39
success=no exit=-13 a0=39a2bf a1=3ff a2=3a0354 a3=94703c8 items=0 ppid=13344
pid=13349 auid=4294967295 uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0
fsgid=0 tty=(none) ses=4294967295 comm="certwatch" exe="/usr/bin/certwatch"
subj=system_u:system_r:certwatch_t:s0 key=(null)
```

   In this example, **certwatch** was denied the write access to a directory labeled with the **var_t** type. Analyze the denial message as per Section 10.3.7, “sealert Messages”. If no label changes or Booleans allowed access, use **audit2allow** to create a local policy module.

2. Run the following command to produce a human-readable description of why the access was denied. The **audit2allow** utility reads **/var/log/audit/audit.log**, and as such, must be run as the root user:

```
 ~]# audit2allow -w -a
type=AVC msg=audit(1226270358.848:238): avc:  denied  { write } for  pid=13349
comm="certwatch" name="cache" dev=dm-0 ino=218171
scontext=system_u:system_r:certwatch_t:s0 tcontext=system_u:object_r:var_t:s0
tclass=dir
 Was caused by:
  Missing type enforcement (TE) allow rule.

 You can use audit2allow to generate a loadable module to allow this access.
```

   The **-a** command-line option causes all audit logs to be read. The **-w** option produces the human-readable description. As shown, access was denied due to a missing Type Enforcement rule.

3. Run the following command to view the Type Enforcement rule that allows the denied access:

```
 ~]# audit2allow -a


#============= certwatch_t ==============
allow certwatch_t var_t:dir write;
```

Important

Missing Type Enforcement rules are usually caused by bugs in the SELinux policy, and should be reported in Red Hat Bugzilla. For Red Hat Enterprise Linux, create bugs against the **Red Hat Enterprise Linux** product, and select the **selinux-policy** component. Include the output of the **audit2allow -w -a** and **audit2allow -a** commands in such bug reports.

4. To use the rule displayed by **audit2allow -a**, run the following command as root to create a custom module. The **-M** option creates a Type Enforcement file (**.te**) with the name specified with **-M**, in your current working directory:

```
~]# audit2allow -a -M mycertwatch
******************** IMPORTANT ***********************
To make this policy package active, execute:

semodule -i mycertwatch.pp
```

5. Also, **audit2allow** compiles the Type Enforcement rule into a policy package (**.pp**):

```
~]# ls
mycertwatch.pp  mycertwatch.te
```

To install the module, run the following command as the root:

```
~]# semodule -i mycertwatch.pp
```

Important

Modules created with **audit2allow** may allow more access than required. It is recommended that policy created with **audit2allow** be posted to the upstream SELinux list for review. If you believe there is a bug in the policy, please create a bug in Red Hat Bugzilla.

If you have multiple denial messages from multiple processes, but only want to create a custom policy for a single process, use the **grep** utility to narrow down the input for **audit2allow**. The following example demonstrates using **grep** to only send denial messages related to **certwatch** through **audit2allow**:

```
~]# grep certwatch /var/log/audit/audit.log | audit2allow -R -M mycertwatch2
******************** IMPORTANT ***********************
To make this policy package active, execute:

semodule -i mycertwatch2.pp
```

[11] Files in **/etc/selinux/targeted/contexts/files/** define contexts for files and directories. Files in this directory are read by the **restorecon** and **setfiles** utilities to restore files and directories to their default contexts.

[12] The **semanage port -a** command adds an entry to the **/etc/selinux/targeted/modules/active/ports.local** file. Note that by default, this file can only be viewed by root.

[13] Refer to the ausearch(8) manual page for further information about **ausearch**.

[14] Refer to the aureport(8) manual page for further information about **aureport**.

[15] See the sealert(8) manual page for further information about **sealert**.

[16] Refer to the audit2allow(1) manual page for more information about **audit2allow**.

Chapter 11. Further Information

11.1. Contributors

- Dominick Grift – Technical Editor
- Murray McAllister – Red Hat Engineering Content Services
- James Morris – Technical Editor
- Eric Paris – Technical Editor
- Scott Radvan – Red Hat Engineering Content Services
- Daniel Walsh – Red Hat Security Engineering

11.2. Other Resources

The National Security Agency (NSA)

NSA was the original developer of SELinux. Researchers in NSA's National Information Assurance Research Laboratory (NIARL) designed and implemented flexible mandatory access controls in the major subsystems of the Linux kernel and implemented the new operating system components provided by the Flask architecture, namely the security server and the access vector cache. [17]

- Main SELinux website: http://www.nsa.gov/research/selinux/index.shtml.
- SELinux documentation: http://www.nsa.gov/research/selinux/docs.shtml.
- SELinux background: http://www.nsa.gov/research/selinux/background.shtml.

Tresys Technology

Tresys Technology are the upstream for:

- SELinux userland libraries and tools.
- SELinux Reference Policy.

SELinux News

- News: http://selinuxnews.org/wp/.
- Planet SELinux (blogs): http://selinuxnews.org/planet/.

SELinux Project Wiki

- Main page: http://selinuxproject.org/page/Main_Page.
- User resources, including links to documentation, mailing lists, websites, and tools: http://selinuxproject.org/page/User_Resources.

Fedora

- Main page: http://fedoraproject.org/wiki/SELinux.
- Troubleshooting: http://fedoraproject.org/wiki/SELinux/Troubleshooting.
- Fedora SELinux FAQ: http://docs.fedoraproject.org/.
- SELinux Managing Confined Services Guide: http://docs.fedoraproject.org/

The UnOfficial SELinux FAQ

http://www.crypt.gen.nz/selinux/faq.html

The SELinux Notebook - The Foundations - 3rd Edition

http://www.freetechbooks.com/the-selinux-notebook-the-foundations-t785.html

IRC

On Freenode:

- #selinux
- #fedora-selinux
- #security

[17] Refer to the NSA Contributors to SELinux page for more information.

Part II. Managing Confined Services

Chapter 12. Introduction

This part of the book focuses more on practical tasks and provides information how to set up and configure various services. For each service, there are listed the most common types and Booleans with the specifications. Also included are real-world examples of configuring those services and demonstrations of how SELinux complements their operation.

When SELinux is in enforcing mode, the default policy used in Red Hat Enterprise Linux, is the targeted policy. Processes that are targeted run in a confined domain, and processes that are not targeted run in an unconfined domain. Refer to Chapter 3, *Targeted Policy* for more information about targeted policy and confined and unconfined processes.

Chapter 13. The Apache HTTP Server

The Apache HTTP Server provides an open-source HTTP server with the current HTTP standards. [18]

In Red Hat Enterprise Linux, the *httpd* package provides the Apache HTTP Server. Run the following command to see if the *httpd* package is installed:

```
~]$ rpm -q httpd
package httpd is not installed
```

If it is not installed and you want to use the Apache HTTP Server, use the **yum** utility as the root user to install it:

```
~]# yum install httpd
```

13.1. The Apache HTTP Server and SELinux

When SELinux is enabled, the Apache HTTP Server (**httpd**) runs confined by default. Confined processes run in their own domains, and are separated from other confined processes. If a confined process is compromised by an attacker, depending on SELinux policy configuration, an attacker's access to resources and the possible damage they can do is limited. The following example demonstrates the **httpd** processes running in their own domain. This example assumes the *httpd*, *setroubleshoot*, *setroubleshoot-server* and *policycoreutils-python* packages are installed:

1. Run the **getenforce** command to confirm SELinux is running in enforcing mode:

   ```
   ~]$ getenforce
   Enforcing
   ```

   The command returns **Enforcing** when SELinux is running in enforcing mode.

2. Run the following command as root to start **httpd**:

   ```
   ~]# systemctl start httpd.service
   ```

   Confirm that the service is running. The output should include the information below (only the time stamp will differ):

   ```
   ~]# systemctl status httpd.service
   httpd.service - The Apache HTTP Server
      Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled)
      Active: active (running) since Mon 2013-08-05 14:00:55 CEST; 8s ago
   ```

3. To view the **httpd** processes, execute the following command:

   ```
   ~]$ ps -eZ | grep httpd
   system_u:system_r:httpd_t:s0    19780 ?        00:00:00 httpd
   system_u:system_r:httpd_t:s0    19781 ?        00:00:00 httpd
   system_u:system_r:httpd_t:s0    19782 ?        00:00:00 httpd
   system_u:system_r:httpd_t:s0    19783 ?        00:00:00 httpd
   system_u:system_r:httpd_t:s0    19784 ?        00:00:00 httpd
   system_u:system_r:httpd_t:s0    19785 ?        00:00:00 httpd
   ```

The SELinux context associated with the **httpd** processes is **system_u:system_r:httpd_t:s0**. The second last part of the context, **httpd_t**, is the type. A type defines a domain for processes and a type for files. In this case, the **httpd** processes are running in the **httpd_t** domain.

SELinux policy defines how processes running in confined domains (such as **httpd_t**) interact with files, other processes, and the system in general. Files must be labeled correctly to allow **httpd** access to them. For example, **httpd** can read files labeled with the **httpd_sys_content_t** type, but cannot write to them, even if Linux (DAC) permissions allow write access. Booleans must be enabled to allow certain behavior, such as allowing scripts network access, allowing **httpd** access to NFS and CIFS volumes, and **httpd** being allowed to execute Common Gateway Interface (CGI) scripts.

When the **/etc/httpd/conf/httpd.conf** file is configured so **httpd** listens on a port other than TCP ports 80, 443, 488, 8008, 8009, or 8443, the **semanage port** command must be used to add the new port number to SELinux policy configuration. The following example demonstrates configuring **httpd** to listen on a port that is not already defined in SELinux policy configuration for **httpd**, and, as a consequence, **httpd** failing to start. This example also demonstrates how to then configure the SELinux system to allow **httpd** to successfully listen on a non-standard port that is not already defined in the policy. This example assumes the *httpd* package is installed. Run each command in the example as the root user:

1. Run the following command to confirm **httpd** is not running:

   ```
   ~]# systemctl status httpd.service
   httpd.service - The Apache HTTP Server
      Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled)
             Active: inactive (dead)
   ```

   If the output differs, stop the process:

   ```
   ~]# systemctl stop httpd.service
   ```

2. Use the **semanage** utility to view the ports SELinux allows **httpd** to listen on:

   ```
   ~]# semanage port -l | grep -w http_port_t
   http_port_t                    tcp      80, 443, 488, 8008, 8009, 8443
   ```

3. Edit the **/etc/httpd/conf/httpd.conf** file as root. Configure the **Listen** option so it lists a port that is not configured in SELinux policy configuration for **httpd**. In this example, **httpd** is configured to listen on port 12345:

   ```
   # Change this to Listen on specific IP addresses as shown below to
   # prevent Apache from glomming onto all bound IP addresses (0.0.0.0)
   #
   #Listen 12.34.56.78:80
   Listen 127.0.0.1:12345
   ```

4. Run the following command to start **httpd**:

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

   An SELinux denial message similar to the following is logged:

```
setroubleshoot: SELinux is preventing the httpd (httpd_t) from binding to port
12345. For complete SELinux messages. run sealert -l f18bca99-db64-4c16-9719-
1db89f0d8c77
```

5. For SELinux to allow **httpd** to listen on port 12345, as used in this example, the following command is required:

```
~]# semanage port -a -t http_port_t -p tcp 12345
```

6. Start **httpd** again and have it listen on the new port:

```
~]# systemctl start httpd.service
```

7. Now that SELinux has been configured to allow **httpd** to listen on a non-standard port (TCP 12345 in this example), **httpd** starts successfully on this port.

8. To prove that **httpd** is listening and communicating on TCP port 12345, open a telnet connection to the specified port and issue a HTTP GET command, as follows:

```
~]# telnet localhost 12345
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET / HTTP/1.0

HTTP/1.1 200 OK
Date: Wed, 02 Dec 2009 14:36:34 GMT
Server: Apache/2.2.13 (Red Hat)
Accept-Ranges: bytes
Content-Length: 3985
Content-Type: text/html; charset=UTF-8
[...continues...]
```

13.2. Types

Type Enforcement is the main permission control used in SELinux targeted policy. All files and processes are labeled with a type: types define a domain for processes and a type for files. SELinux policy rules define how types access each other, whether it be a domain accessing a type, or a domain accessing another domain. Access is only allowed if a specific SELinux policy rule exists that allows it.

The following example creates a new file in the **/var/www/html/** directory, and shows the file inheriting the **httpd_sys_content_t** type from its parent directory (**/var/www/html/**):

1. Run the following command to view the SELinux context of **/var/www/html/**:

```
~]$ ls -dZ /var/www/html
drwxr-xr-x  root root system_u:object_r:httpd_sys_content_t:s0 /var/www/html
```

   This shows **/var/www/html/** is labeled with the **httpd_sys_content_t** type.

2. Create a new file by using the **touch** utility as root:

```
~]# touch /var/www/html/file1
```

3. Run the following command to view the SELinux context:

```
~]$ ls -Z /var/www/html/file1
-rw-r--r--  root root unconfined_u:object_r:httpd_sys_content_t:s0
/var/www/html/file1
```

The **`ls -Z`** command shows **`file1`** labeled with the **`httpd_sys_content_t`** type. SELinux allows **`httpd`** to read files labeled with this type, but not write to them, even if Linux permissions allow write access. SELinux policy defines what types a process running in the **`httpd_t`** domain (where **`httpd`** runs) can read and write to. This helps prevent processes from accessing files intended for use by another process.

For example, **`httpd`** can access files labeled with the **`httpd_sys_content_t`** type (intended for the Apache HTTP Server), but by default, cannot access files labeled with the **`samba_share_t`** type (intended for Samba). Also, files in user home directories are labeled with the **`user_home_t`** type: by default, this prevents **`httpd`** from reading or writing to files in user home directories.

The following lists some of the types used with **`httpd`**. Different types allow you to configure flexible access:

`httpd_sys_content_t`

Use this type for static web content, such as **`.html`** files used by a static website. Files labeled with this type are accessible (read only) to **`httpd`** and scripts executed by **`httpd`**. By default, files and directories labeled with this type cannot be written to or modified by **`httpd`** or other processes. Note that by default, files created in or copied into the **`/var/www/html/`** directory are labeled with the **`httpd_sys_content_t`** type.

`httpd_sys_script_exec_t`

Use this type for scripts you want **`httpd`** to execute. This type is commonly used for Common Gateway Interface (CGI) scripts in the **`/var/www/cgi-bin/`** directory. By default, SELinux policy prevents **`httpd`** from executing CGI scripts. To allow this, label the scripts with the **`httpd_sys_script_exec_t`** type and enable the **`httpd_enable_cgi`** Boolean. Scripts labeled with **`httpd_sys_script_exec_t`** run in the **`httpd_sys_script_t`** domain when executed by **`httpd`**. The **`httpd_sys_script_t`** domain has access to other system domains, such as **`postgresql_t`** and **`mysqld_t`**.

`httpd_sys_rw_content_t`

Files labeled with this type can be written to by scripts labeled with the **`httpd_sys_script_exec_t`** type, but cannot be modified by scripts labeled with any other type. You must use the **`httpd_sys_rw_content_t`** type to label files that will be read from and written to by scripts labeled with the **`httpd_sys_script_exec_t`** type.

`httpd_sys_ra_content_t`

Files labeled with this type can be appended to by scripts labeled with the **`httpd_sys_script_exec_t`** type, but cannot be modified by scripts labeled with any other type. You must use the **`httpd_sys_ra_content_t`** type to label files that will be read from and appended to by scripts labeled with the **`httpd_sys_script_exec_t`** type.

`httpd_unconfined_script_exec_t`

Scripts labeled with this type run without SELinux protection. Only use this type for complex scripts, after exhausting all other options. It is better to use this type instead of disabling SELinux protection for **`httpd`**, or for the entire system.

Note

To see more of the available types for httpd, run the following command:

```
~]$ grep httpd /etc/selinux/targeted/contexts/files/file_contexts
```

Procedure 13.1. Changing the SELinux Context

The type for files and directories can be changed with the **chcon** command. Changes made with **chcon** do not survive a file system relabel or the **restorecon** command. SELinux policy controls whether users are able to modify the SELinux context for any given file. The following example demonstrates creating a new directory and an **index.html** file for use by **httpd**, and labeling that file and directory to allow **httpd** access to them:

1. Use the **mkdir** utility as root to create a top-level directory structure to store files to be used by **httpd**:

   ```
   ~]# mkdir -p /my/website
   ```

2. Files and directories that do not match a pattern in file-context configuration may be labeled with the **default_t** type. This type is inaccessible to confined services:

   ```
   ~]$ ls -dZ /my
   drwxr-xr-x  root root unconfined_u:object_r:default_t:s0 /my
   ```

3. Run the following command as root to change the type of the **/my/** directory and subdirectories, to a type accessible to **httpd**. Now, files created under **/my/website/** inherit the **httpd_sys_content_t** type, rather than the **default_t** type, and are therefore accessible to httpd:

   ```
   ~]# chcon -R -t httpd_sys_content_t /my/
   ~]# touch /my/website/index.html
   ~]# ls -Z /my/website/index.html
   -rw-r--r--  root root unconfined_u:object_r:httpd_sys_content_t:s0
   /my/website/index.html
   ```

Refer to Section 4.6.1, “Temporary Changes: chcon” for further information about **chcon**.

Use the **semanage fcontext** command (**semanage** is provided by the *policycoreutils-python* package) to make label changes that survive a relabel and the **restorecon** command. This command adds changes to file-context configuration. Then, run **restorecon**, which reads file-context configuration, to apply the label change. The following example demonstrates creating a new directory and an **index.html** file for use by **httpd**, and persistently changing the label of that directory and file to allow **httpd** access to them:

1. Use the **mkdir** utility as root to create a top-level directory structure to store files to be used by **httpd**:

   ```
   ~]# mkdir -p /my/website
   ```

2. Run the following command as root to add the label change to file-context configuration:

   ```
   ~]# semanage fcontext -a -t httpd_sys_content_t "/my(/.*)?"
   ```

The **"/my(/.*)?"** expression means the label change applies to the **/my/** directory and all files and directories under it.

3. Use the **touch** utility as root to create a new file:

   ```
   ~]# touch /my/website/index.html
   ```

4. Run the following command as root to apply the label changes (**restorecon** reads file-context configuration, which was modified by the **semanage** command in step 2):

   ```
   ~]# restorecon -R -v /my/
   restorecon reset /my context unconfined_u:object_r:default_t:s0-
   >system_u:object_r:httpd_sys_content_t:s0
   restorecon reset /my/website context unconfined_u:object_r:default_t:s0-
   >system_u:object_r:httpd_sys_content_t:s0
   restorecon reset /my/website/index.html context
   unconfined_u:object_r:default_t:s0->system_u:object_r:httpd_sys_content_t:s0
   ```

Refer to Section 4.6.2, “Persistent Changes: semanage fcontext” for further information on semanage.

13.3. Booleans

SELinux is based on the least level of access required for a service to run. Services can be run in a variety of ways; therefore, you need to specify how you run your services. This can be achieved using Booleans that allow parts of SELinux policy to be changed at runtime, without any knowledge of SELinux policy writing. This allows changes, such as allowing services access to NFS volumes, without reloading or recompiling SELinux policy.

To modify the state of a Boolean, use the **setsebool** command. For example, to enable the **httpd_anon_write** Boolean, run the following command as the root user:

```
~]# setsebool -P httpd_anon_write on
```

To disable a Boolean, using the same example, simply change **on** to **off** in the command, as shown below:

```
~]# setsebool -P httpd_anon_write off
```

Note

Do not use the **-P** option if you do not want **setsebool** changes to persist across reboots.

Below is a description of common Booleans available that cater for the way **httpd** is running:

httpd_anon_write

When disabled, this Boolean allows **httpd** to only have read access to files labeled with the **public_content_rw_t** type. Enabling this Boolean allows **httpd** to write to files labeled with the **public_content_rw_t** type, such as a public directory containing files for a public file transfer service.

httpd_mod_auth_ntlm_winbind

Enabling this Boolean allows access to NTLM and Winbind authentication mechanisms via the **mod_auth_ntlm_winbind** module in **httpd**.

httpd_mod_auth_pam

Enabling this Boolean allows access to PAM authentication mechanisms via the **mod_auth_pam** module in **httpd**.

httpd_sys_script_anon_write

This Boolean defines whether or not HTTP scripts are allowed write access to files labeled with the **public_content_rw_t** type, as used in a public file transfer service.

httpd_builtin_scripting

This Boolean defines access to **httpd** scripting. Having this Boolean enabled is often required for PHP content.

httpd_can_network_connect

When disabled, this Boolean prevents HTTP scripts and modules from initiating a connection to a network or remote port. Enable this Boolean to allow this access.

httpd_can_network_connect_db

When disabled, this Boolean prevents HTTP scripts and modules from initiating a connection to database servers. Enable this Boolean to allow this access.

httpd_can_network_relay

Enable this Boolean when **httpd** is being used as a forward or reverse proxy.

httpd_can_sendmail

When disabled, this Boolean prevents HTTP modules from sending mail. This can prevent spam attacks should a vulnerability be found in **httpd**. Enable this Boolean to allow HTTP modules to send mail.

httpd_dbus_avahi

When disabled, this Boolean denies **httpd** access to the **avahi** service via **D-Bus**. Enable this Boolean to allow this access.

httpd_enable_cgi

When disabled, this Boolean prevents **httpd** from executing CGI scripts. Enable this Boolean to allow **httpd** to execute CGI scripts (CGI scripts must be labeled with the **httpd_sys_script_exec_t** type).

httpd_enable_ftp_server

Enabling this Boolean allows **httpd** to listen on the FTP port and act as an FTP server.

httpd_enable_homedirs

When disabled, this Boolean prevents **httpd** from accessing user home directories. Enable this Boolean to allow **httpd** access to user home directories; for example, content in **/home/*/**.

`httpd_execmem`

When enabled, this Boolean allows **`httpd`** to execute programs that require memory addresses that are both executable and writeable. Enabling this Boolean is not recommended from a security standpoint as it reduces protection against buffer overflows, however certain modules and applications (such as Java and Mono applications) require this privilege.

`httpd_ssi_exec`

This Boolean defines whether or not server side include (SSI) elements in a web page can be executed.

`httpd_tty_comm`

This Boolean defines whether or not **`httpd`** is allowed access to the controlling terminal. Usually this access is not required, however in cases such as configuring an SSL certificate file, terminal access is required to display and process a password prompt.

`httpd_unified`

When enabled, this Boolean allows **`httpd_t`** complete access to all of the **`httpd`** types (that is to execute, read, or write sys_content_t). When disabled, there is separation in place between web content that is read-only, writeable, or executable. Disabling this Boolean ensures an extra level of security but adds the administrative overhead of having to individually label scripts and other web content based on the file access that each should have.

`httpd_use_cifs`

Enable this Boolean to allow **`httpd`** access to files on CIFS volumes that are labeled with the **`cifs_t`** type, such as file systems mounted via Samba.

`httpd_use_nfs`

Enable this Boolean to allow **`httpd`** access to files on NFS volumes that are labeled with the **`nfs_t`** type, such as file systems mounted using NFS.

Note

Due to the continuous development of the SELinux policy, the list above might not contain all Booleans related to the service at all times. To list them, run the following command:

```
~]$ getsebool -a | grep service_name
```

Run the following command to view description of a particular Boolean:

```
~]$ sepolicy booleans -b boolean_name
```

Note that the additional *policycoreutils-devel* package providing the **`sepolicy`** utility is required.

13.4. Configuration examples

The following examples provide real-world demonstrations of how SELinux complements the Apache

HTTP Server and how full function of the Apache HTTP Server can be maintained.

13.4.1. Running a static site

To create a static website, label the **.html** files for that website with the **httpd_sys_content_t** type. By default, the Apache HTTP Server cannot write to files that are labeled with the **httpd_sys_content_t** type. The following example creates a new directory to store files for a read-only website:

1. Use the **mkdir** utility as root to create a top-level directory:

   ```
   ~]# mkdir /mywebsite
   ```

2. As root, create a **/mywebsite/index.html** file. Copy and paste the following content into **/mywebsite/index.html**:

   ```
   <html>
   <h2>index.html from /mywebsite/</h2>
   </html>
   ```

3. To allow the Apache HTTP Server read only access to **/mywebsite/**, as well as files and subdirectories under it, label the directory with the **httpd_sys_content_t** type. Run the following command as root to add the label change to file-context configuration:

   ```
   ~]# semanage fcontext -a -t httpd_sys_content_t "/mywebsite(/.*)?"
   ```

4. Use the **restorecon** utility as root to make the label changes:

   ```
   ~]# restorecon -R -v /mywebsite
   restorecon reset /mywebsite context unconfined_u:object_r:default_t:s0-
   >system_u:object_r:httpd_sys_content_t:s0
   restorecon reset /mywebsite/index.html context
   unconfined_u:object_r:default_t:s0->system_u:object_r:httpd_sys_content_t:s0
   ```

5. For this example, edit the **/etc/httpd/conf/httpd.conf** file as root. Comment out the existing **DocumentRoot** option. Add a **DocumentRoot "/mywebsite"** option. After editing, these options should look as follows:

   ```
   #DocumentRoot "/var/www/html"
   DocumentRoot "/mywebsite"
   ```

6. Run the following command as root to see the status of the Apache HTTP Server. If the server is stopped, start it:

   ```
   ~]# systemctl status httpd.service
   httpd.service - The Apache HTTP Server
      Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled)
      Active: inactive (dead)
   ```

   ```
   ~]# systemctl start httpd.service
   ```

   If the server is running, restart the service by executing the following command as root (this also applies any changes made to **httpd.conf**):

```
~]# systemctl status httpd.service
httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled)
   Active: active (running) since Wed 2014-02-05 13:16:46 CET; 2s ago
```

```
~]# systemctl restart httpd.service
```

7. Use a web browser to navigate to **http://localhost/index.html**. The following is displayed:

```
index.html from /mywebsite/
```

13.4.2. Sharing NFS and CIFS volumes

By default, NFS mounts on the client side are labeled with a default context defined by policy for NFS volumes. In common policies, this default context uses the **nfs_t** type. Also, by default, Samba shares mounted on the client side are labeled with a default context defined by policy. In common policies, this default context uses the **cifs_t** type.

Depending on policy configuration, services may not be able to read files labeled with the **nfs_t** or **cifs_t** types. This may prevent file systems labeled with these types from being mounted and then read or exported by other services. Booleans can be enabled or disabled to control which services are allowed to access the **nfs_t** and **cifs_t** types.

Enable the **httpd_use_nfs** Boolean to allow **httpd** to access and share NFS volumes (labeled with the **nfs_t** type):

```
~]# setsebool -P httpd_use_nfs on
```

Enable the **httpd_use_cifs** Boolean to allow **httpd** to access and share CIFS volumes (labeled with the **cifs_t** type):

```
~]# setsebool -P httpd_use_cifs on
```

Note

Do not use the **-P** option if you do not want **setsebool** changes to persist across reboots.

13.4.3. Sharing files between services

Type Enforcement helps prevent processes from accessing files intended for use by another process. For example, by default, Samba cannot read files labeled with the **httpd_sys_content_t** type, which are intended for use by the Apache HTTP Server. Files can be shared between the Apache HTTP Server, FTP, rsync, and Samba, if the desired files are labeled with the **public_content_t** or **public_content_rw_t** type.

The following example creates a directory and files, and allows that directory and files to be shared (read only) through the Apache HTTP Server, FTP, rsync, and Samba:

1. Use the **mkdir** utility as root to create a new top-level directory to share files between multiple services:

```
~]# mkdir /shares
```

2. Files and directories that do not match a pattern in file-context configuration may be labeled with the **default_t** type. This type is inaccessible to confined services:

```
~]$ ls -dZ /shares
drwxr-xr-x  root root unconfined_u:object_r:default_t:s0 /shares
```

3. As root, create a **/shares/index.html** file. Copy and paste the following content into **/shares/index.html**:

```
<html>
<body>
<p>Hello</p>
</body>
</html>
```

4. Labeling **/shares/** with the **public_content_t** type allows read-only access by the Apache HTTP Server, FTP, rsync, and Samba. Run the following command as root to add the label change to file-context configuration:

```
~]# semanage fcontext -a -t public_content_t "/shares(/.*)?"
```

5. Use the **restorecon** utility as root to apply the label changes:

```
~]# restorecon -R -v /shares/
restorecon reset /shares context unconfined_u:object_r:default_t:s0-
>system_u:object_r:public_content_t:s0
restorecon reset /shares/index.html context
unconfined_u:object_r:default_t:s0->system_u:object_r:public_content_t:s0
```

To share **/shares/** through Samba:

1. Confirm the *samba*, *samba-common*, and *samba-client* packages are installed (version numbers may differ):

```
~]$ rpm -q samba samba-common samba-client
samba-3.4.0-0.41.el6.3.i686
samba-common-3.4.0-0.41.el6.3.i686
samba-client-3.4.0-0.41.el6.3.i686
```

   If any of these packages are not installed, install them by running the following command as root:

```
~]# yum install package-name
```

2. Edit the **/etc/samba/smb.conf** file as root. Add the following entry to the bottom of this file to share the **/shares/** directory through Samba:

```
[shares]
comment = Documents for Apache HTTP Server, FTP, rsync, and Samba
path = /shares
public = yes
writeable = no
```

3. A Samba account is required to mount a Samba file system. Run the following command as root to create a Samba account, where *username* is an existing Linux user. For example, **`smbpasswd -a testuser`** creates a Samba account for the Linux **`testuser`** user:

   ```
   ~]# smbpasswd -a testuser
   New SMB password: Enter a password
   Retype new SMB password: Enter the same password again
   Added user testuser.
   ```

   If you run the above command, specifying a user name of an account that does not exist on the system, it causes a **`Cannot locate Unix account for '`*`username`*`'!`** error.

4. Start the Samba service:

   ```
   ~]# systemctl start smb.service
   ```

5. Run the following command to list the available shares, where *username* is the Samba account added in step 3. When prompted for a password, enter the password assigned to the Samba account in step 3 (version numbers may differ):

   ```
   ~]$ smbclient -U username -L localhost
   Enter username's password:
   Domain=[HOSTNAME] OS=[Unix] Server=[Samba 3.4.0-0.41.el6]

   Sharename       Type      Comment
   ---------       ----      -------
   shares          Disk      Documents for Apache HTTP Server, FTP, rsync, and
   Samba
   IPC$            IPC       IPC Service (Samba Server Version 3.4.0-0.41.el6)
   username        Disk      Home Directories
   Domain=[HOSTNAME] OS=[Unix] Server=[Samba 3.4.0-0.41.el6]

   Server               Comment
   ---------            -------

   Workgroup            Master
   ---------            -------
   ```

6. User the **`mkdir`** utility to create a new directory. This directory will be used to mount the **`shares`** Samba share:

   ```
   ~]# mkdir /test/
   ```

7. Run the following command as root to mount the **`shares`** Samba share to **`/test/`**, replacing *username* with the user name from step 3:

   ```
   ~]# mount //localhost/shares /test/ -o user=username
   ```

   Enter the password for *username*, which was configured in step 3.

8. View the content of the file, which is being shared through Samba:

   ```
   ~]$ cat /test/index.html
   <html>
   <body>
   <p>Hello</p>
   </body>
   </html>
   ```

To share **/shares/** through the Apache HTTP Server:

1. Confirm the *httpd* package is installed (version number may differ):

   ```
   ~]$ rpm -q httpd
   httpd-2.2.11-6.i386
   ```

   If this package is not installed, use the **yum** utility as root to install it:

   ```
   ~]# yum install httpd
   ```

2. Change into the **/var/www/html/** directory. Run the following command as root to create a link (named **shares**) to the **/shares/** directory:

   ```
   html]# ln -s /shares/ shares
   ```

3. Start the Apache HTTP Server:

   ```
   ~]# systemctl start httpd.service
   ```

4. Use a web browser to navigate to **http://localhost/shares**. The **/shares/index.html** file is displayed.

By default, the Apache HTTP Server reads an **index.html** file if it exists. If **/shares/** did not have **index.html**, and instead had **file1**, **file2**, and **file3**, a directory listing would occur when accessing **http://localhost/shares**:

1. Remove the **index.html** file:

   ```
   ~]# rm -i /shares/index.html
   ```

2. Use the **touch** utility as root to create three files in **/shares/**:

   ```
   ~]# touch /shares/file{1,2,3}
   ~]# ls -Z /shares/
   -rw-r--r--  root root system_u:object_r:public_content_t:s0 file1
   -rw-r--r--  root root unconfined_u:object_r:public_content_t:s0 file2
   -rw-r--r--  root root unconfined_u:object_r:public_content_t:s0 file3
   ```

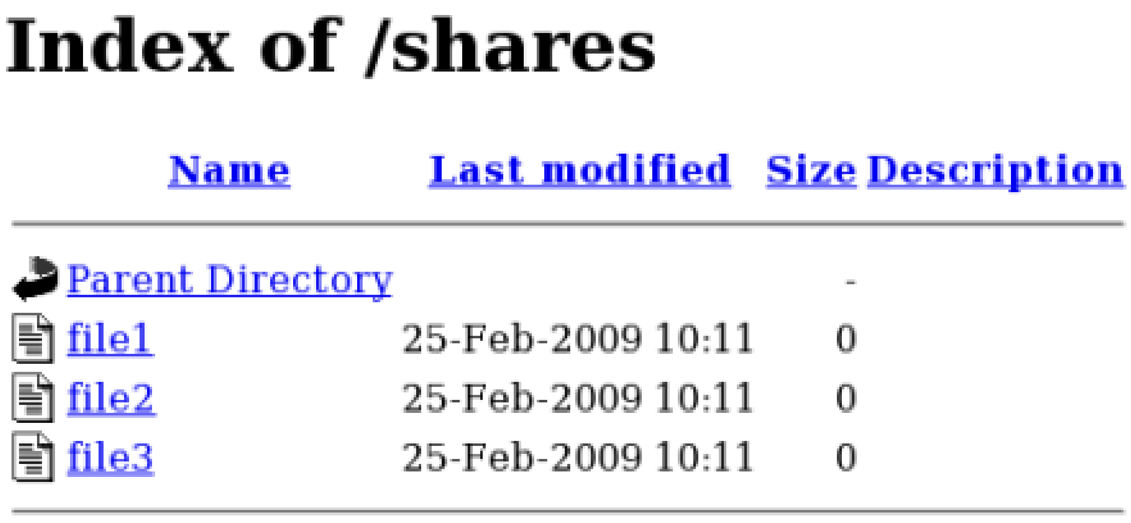
3. Run the following command as root to see the status of the Apache HTTP Server:

   ```
   ~]# systemctl status httpd.service
   httpd.service - The Apache HTTP Server
      Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled)
      Active: inactive (dead)
   ```

   If the server is stopped, start it:

   ```
   ~]# systemctl start httpd.service
   ```

4. Use a web browser to navigate to **http://localhost/shares**. A directory listing is displayed:



Index of /shares

Name	Last modified	Size	Description
Parent Directory		-	
file1	25-Feb-2009 10:11	0	
file2	25-Feb-2009 10:11	0	
file3	25-Feb-2009 10:11	0	

13.4.4. Changing port numbers

Depending on policy configuration, services may 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. Use the **semanage** utility as the root user to list the ports SELinux allows **httpd** to listen on:

```
~]# semanage port -l | grep -w http_port_t
http_port_t                    tcp      80, 443, 488, 8008, 8009, 8443
```

By default, SELinux allows **http** to listen on TCP ports 80, 443, 488, 8008, 8009, or 8443. If **/etc/httpd/conf/httpd.conf** is configured so that **httpd** listens on any port not listed for **http_port_t**, **httpd** fails to start.

To configure **httpd** to run on a port other than TCP ports 80, 443, 488, 8008, 8009, or 8443:

1. Edit the **/etc/httpd/conf/httpd.conf** file as root so the **Listen** option lists a port that is not configured in SELinux policy for **httpd**. The following example configures **httpd** to listen on the 10.0.0.1 IP address, and on TCP port 12345:

```
# Change this to Listen on specific IP addresses as shown below to
# prevent Apache from glomming onto all bound IP addresses (0.0.0.0)
#
#Listen 12.34.56.78:80
Listen 10.0.0.1:12345
```

2. Run the following command as the root user to add the port to SELinux policy configuration:

```
~]# semanage port -a -t http_port_t -p tcp 12345
```

3. Confirm that the port is added:

```
~]# semanage port -l | grep -w http_port_t
http_port_t                    tcp      12345, 80, 443, 488, 8008, 8009, 8443
```

If you no longer run **httpd** on port 12345, use the **semanage** utility as root to remove the port from policy configuration:

```
~]# semanage port -d -t http_port_t -p tcp 12345
```

[18] For more information, see the appropriate chapter in the System Administrator's Guide.

Chapter 14. Samba

Samba is an open-source implementation of the Server Message Block (SMB) and Common Internet File System (CIFS) protocols that provides file and print services between clients across various operating systems. [19]

In Red Hat Enterprise Linux, the *samba* package provides the Samba server. Run the following command to see if the *samba* package is installed:

```
~]$ rpm -q samba
package samba is not installed
```

If it is not installed and you want to use Samba, use the **yum** utility as the root user to install it:

```
~]# yum install samba
```

14.1. Samba and SELinux

When SELinux is enabled, the Samba server (**smbd**) runs confined by default. Confined services run in their own domains, and are separated from other confined services. The following example demonstrates the **smbd** process running in its own domain. This example assumes the *samba* package is installed:

1. Run the **getenforce** command to confirm SELinux is running in enforcing mode:

   ```
   ~]$ getenforce
   Enforcing
   ```

   The command returns **Enforcing** when SELinux is running in enforcing mode.

2. Run the following command as root to start **smbd**:

   ```
   ~]# systemctl start smb.service
   ```

   Confirm that the service is running. The output should include the information below (only the time stamp will differ):

   ```
   ~]# systemctl status smb.service
   smb.service - Samba SMB Daemon
      Loaded: loaded (/usr/lib/systemd/system/smb.service; disabled)
      Active: active (running) since Mon 2013-08-05 12:17:26 CEST; 2h 22min ago
   ```

3. To view the **smbd** processes, execute the following command:

   ```
   ~]$ ps -eZ | grep smb
   system_u:system_r:smbd_t:s0      9653 ?        00:00:00 smbd
   system_u:system_r:smbd_t:s0      9654?        00:00:00 smbd
   ```

   The SELinux context associated with the **smbd** processes is **system_u:system_r:smbd_t:s0**. The second last part of the context, **smbd_t**, is the type. A type defines a domain for processes and a type for files. In this case, the **smbd** processes are running in the **smbd_t** domain.

Files must be labeled correctly to allow **smbd** to access and share them. For example, **smbd** can read and write to files labeled with the **samba_share_t** type, but by default, cannot access files labeled with the **httpd_sys_content_t** type, which is intended for use by the Apache HTTP Server. Booleans must be enabled to allow certain behavior, such as allowing home directories and NFS volumes to be exported through Samba, as well as to allow Samba to act as a domain controller.

14.2. Types

Label files with the **samba_share_t** type to allow Samba to share them. Only label files you have created, and do not relabel system files with the **samba_share_t** type: Booleans can be enabled to share such files and directories. SELinux allows Samba to write to files labeled with the **samba_share_t** type, as long as the **/etc/samba/smb.conf** file and Linux permissions are set accordingly.

The **samba_etc_t** type is used on certain files in the **/etc/samba/** directory, such as **smb.conf**. Do not manually label files with the **samba_etc_t** type. If files in this directory are not labeled correctly, run the **restorecon -R -v /etc/samba** command as the root user to restore such files to their default contexts. If **/etc/samba/smb.conf** is not labeled with the **samba_etc_t** type, starting the Samba service may fail and an SELinux denial message may be logged. The following is an example denial message when **/etc/samba/smb.conf** was labeled with the **httpd_sys_content_t** type:

```
setroubleshoot: SELinux is preventing smbd (smbd_t) "read" to ./smb.conf
(httpd_sys_content_t). For complete SELinux messages. run sealert -l deb33473-1069-
482b-bb50-e4cd05ab18af
```

14.3. Booleans

SELinux is based on the least level of access required for a service to run. Services can be run in a variety of ways; therefore, you need to specify how you run your services. Use the following Booleans to set up SELinux:

smbd_anon_write

Having this Boolean enabled allows **smbd** to write to a public directory, such as an area reserved for common files that otherwise has no special access restrictions.

samba_create_home_dirs

Having this Boolean enabled allows Samba to create new home directories independently. This is often done by mechanisms such as PAM.

samba_domain_controller

When enabled, this Boolean allows Samba to act as a domain controller, as well as giving it permission to execute related commands such as **useradd**, **groupadd**, and **passwd**.

samba_enable_home_dirs

Enabling this Boolean allows Samba to share users' home directories.

samba_export_all_ro

Export any file or directory, allowing read-only permissions. This allows files and directories that are not labeled with the **samba_share_t** type to be shared through Samba. When the **samba_export_all_ro** Boolean is enabled, but the **samba_export_all_rw**

Boolean is disabled, write access to Samba shares is denied, even if write access is configured in **`/etc/samba/smb.conf`**, as well as Linux permissions allowing write access.

`samba_export_all_rw`

Export any file or directory, allowing read and write permissions. This allows files and directories that are not labeled with the **`samba_share_t`** type to be exported through Samba. Permissions in **`/etc/samba/smb.conf`** and Linux permissions must be configured to allow write access.

`samba_run_unconfined`

Having this Boolean enabled allows Samba to run unconfined scripts in the **`/var/lib/samba/scripts/`** directory.

`samba_share_fusefs`

This Boolean must be enabled for Samba to share fusefs file systems.

`samba_share_nfs`

Disabling this Boolean prevents **`smbd`** from having full access to NFS shares through Samba. Enabling this Boolean will allow Samba to share NFS volumes.

`use_samba_home_dirs`

Enable this Boolean to use a remote server for Samba home directories.

`virt_use_samba`

Allow virtual machine access to CIFS files.

Note

Due to the continuous development of the SELinux policy, the list above might not contain all Booleans related to the service at all times. To list them, run the following command:

```
~]$ getsebool -a | grep service_name
```

Run the following command to view description of a particular Boolean:

```
~]$ sepolicy booleans -b boolean_name
```

Note that the additional *policycoreutils-devel* package providing the **`sepolicy`** utility is required.

14.4. Configuration examples

The following examples provide real-world demonstrations of how SELinux complements the Samba server and how full function of the Samba server can be maintained.

14.4.1. Sharing directories you create

The following example creates a new directory, and shares that directory through Samba:

1. Confirm that the *samba*, *samba-common*, and *samba-client* packages are installed:

   ```
   ~]$ rpm -q samba samba-common samba-client
   package samba is not installed
   package samba-common is not installed
   package samba-client is not installed
   ```

   If any of these packages are not installed, install them by using the **yum** utility as root:

   ```
   ~]# yum install package-name
   ```

2. Use the **mkdir** utility as root to create a new top-level directory to share files through Samba:

   ```
   ~]# mkdir /myshare
   ```

3. Use the **touch** utility root to create an empty file. This file is used later to verify the Samba share mounted correctly:

   ```
   ~]# touch /myshare/file1
   ```

4. SELinux allows Samba to read and write to files labeled with the **samba_share_t** type, as long as the **/etc/samba/smb.conf** file and Linux permissions are set accordingly. Run the following command as root to add the label change to file-context configuration:

   ```
   ~]# semanage fcontext -a -t samba_share_t "/myshare(/.*)?"
   ```

5. Use the **restorecon** utility as root to apply the label changes:

   ```
   ~]# restorecon -R -v /myshare
   restorecon reset /myshare context unconfined_u:object_r:default_t:s0-
   >system_u:object_r:samba_share_t:s0
   restorecon reset /myshare/file1 context unconfined_u:object_r:default_t:s0-
   >system_u:object_r:samba_share_t:s0
   ```

6. Edit **/etc/samba/smb.conf** as root. Add the following to the bottom of this file to share the **/myshare/** directory through Samba:

   ```
   [myshare]
   comment = My share
   path = /myshare
   public = yes
   writeable = no
   ```

7. A Samba account is required to mount a Samba file system. Run the following command as root to create a Samba account, where *username* is an existing Linux user. For example, **smbpasswd -a testuser** creates a Samba account for the Linux **testuser** user:

   ```
   ~]# smbpasswd -a testuser
   New SMB password: Enter a password
   Retype new SMB password: Enter the same password again
   Added user testuser.
   ```

   If you run the above command, specifying a user name of an account that does not exist on the system, it causes a **Cannot locate Unix account for '*username*'!** error.

8. Start the Samba service:

```
~]# systemctl start smb.service
```

9. Run the following command to list the available shares, where *username* is the Samba account added in step 7. When prompted for a password, enter the password assigned to the Samba account in step 7 (version numbers may differ):

```
~]$ smbclient -U username -L localhost
Enter username's password:
Domain=[HOSTNAME] OS=[Unix] Server=[Samba 3.4.0-0.41.el6]

Sharename       Type      Comment
---------       ----      -------
myshare         Disk      My share
IPC$            IPC       IPC Service (Samba Server Version 3.4.0-0.41.el6)
username        Disk      Home Directories
Domain=[HOSTNAME] OS=[Unix] Server=[Samba 3.4.0-0.41.el6]

Server               Comment
---------            -------

Workgroup            Master
---------            -------
```

10. Use the **mkdir** utility as root to create a new directory. This directory will be used to mount the **myshare** Samba share:

```
~]# mkdir /test/
```

11. Run the following command as root to mount the **myshare** Samba share to **/test/**, replacing *username* with the user name from step 7:

```
~]# mount //localhost/myshare /test/ -o user=username
```

Enter the password for *username*, which was configured in step 7.

12. Run the following command to view the **file1** file created in step 3:

```
~]$ ls /test/
file1
```

14.4.2. Sharing a website

It may not be possible to label files with the **samba_share_t** type, for example, when wanting to share a website in the **/var/www/html/** directory. For these cases, use the **samba_export_all_ro** Boolean to share any file or directory (regardless of the current label), allowing read only permissions, or the **samba_export_all_rw** Boolean to share any file or directory (regardless of the current label), allowing read and write permissions.

The following example creates a file for a website in **/var/www/html/**, and then shares that file through Samba, allowing read and write permissions. This example assumes the *httpd*, *samba*, *samba-common*, *samba-client*, and *wget* packages are installed:

1. As the root user, create a **/var/www/html/file1.html** file. Copy and paste the following content into this file:

```
<html>
<h2>File being shared through the Apache HTTP Server and Samba.</h2>
</html>
```

2. Run the following command to view the SELinux context of **`file1.html`**:

   ```
   ~]$ ls -Z /var/www/html/file1.html
   -rw-r--r--. root root unconfined_u:object_r:httpd_sys_content_t:s0
   /var/www/html/file1.html
   ```

   The file is labeled with the **`httpd_sys_content_t`**. By default, the Apache HTTP Server can access this type, but Samba cannot.

3. Start the Apache HTTP Server:

   ```
   ~]# systemctl start httpd.service
   ```

4. Change into a directory your user has write access to, and run the following command. Unless there are changes to the default configuration, this command succeeds:

   ```
   ~]$ wget http://localhost/file1.html
   Resolving localhost... 127.0.0.1
   Connecting to localhost|127.0.0.1|:80... connected.
   HTTP request sent, awaiting response... 200 OK
   Length: 84 [text/html]
   Saving to: `file1.html.1'

   100%[=======================>] 84          --.-K/s   in 0s

   `file1.html.1' saved [84/84]
   ```

5. Edit **`/etc/samba/smb.conf`** as root. Add the following to the bottom of this file to share the **`/var/www/html/`** directory through Samba:

   ```
   [website]
   comment = Sharing a website
   path = /var/www/html/
   public = no
   writeable = no
   ```

6. The **`/var/www/html/`** directory is labeled with the **`httpd_sys_content_t`** type. By default, Samba cannot access files and directories labeled with the this type, even if Linux permissions allow it. To allow Samba access, enable the **`samba_export_all_ro`** Boolean:

   ```
   ~]# setsebool -P samba_export_all_ro on
   ```

   Do not use the **`-P`** option if you do not want the change to persist across reboots. Note that enabling the **`samba_export_all_ro`** Boolean allows Samba to access any type.

7. Start the Samba service:

   ```
   ~]# systemctl start smb.service
   ```

[19] For more information, see the appropriate chapter in the System Administrator's Guide.

Chapter 15. File Transfer Protocol

File Transfer Protocol (FTP) is one of the oldest and most commonly used protocols found on the Internet today. Its purpose is to reliably transfer files between computer hosts on a network without requiring the user to log directly into the remote host or have knowledge of how to use the remote system. It allows users to access files on remote systems using a standard set of simple commands.

The Very Secure FTP Daemon (**vsftpd**) is designed from the ground up to be fast, stable, and, most importantly, secure. Its ability to handle large numbers of connections efficiently and securely is why **vsftpd** is the only stand-alone FTP distributed with Red Hat Enterprise Linux.

In Red Hat Enterprise Linux, the *vsftpd* package provides the Very Secure FTP daemon. Run the following command to see if *vsftpd* is installed:

```
~]$ rpm -q vsftpd
package vsftpd is not installed
```

If you want an FTP server and the *vsftpd* package is not installed, use the **yum** utility as the root user to install it:

```
~]# yum install vsftpd
```

15.1. FTP and SELinux

The **vsftpd** FTP daemon runs confined by default. SELinux policy defines how **vsftpd** interacts with files, processes, and with the system in general. For example, when an authenticated user logs in via FTP, they cannot read from or write to files in their home directories: SELinux prevents **vsftpd** from accessing user home directories by default. Also, by default, **vsftpd** does not have access to NFS or CIFS volumes, and anonymous users do not have write access, even if such write access is configured in the **/etc/vsftpd/vsftpd.conf** file. Booleans can be enabled to allow the previously mentioned access.

The following example demonstrates an authenticated user logging in, and an SELinux denial when trying to view files in their home directory. This example assumes that the *vsftpd* package is installed, that the SELinux targeted policy is used, and that SELinux is running in enforcing mode.

1. In Red Hat Enterprise Linux, **vsftpd** only allows anonymous users to log in by default. To allow authenticated users to log in, edit **/etc/vsftpd/vsftpd.conf** as root. Make sure the **local_enable=YES** option is uncommented:

   ```
   # Uncomment this to allow local users to log in.
   local_enable=YES
   ```

2. Start the **vsftpd** service:

   ```
   ~]# systemctl start vsftpd.service
   ```

   Confirm that the service is running. The output should include the information below (only the time stamp will differ):

   ```
   ~]# systemctl status vsftpd.service
   vsftpd.service - Vsftpd ftp daemon
      Loaded: loaded (/usr/lib/systemd/system/vsftpd.service; disabled)
      Active: active (running) since Tue 2013-08-06 14:42:07 CEST; 6s ago
   ```

If the service was running before editing **vsftpd.conf**, restart the service to apply the configuration changes:

```
~]# systemctl restart vsftpd.service
```

3. Run the following command as the user you are currently logged in with. When prompted for your name, make sure your user name is displayed. If the correct user name is displayed, press **Enter**, otherwise, enter the correct user name:

```
~]$ ftp localhost
Connected to localhost (127.0.0.1).
220 (vsFTPd 2.1.0)
Name (localhost:username):
331 Please specify the password.
Password: Enter your password
500 OOPS: cannot change directory:/home/username
Login failed.
ftp>
```

4. An SELinux denial message similar to the following is logged:

```
setroubleshoot: SELinux is preventing the ftp daemon from reading users home
directories (username). For complete SELinux messages. run sealert -l
c366e889-2553-4c16-b73f-92f36a1730ce
```

5. Access to home directories has been denied by SELinux. This can be fixed by activating the **ftp_home_dir** Boolean. Enable this Boolean by running the following command as root:

```
~]# setsebool -P ftp_home_dir=1
```

Note

Do not use the -P option if you do not want changes to persist across reboots.

Try to log in again. Now that SELinux is allowing access to home directories using the **ftp_home_dir** Boolean, logging in will succeed.

15.2. Types

By default, anonymous users have read access to files in the **/var/ftp/** directory when they log in via FTP. This directory is labeled with the **public_content_t** type, allowing only read access, even if write access is configured in **/etc/vsftpd/vsftpd.conf**. The **public_content_t** type is accessible to other services, such as Apache HTTP Server, Samba, and NFS.

Use one of the following types to share files through FTP:

public_content_t

Label files and directories you have created with the **public_content_t** type to share them read-only through **vsftpd**. Other services, such as Apache HTTP Server, Samba, and NFS, also have access to files labeled with this type. Files labeled with the **public_content_t** type cannot be written to, even if Linux permissions allow write access. If you require write access, use the **public_content_rw_t** type.

`public_content_rw_t`

Label files and directories you have created with the **`public_content_rw_t`** type to share them with read and write permissions through **`vsftpd`**. Other services, such as Apache HTTP Server, Samba, and NFS, also have access to files labeled with this type. Remember that Booleans for each service must be enabled before they can write to files labeled with this type.

15.3. Booleans

SELinux is based on the least level of access required for a service to run. Services can be run in a variety of ways; therefore, you need to specify how you run your services. Use the following Booleans to set up SELinux:

`ftpd_anon_write`

When disabled, this Boolean prevents **`vsftpd`** from writing to files and directories labeled with the **`public_content_rw_t`** type. Enable this Boolean to allow users to upload files using FTP. The directory where files are uploaded to must be labeled with the **`public_content_rw_t`** type and Linux permissions must be set accordingly.

`ftpd_full_access`

When this Boolean is enabled, only Linux (DAC) permissions are used to control access, and authenticated users can read and write to files that are not labeled with the **`public_content_t`** or **`public_content_rw_t`** types.

`ftpd_use_cifs`

Having this Boolean enabled allows **`vsftpd`** to access files and directories labeled with the **`cifs_t`** type; therefore, having this Boolean enabled allows you to share file systems mounted via Samba through **`vsftpd`**.

`ftpd_use_nfs`

Having this Boolean enabled allows **`vsftpd`** to access files and directories labeled with the **`nfs_t`** type; therefore, this Boolean allows you to share file systems mounted using NFS through **`vsftpd`**.

`ftp_home_dir`

Having this Boolean enabled allows authenticated users to read and write to files in their home directories. When this Boolean is disabled, attempting to download a file from a home directory results in an error such as **`550 Failed to open file`**. An SELinux denial message is logged.

`ftpd_connect_db`

Allow FTP daemons to initiate a connection to a database.

`httpd_enable_ftp_server`

Allow the **`httpd`** daemon to listen on the FTP port and act as a FTP server.

`tftp_anon_write`

Having this Boolean enabled allows TFTP access to a public directory, such as an area reserved for common files that otherwise has no special access restrictions.

Note

Due to the continuous development of the SELinux policy, the list above might not contain all Booleans related to the service at all times. To list them, run the following command:

```
~]$ getsebool -a | grep service_name
```

Run the following command to view description of a particular Boolean:

```
~]$ sepolicy booleans -b boolean_name
```

Note that the additional *policycoreutils-devel* package providing the **sepolicy** utility is required.

15.4. Configuration Examples

15.4.1. Uploading to an FTP site

The following example creates an FTP site that allows a dedicated user to upload files. It creates the directory structure and the required SELinux configuration changes:

1. Run the following command as the root user to enable access to FTP home directories:

   ```
   ~]# setsebool ftp_home_dir=1
   ```

2. Use the **mkdir** utility as root to create a new top-level directory:

   ```
   ~]# mkdir -p /myftp/pub
   ```

3. Set Linux permissions on the **/myftp/pub/** directory to allow a Linux user write access. This example changes the owner and group from root to owner **user1** and group root. Replace **user1** with the user you want to give write access to:

   ```
   ~]# chown user1:root /myftp/pub
   ~]# chmod 775 /myftp/pub
   ```

   The **chown** command changes the owner and group permissions. It changes the mode, allowing the **user1** user read, write, and execute permissions, and members of the root group read, write, and execute permissions. Everyone else has read and execute permissions: this is required to allow the Apache HTTP Server to read files from this directory.

4. When running SELinux, files and directories must be labeled correctly to allow access. Setting Linux permissions is not enough. Files labeled with the **public_content_t** type allow them to be read by FTP, Apache HTTP Server, Samba, and rsync. Files labeled with the **public_content_rw_t** type can be written to by FTP. Other services, such as Samba, require Booleans to be set before they can write to files labeled with the

public_content_rw_t type. Label the top-level directory (**/myftp/**) with the **public_content_t** type, to prevent copied or newly-created files under this directory from being written to or modified by services. Run the following command as root to add the label change to file-context configuration:

```
~]# semanage fcontext -a -t public_content_t /myftp
```

5. Use the **restorecon** utility to apply the label change:

```
~]# restorecon -R -v /myftp/
restorecon reset /myftp context unconfined_u:object_r:default_t:s0-
>system_u:object_r:public_content_t:s0
```

6. Confirm **/myftp/** is labeled with the **public_content_t** type, and **/myftp/pub/** is labeled with the **default_t** type:

```
~]$ ls -dZ /myftp/
drwxr-xr-x. root root system_u:object_r:public_content_t:s0 /myftp/
~]$ ls -dZ /myftp/pub/
drwxrwxr-x. user1 root unconfined_u:object_r:default_t:s0 /myftp/pub/
```

7. FTP must be allowed to write to a directory before users can upload files through FTP. SELinux allows FTP to write to directories labeled with the **public_content_rw_t** type. This example uses **/myftp/pub/** as the directory FTP can write to. Run the following command as root to add the label change to file-context configuration:

```
~]# semanage fcontext -a -t public_content_rw_t "/myftp/pub(/.*)?"
```

8. Use **restorecon** as root to apply the label change:

```
~]# restorecon -R -v /myftp/pub
restorecon reset /myftp/pub context system_u:object_r:default_t:s0-
>system_u:object_r:public_content_rw_t:s0
```

9. The **ftpd_anon_write** Boolean must be enabled to allow **vsftpd** to write to files that are labeled with the **public_content_rw_t** type. Run the following command as root to enable this Boolean:

```
~]# setsebool -P ftpd_anon_write on
```

Note

Do not use the **-P** option if you do not want changes to persist across reboots.

The following example demonstrates logging in via FTP and uploading a file. This example uses the **user1** user from the previous example, where **user1** is the dedicated owner of the **/myftp/pub/** directory:

1. Change into your home directory. Then, create a directory to store files to upload via FTP:

```
~]$ cd ~/
```

```
~]$ mkdir myftp
```

2. Change into the **~/myftp/** directory:

   ```
   ~]$ cd ~/myftp
   ```

   In this directory, create an **ftpupload** file. Copy the following contents into this file:

   ```
   File upload via FTP from a home directory.
   ```

3. Confirm that the **ftpd_anon_write** Boolean is enabled:

   ```
   ~]$ getsebool ftpd_anon_write
   ftpd_anon_write --> on
   ```

   If this Boolean is disabled, run the following command as root to enable it:

   ```
   ~]# setsebool -P ftpd_anon_write on
   ```

   Note

   Do not use the **-P** option if you do not want the change to persist across reboots.

4. Start the **vsftpd** service:

   ```
   ~]# systemctl start vsftpd.service
   ```

5. Run the following command. When prompted for a user name, enter the user name of the user who has write access, then, enter the correct password for that user:

   ```
   ~]$ ftp localhost
   Connected to localhost (127.0.0.1).
   220 (vsFTPd 2.1.0)
   Name (localhost:username):
   331 Please specify the password.
   Password: Enter the correct password
   230 Login successful.
   Remote system type is UNIX.
   Using binary mode to transfer files.
   ftp> cd myftp
   250 Directory successfully changed.
   ftp> put ftpupload
   local: ftpupload remote: ftpupload
   227 Entering Passive Mode (127,0,0,1,241,41).
   150 Ok to send data.
   226 File receive OK.
   ftp> 221 Goodbye.
   ```

   The upload succeeds as the **ftpd_anon_write** Boolean is enabled.

Chapter 16. Network File System

A Network File System (NFS) allows remote hosts to mount file systems over a network and interact with those file systems as though they are mounted locally. This enables system administrators to consolidate resources onto centralized servers on the network. [20]

In Red Hat Enterprise Linux, the *nfs-utils* package is required for full NFS support. Run the following command to see if the *nfs-utils* is installed:

```
~]$ rpm -q nfs-utils
package nfs-utils is not installed
```

If it is not installed and you want to use NFS, use the **yum** utility as root to install it:

```
~]# yum install nfs-utils
```

16.1. NFS and SELinux

When running SELinux, the NFS daemons are confined by default except the **nfsd** process, which is labeled with the unconfined **kernel_t** domain type. The SELinux policy allows NFS to share files by default. Also, passing SELinux labels between a client and the server is supported, which provides better security control of confined domains accessing NFS volumes. For example, when a home directory is set up on an NFS volume, it is possible to specify confined domains that are able to access only the home directory and not other directories on the volume. Similarly, applications, such as Secure Virtualization, can set the label of an image file on an NFS volume, thus increasing the level of separation of virtual machines.

The support for labeled NFS is disabled by default. To enable it, see Section 16.4.1, “Enabling SELinux Labeled NFS Support”.

16.2. Types

By default, mounted NFS volumes on the client side are labeled with a default context defined by policy for NFS. In common policies, this default context uses the **nfs_t** type. The root user is able to override the default type using the **mount -context** option. The following types are used with NFS. Different types allow you to configure flexible access:

var_lib_nfs_t

This type is used for existing and new files copied to or created in the **/var/lib/nfs/** directory. This type should not need to be changed in normal operation. To restore changes to the default settings, run the **restorecon -R -v /var/lib/nfs** command as the root user.

nfsd_exec_t

The **/usr/sbin/rpc.nfsd** file is labeled with the **nfsd_exec_t**, as are other system executables and libraries related to NFS. Users should not label any files with this type. **nfsd_exec_t** will transition to **nfsd_t**.

16.3. Booleans

SELinux is based on the least level of access required for a service to run. Services can be run in a variety of ways; therefore, you need to specify how you run your services. Use the following Booleans to set up SELinux:

`ftpd_use_nfs`

When enabled, this Boolean allows the **`ftpd`** daemon to access NFS volumes.

`cobbler_use_nfs`

When enabled, this Boolean allows the **`cobblerd`** daemon to access NFS volumes.

`git_system_use_nfs`

When enabled, this Boolean allows the Git system daemon to read system shared repositories on NFS volumes.

`httpd_use_nfs`

When enabled, this Boolean allows the **`httpd`** daemon to access files stored on NFS volumes.

`samba_share_nfs`

When enabled, this Boolean allows the **`smbd`** daemon to share NFS volumes. When disabled, this Boolean prevents **`smbd`** from having full access to NFS shares via Samba.

`sanlock_use_nfs`

When enabled, this Boolean allows the **`sanlock`** daemon to manage NFS volumes.

`sge_use_nfs`

When enabled, this Boolean allows the **`sge`** scheduler to access NFS volumes.

`use_nfs_home_dirs`

When enabled, this Boolean adds support for NFS home directories.

`virt_use_nfs`

When enabled, this Boolean allows confident virtual guests to manage files on NFS volumes.

`xen_use_nfs`

When enabled, this Boolean allows **`Xen`** to manage files on NFS volumes.

`git_cgi_use_nfs`

When enabled, this Boolean allows the Git Common Gateway Interface (CGI) to access NFS volumes.

Note

Due to the continuous development of the SELinux policy, the list above might not contain all Booleans related to the service at all times. To list them, run the following command:

```
~]$ getsebool -a | grep service_name
```

Run the following command to view description of a particular Boolean:

```
~]$ sepolicy booleans -b boolean_name
```

Note that the additional *policycoreutils-devel* package providing the **sepolicy** utility is required.

16.4. Configuration Examples

16.4.1. Enabling SELinux Labeled NFS Support

The following example demonstrates how to enable SELinux labeled NFS support. This example assumes that the *nfs-utils* package is installed, that the SELinux targeted policy is used, and that SELinux is running in enforcing mode.

Note

Steps 1-3 are supposed to be performed on the NFS server, **nfs-srv**.

1. If the NFS server is running, stop it:

   ```
   [nfs-srv]# systemctl stop nfs
   ```

   Confirm that the server is stopped:

   ```
   [nfs-srv]# systemctl status nfs
   nfs-server.service - NFS Server
      Loaded: loaded (/usr/lib/systemd/system/nfs-server.service; disabled)
      Active: inactive (dead)
   ```

2. Edit the **/etc/sysconfig/nfs** file to set the **RPCNFSDARGS** flag to **"-V 4.2"**:

   ```
   # Optional arguments passed to rpc.nfsd. See rpc.nfsd(8)
   RPCNFSDARGS="-V 4.2"
   ```

3. Start the server again and confirm that it is running. The output will contain information below, only the time stamp will differ:

   ```
   [nfs-srv]# systemctl start nfs
   ```

   ```
   [nfs-srv]# systemctl status nfs
   nfs-server.service - NFS Server
      Loaded: loaded (/usr/lib/systemd/system/nfs-server.service; disabled)
      Active: active (exited) since Wed 2013-08-28 14:07:11 CEST; 4s ago
   ```

4. On the client side, mount the NFS server:

```
[nfs-client]# mount -o v4.2 server:mntpoint localmountpoint
```

5. All SELinux labels are now successfully passed from the server to the client:

```
[nfs-srv]$ ls -Z file
-rw-rw-r--. user user unconfined_u:object_r:svirt_image_t:s0 file
[nfs-client]$ ls -Z file
-rw-rw-r--. user user unconfined_u:object_r:svirt_image_t:s0 file
```

Note

If you enable labeled NFS support for home directories or other content, the content will be labeled the same as it was on an EXT file system. Also note that mounting systems with different versions of NFS or an attempt to mount a server that does not support labeled NFS could cause errors to be returned.

[20] Refer to the Network File System (NFS) chapter in the Storage Administration Guide for more information.

Chapter 17. Berkeley Internet Name Domain

BIND performs name resolution services via the **named** daemon. BIND lets users locate computer resources and services by name instead of numerical addresses.

In Red Hat Enterprise Linux, the *bind* package provides a DNS server. Run the following command to see if the *bind* package is installed:

```
~]$ rpm -q bind
package bind is not installed
```

If it is not installed, use the **yum** utility as the root user to install it:

```
~]# yum install bind
```

17.1. BIND and SELinux

The default permissions on the **/var/named/slaves/**, **/var/named/dynamic/** and **/var/named/data/** directories allow zone files to be updated via zone transfers and dynamic DNS updates. Files in **/var/named/** are labeled with the **named_zone_t** type, which is used for master zone files.

For a slave server, configure the **/etc/named.conf** file to place slave zones in **/var/named/slaves/**. The following is an example of a domain entry in **/etc/named.conf** for a slave DNS server that stores the zone file for **testdomain.com** in **/var/named/slaves/**:

```
zone "testdomain.com" {
   type slave;
   masters { IP-address; };
   file "/var/named/slaves/db.testdomain.com";
         };
```

If a zone file is labeled **named_zone_t**, the **named_write_master_zones** Boolean must be enabled to allow zone transfers and dynamic DNS to update the zone file. Also, the mode of the parent directory has to be changed to allow the **named** user or group read, write and execute access.

If zone files in **/var/named/** are labeled with the **named_cache_t** type, a file system relabel or running **restorecon -R /var/** will change their type to **named_zone_t**.

17.2. Types

The following types are used with BIND. Different types allow you to configure flexible access:

named_zone_t

Used for master zone files. Other services cannot modify files of this type. The **named** daemon can only modify files of this type if the **named_write_master_zones** Boolean is enabled.

named_cache_t

By default, **named** can write to files labeled with this type, without additional Booleans being set. Files copied or created in the **/var/named/slaves/**,**/var/named/dynamic/** and **/var/named/data/** directories are automatically labeled with the **named_cache_t** type.

named_var_run_t

Files copied or created in the **/var/run/bind/**, **/var/run/named/**, and **/var/run/unbound/** directories are automatically labeled with the **named_var_run_t** type.

named_conf_t

BIND-related configuration files, usually stored in the **/etc/** directory, are automatically labeled with the **named_conf_t** type.

named_exec_t

BIND-related executable files, usually stored in the **/usr/sbin/** directory, are automatically labeled with the **named_exec_t** type.

named_log_t

BIND-related log files, usually stored in the **/var/log/** directory, are automatically labeled with the **named_log_t** type.

named_unit_file_t

Executable BIND-related files in the **/usr/lib/systemd/system/** directory are automatically labeled with the **named_unit_file_t** type.

17.3. Booleans

SELinux is based on the least level of access required for a service to run. Services can be run in a variety of ways; therefore, you need to specify how you run your services. Use the following Booleans to set up SELinux:

named_write_master_zones

When disabled, this Boolean prevents **named** from writing to zone files or directories labeled with the **named_zone_t** type. The daemon does not usually need to write to zone files; but in the case that it needs to, or if a secondary server needs to write to zone files, enable this Boolean to allow this action.

named_tcp_bind_http_port

When enabled, this Boolean allows BIND to bind an Apache port.

Note

Due to the continuous development of the SELinux policy, the list above might not contain all Booleans related to the service at all times. To list them, run the following command:

```
~]$ getsebool -a | grep service_name
```

Run the following command to view description of a particular Boolean:

```
~]$ sepolicy booleans -b boolean_name
```

Note that the additional *policycoreutils-devel* package providing the **sepolicy** utility is required.

17.4. Configuration Examples

17.4.1. Dynamic DNS

BIND allows hosts to update their records in DNS and zone files dynamically. This is used when a host computer's IP address changes frequently and the DNS record requires real-time modification.

Use the **/var/named/dynamic/** directory for zone files you want updated via dynamic DNS. Files created in or copied into this directory inherit Linux permissions that allow **named** to write to them. As such files are labeled with the **named_cache_t** type, SELinux allows **named** to write to them.

If a zone file in **/var/named/dynamic/** is labeled with the **named_zone_t** type, dynamic DNS updates may not be successful for a certain period of time as the update needs to be written to a journal first before being merged. If the zone file is labeled with the **named_zone_t** type when the journal attempts to be merged, an error such as the following is logged:

```
named[PID]: dumping master file: rename: /var/named/dynamic/zone-name: permission
denied
```

Also, the following SELinux denial message is logged:

```
setroubleshoot: SELinux is preventing named (named_t) "unlink" to zone-name
(named_zone_t)
```

To resolve this labeling issue, use the **restorecon** utility as root:

```
~]# restorecon -R -v /var/named/dynamic
```

Chapter 18. Concurrent Versioning System

The Concurrent Versioning System (CVS) is a free revision-control system. It is used to monitor and keep track of modifications to a central set of files which are usually accessed by several different users. It is commonly used by programmers to manage a source code repository and is widely used by open source developers.

In Red Hat Enterprise Linux, the *cvs* package provides CVS. Run the following command to see if the *cvs* package is installed:

```
~]$ rpm -q cvs
package cvs is not installed
```

If it is not installed and you want to use CVS, use the **yum** utility as root to install it:

```
~]# yum install cvs
```

18.1. CVS and SELinux

The **cvs** daemon runs labeled with the **cvs_t** type. By default in Red Hat Enterprise Linux, CVS is only allowed to read and write certain directories. The label **cvs_data_t** defines which areas **cvs** has read and write access to. When using CVS with SELinux, assigning the correct label is essential for clients to have full access to the area reserved for CVS data.

18.2. Types

The following types are used with CVS. Different types allow you to configure flexible access:

cvs_data_t

This type is used for data in a CVS repository. CVS can only gain full access to data if it has this type.

cvs_exec_t

This type is used for the **/usr/bin/cvs** binary.

18.3. Booleans

SELinux is based on the least level of access required for a service to run. Services can be run in a variety of ways; therefore, you need to specify how you run your services. Use the following Booleans to set up SELinux:

cvs_read_shadow

This Boolean allows the **cvs** daemon to access the **/etc/shadow** file for user authentication.

Note

Due to the continuous development of the SELinux policy, the list above might not contain all Booleans related to the service at all times. To list them, run the following command:

```
~]$ getsebool -a | grep service_name
```

Run the following command to view description of a particular Boolean:

```
~]$ sepolicy booleans -b boolean_name
```

Note that the additional *policycoreutils-devel* package providing the **sepolicy** utility is required.

18.4. Configuration Examples

18.4.1. Setting up CVS

This example describes a simple CVS setup and an SELinux configuration which allows remote access. Two hosts are used in this example; a CVS server with a host name of **cvs-srv** with an IP address of **192.168.1.1** and a client with a host name of **cvs-client** and an IP address of **192.168.1.100**. Both hosts are on the same subnet (192.168.1.0/24). This is an example only and assumes that the *cvs* and *xinetd* packages are installed, that the SELinux targeted policy is used, and that SELinux is running in enforced mode.

This example will show that even with full DAC permissions, SELinux can still enforce policy rules based on file labels and only allow access to certain areas that have been specifically labeled for access by CVS.

Note

Steps 1-9 are supposed be performed on the CVS server, **cvs-srv**.

1. This example requires the *cvs* and *xinetd* packages. Confirm that the packages are installed:

   ```
   [cvs-srv]$ rpm -q cvs xinetd
   package cvs is not installed
   package xinetd is not installed
   ```

   If they are not installed, use the **yum** utility as root to install it:

   ```
   [cvs-srv]# yum install cvs xinetd
   ```

2. Run the following command as root to create a group named **CVS**:

   ```
   [cvs-srv]# groupadd CVS
   ```

   This can by also done by using the **system-config-users** utility.

3. Create a user with a user name of **cvsuser** and make this user a member of the CVS group. This can be done using **system-config-users**.

4. Edit the **/etc/services** file and make sure that the CVS server has uncommented entries looking similar to the following:

```
cvspserver 2401/tcp   # CVS client/server operations
cvspserver 2401/udp   # CVS client/server operations
```

5. Create the CVS repository in the root area of the file system. When using SELinux, it is best to have the repository in the root file system so that recursive labels can be given to it without affecting any other subdirectories. For example, as root, create a **/cvs/** directory to house the repository:

```
[root@cvs-srv]# mkdir /cvs
```

6. Give full permissions to the **/cvs/** directory to all users:

```
[root@cvs-srv]# chmod -R 777 /cvs
```

Warning

This is an example only and these permissions should not be used in a production system.

7. Edit the **/etc/xinetd.d/cvs** file and make sure that the CVS section is uncommented and configured to use the **/cvs/** directory. The file should look similar to:

```
service cvspserver
{
 disable = no
 port    = 2401
 socket_type  = stream
 protocol   = tcp
 wait    = no
 user    = root
 passenv    = PATH
 server    = /usr/bin/cvs
 env    = HOME=/cvs
 server_args  = -f --allow-root=/cvs pserver
# bind    = 127.0.0.1
```

8. Start the **xinetd** daemon:

```
[cvs-srv]# systemctl start xinetd.service
```

9. Add a rule which allows inbound connections through TCP on port 2401 by using the **system-config-firewall** utility.

10. On the client side, run the following command as the **cvsuser** user:

```
[cvsuser@cvs-client]$ cvs -d /cvs init
```

11. At this point, CVS has been configured but SELinux will still deny logins and file access. To demonstrate this, set the **$CVSROOT** variable on **cvs-client** and try to log in remotely. The

following step is supposed to be performed on **`cvs-client`**:

```
[cvsuser@cvs-client]$ export CVSROOT=:pserver:cvsuser@192.168.1.1:/cvs
[cvsuser@cvs-client]$
[cvsuser@cvs-client]$ cvs login
Logging in to :pserver:cvsuser@192.168.1.1:2401/cvs
CVS password: ********
cvs [login aborted]: unrecognized auth response from 192.168.100.1: cvs
pserver: cannot open /cvs/CVSROOT/config: Permission denied
```

SELinux has blocked access. In order to get SELinux to allow this access, the following step is supposed to be performed on **`cvs-srv`**:

12. Change the context of the **`/cvs/`** directory as root in order to recursively label any existing and new data in the **`/cvs/`** directory, giving it the **`cvs_data_t`** type:

    ```
    [root@cvs-srv]# semanage fcontext -a -t cvs_data_t '/cvs(/.*)?'
    [root@cvs-srv]# restorecon -R -v /cvs
    ```

13. The client, **`cvs-client`** should now be able to log in and access all CVS resources in this repository:

    ```
    [cvsuser@cvs-client]$ export CVSROOT=:pserver:cvsuser@192.168.1.1:/cvs
    [cvsuser@cvs-client]$
    [cvsuser@cvs-client]$ cvs login
    Logging in to :pserver:cvsuser@192.168.1.1:2401/cvs
    CVS password: ********
    [cvsuser@cvs-client]$
    ```

Chapter 19. Squid Caching Proxy

Squid is a high-performance proxy caching server for web clients, supporting FTP, Gopher, and HTTP data objects. It reduces bandwidth and improves response times by caching and reusing frequently-requested web pages. [21]

In Red Hat Enterprise Linux, the *squid* package provides the Squid Caching Proxy. Run the following command to see if the *squid* package is installed:

```
~]$ rpm -q squid
package squid is not installed
```

If it is not installed and you want to use squid, use the **yum** utility as root to install it:

```
~]# yum install squid
```

19.1. Squid Caching Proxy and SELinux

When SELinux is enabled, Squid runs confined by default. Confined processes run in their own domains, and are separated from other confined processes. If a confined process is compromised by an attacker, depending on SELinux policy configuration, an attacker's access to resources and the possible damage they can do is limited. The following example demonstrates the Squid processes running in their own domain. This example assumes the *squid* package is installed:

1. Run the **getenforce** command to confirm SELinux is running in enforcing mode:

   ```
   ~]$ getenforce
   Enforcing
   ```

   The command returns **Enforcing** when SELinux is running in enforcing mode.

2. Run the following command as the root user to start the **squid** daemon:

   ```
   ~]# systemctl start squid.service
   ```

   Confirm that the service is running. The output should include the information below (only the time stamp will differ):

   ```
   ~]# systemctl status squid.service
   squid.service - Squid caching proxy
      Loaded: loaded (/usr/lib/systemd/system/squid.service; disabled)
      Active: active (running) since Mon 2013-08-05 14:45:53 CEST; 2s ago
   ```

3. Run the following command to view the **squid** processes:

   ```
   ~]$ ps -eZ | grep squid
   system_u:system_r:squid_t:s0    27018 ?        00:00:00 squid
   system_u:system_r:squid_t:s0    27020 ?        00:00:00 log_file_daemon
   ```

   The SELinux context associated with the **squid** processes is **system_u:system_r:squid_t:s0**. The second last part of the context, **squid_t**, is the type. A type defines a domain for processes and a type for files. In this case, the Squid processes are running in the **squid_t** domain.

SELinux policy defines how processes running in confined domains, such as **squid_t**, interact with files, other processes, and the system in general. Files must be labeled correctly to allow squid access to them.

When the **/etc/squid/squid.conf** file is configured so **squid** listens on a port other than the default TCP ports 3128, 3401 or 4827, the **semanage port** command must be used to add the required port number to the SELinux policy configuration. The following example demonstrates configuring **squid** to listen on a port that is not initially defined in SELinux policy configuration for it, and, as a consequence, the server failing to start. This example also demonstrates how to then configure the SELinux system to allow the daemon to successfully listen on a non-standard port that is not already defined in the policy. This example assumes the *squid* package is installed. Run each command in the example as the root user:

1. Connfirm the **squid** daemon is not running:

   ```
   ~]# systemctl status squid.service
   squid.service - Squid caching proxy
      Loaded: loaded (/usr/lib/systemd/system/squid.service; disabled)
      Active: inactive (dead)
   ```

   If the output differs, run stop the process:

   ```
   ~]# systemctl stop squid.service
   ```

2. Run the following command to view the ports SELinux allows **squid** to listen on:

   ```
   ~]# semanage port -l | grep -w -i squid_port_t
   squid_port_t                   tcp      3401, 4827
   squid_port_t                   udp      3401, 4827
   ```

3. Edit **/etc/squid/squid.conf** as root. Configure the **http_port** option so it lists a port that is not configured in SELinux policy configuration for **squid**. In this example, the daemon is configured to listen on port 10000:

   ```
   # Squid normally listens to port 3128
   http_port 10000
   ```

4. Run the **setsebool** command to make sure the **squid_connect_any** Boolean is set to off. This ensures **squid** is only permitted to operate on specific ports:

   ```
   ~]# setsebool -P squid_connect_any 0
   ```

5. Start the **squid** daemon:

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

   An SELinux denial message similar to the following is logged:

   ```
   localhost setroubleshoot: SELinux is preventing the squid (squid_t) from
   binding to port 10000. For complete SELinux messages. run sealert -l 97136444-
   4497-4fff-a7a7-c4d8442db982
   ```

6. For SELinux to allow **squid** to listen on port 10000, as used in this example, the following command is required:

```
~]# semanage port -a -t squid_port_t -p tcp 10000
```

7. Start **squid** again and have it listen on the new port:

```
~]# systemctl start squid.service
```

8. Now that SELinux has been configured to allow Squid to listen on a non-standard port (TCP 10000 in this example), it starts successfully on this port.

19.2. Types

Type Enforcement is the main permission control used in SELinux targeted policy. All files and processes are labeled with a type: types define a domain for processes and a type for files. SELinux policy rules define how types access each other, whether it be a domain accessing a type, or a domain accessing another domain. Access is only allowed if a specific SELinux policy rule exists that allows it.

The following types are used with Squid. Different types allow you to configure flexible access:

`httpd_squid_script_exec_t`

This type is used for utilities such as **`cachemgr.cgi`**, which provides a variety of statistics about Squid and its configuration.

`squid_cache_t`

Use this type for data that is cached by Squid, as defined by the **`cache_dir`** directive in **`/etc/squid/squid.conf`**. By default, files created in or copied into the **`/var/cache/squid/`** and **`/var/spool/squid/`** directories are labeled with the **`squid_cache_t`** type. Files for the squidGuard URL redirector plug-in for **`squid`** created in or copied to the **`/var/squidGuard/`** directory are also labeled with the **`squid_cache_t`** type. Squid is only able to use files and directories that are labeled with this type for its cached data.

`squid_conf_t`

This type is used for the directories and files that Squid uses for its configuration. Existing files, or those created in or copied to the **`/etc/squid/`** and **`/usr/share/squid/`** directories are labeled with this type, including error messages and icons.

`squid_exec_t`

This type is used for the **`squid`** binary, **`/usr/sbin/squid`**.

`squid_log_t`

This type is used for logs. Existing files, or those created in or copied to **`/var/log/squid/`** or **`/var/log/squidGuard/`** must be labeled with this type.

`squid_initrc_exec_t`

This type is used for the initialization file required to start **`squid`** which is located at **`/etc/rc.d/init.d/squid`**.

`squid_var_run_t`

This type is used by files in the **/var/run/** directory, especially the process id (PID) named **/var/run/squid.pid** which is created by Squid when it runs.

19.3. Booleans

SELinux is based on the least level of access required for a service to run. Services can be run in a variety of ways; therefore, you need to specify how you run your services. Use the following Booleans to set up SELinux:

squid_connect_any

When enabled, this Boolean permits Squid to initiate a connection to a remote host on any port.

squid_use_tproxy

When enabled, this Boolean allows Squid to run as a transparent proxy.

Note

Due to the continuous development of the SELinux policy, the list above might not contain all Booleans related to the service at all times. To list them, run the following command:

```
~]$ getsebool -a | grep service_name
```

Run the following command to view description of a particular Boolean:

```
~]$ sepolicy booleans -b boolean_name
```

Note that the additional *policycoreutils-devel* package providing the **sepolicy** utility is required.

19.4. Configuration Examples

19.4.1. Squid Connecting to Non-Standard Ports

The following example provides a real-world demonstration of how SELinux complements Squid by enforcing the above Boolean and by default only allowing access to certain ports. This example will then demonstrate how to change the Boolean and show that access is then allowed.

Note that this is an example only and demonstrates how SELinux can affect a simple configuration of Squid. Comprehensive documentation of Squid is beyond the scope of this document. Refer to the official Squid documentation for further details. This example assumes that the Squid host has two network interfaces, Internet access, and that any firewall has been configured to allow access on the internal interface using the default TCP port on which Squid listens (TCP 3128).

1. Confirm that the *squid* is installed:

   ```
   ~]$ rpm -q squid
   package squid is not installed
   ```

   If the package is not installed, use the **yum** utility as root to install it:

```
~]# yum install squid
```

2. Edit the main configuration file, **`/etc/squid/squid.conf`**, and confirm that the **`cache_dir`** directive is uncommented and looks similar to the following:

   ```
   cache_dir ufs /var/spool/squid 100 16 256
   ```

   This line specifies the default settings for the **`cache_dir`** directive to be used in this example; it consists of the Squid storage format (**`ufs`**), the directory on the system where the cache resides (**`/var/spool/squid`**), the amount of disk space in megabytes to be used for the cache (**`100`**), and finally the number of first-level and second-level cache directories to be created (**`16`** and **`256`** respectively).

3. In the same configuration file, make sure the **`http_access allow localnet`** directive is uncommented. This allows traffic from the **`localnet`** ACL which is automatically configured in a default installation of Squid on Red Hat Enterprise Linux. It will allow client machines on any existing RFC1918 network to have access through the proxy, which is sufficient for this simple example.

4. In the same configuration file, make sure the **`visible_hostname`** directive is uncommented and is configured to the host name of the machine. The value should be the fully qualified domain name (FQDN) of the host:

   ```
   visible_hostname squid.example.com
   ```

5. As root, run the following command to start the **`squid`** daemon. As this is the first time **`squid`** has started, this command will initialise the cache directories as specified above in the **`cache_dir`** directive and will then start the daemon:

   ```
   ~]# systemctl start squid.service
   ```

   Ensure that **`squid`** starts successfully. The output will include the information below, only the time stamp will differ:

   ```
   ~]# systemctl status squid.service
   squid.service - Squid caching proxy
      Loaded: loaded (/usr/lib/systemd/system/squid.service; disabled)
      Active: active (running) since Thu 2014-02-06 15:00:24 CET; 6s ago
   ```

6. Confirm that the **`squid`** process ID (PID) has started as a confined service, as seen here by the **`squid_var_run_t`** value:

   ```
   ~]# ls -lZ /var/run/squid.pid
   -rw-r--r--. root squid unconfined_u:object_r:squid_var_run_t:s0
   /var/run/squid.pid
   ```

7. At this point, a client machine connected to the **`localnet`** ACL configured earlier is successfully able to use the internal interface of this host as its proxy. This can be configured in the settings for all common web browsers, or system-wide. Squid is now listening on the default port of the target machine (TCP 3128), but the target machine will only allow outgoing connections to other services on the Internet via common ports. This is a policy defined by SELinux itself. SELinux will deny access to non-standard ports, as shown in the next step:

8. When a client makes a request using a non-standard port through the Squid proxy such as a website listening on TCP port 10000, a denial similar to the following is logged:

```
SELinux is preventing the squid daemon from connecting to network port 10000
```

9. To allow this access, the **`squid_connect_any`** Boolean must be modified, as it is disabled by default:

   ```
   ~]# setsebool -P squid_connect_any on
   ```

   Note

   Do not use the **`-P`** option if you do not want **`setsebool`** changes to persist across reboots.

10. The client will now be able to access non-standard ports on the Internet as Squid is now permitted to initiate connections to any port, on behalf of its clients.

[21] Refer to the Squid Caching Proxy project page for more information.

Chapter 20. MariaDB (a replacement for MySQL)

The MariaDB database is a multi-user, multi-threaded SQL database server that consists of the MariaDB server daemon (**mysqld**) and many client programs and libraries. [22]

In Red Hat Enterprise Linux, the *mariadb-server* package provides MariaDB. Run the following command to see if the *mariadb-server* package is installed:

```
~]$ rpm -q mariadb-server
package mariadb-server is not installed
```

If it is not installed, use the **yum** utility as root to install it:

```
~]# yum install mariadb-server
```

20.1. MariaDB and SELinux

When MariaDB is enabled, it runs confined by default. Confined processes run in their own domains, and are separated from other confined processes. If a confined process is compromised by an attacker, depending on SELinux policy configuration, an attacker's access to resources and the possible damage they can do is limited. The following example demonstrates the MariaDB processes running in their own domain. This example assumes the *mariadb-server* package is installed:

1. Run the **getenforce** command to confirm SELinux is running in enforcing mode:

   ```
   ~]$ getenforce
   Enforcing
   ```

   The command returns **Enforcing** when SELinux is running in enforcing mode.

2. Run the following command as the root user to start **mariadb**:

   ```
   ~]# systemctl start mariadb.service
   ```

   Confirm that the service is running. The output should include the information below (only the time stamp will differ):

   ```
   ~]# systemctl status mariadb.service
   mariadb.service - MariaDB database server
      Loaded: loaded (/usr/lib/systemd/system/mariadb.service; disabled)
      Active: active (running) since Mon 2013-08-05 11:20:11 CEST; 3h 28min ago
   ```

3. Run the following command to view the **mysqld** processes:

   ```
   ~]$ ps -eZ | grep mysqld
   system_u:system_r:mysqld_safe_t:s0 12831 ?     00:00:00 mysqld_safe
   system_u:system_r:mysqld_t:s0   13014 ?        00:00:00 mysqld
   ```

   The SELinux context associated with the **mysqld** processes is **system_u:system_r:mysqld_t:s0**. The second last part of the context, **mysqld_t**, is the type. A type defines a domain for processes and a type for files. In this case, the **mysqld** processes are running in the **mysqld_t** domain.

20.2. Types

Type Enforcement is the main permission control used in SELinux targeted policy. All files and processes are labeled with a type: types define a domain for processes and a type for files. SELinux policy rules define how types access each other, whether it be a domain accessing a type, or a domain accessing another domain. Access is only allowed if a specific SELinux policy rule exists that allows it.

The following types are used with **`mysqld`**. Different types allow you to configure flexible access:

`mysqld_db_t`

> This type is used for the location of the MariaDB database. In Red Hat Enterprise Linux, the default location for the database is the **`/var/lib/mysql/`** directory, however this can be changed. If the location for the MariaDB database is changed, the new location must be labeled with this type. Refer to the example in Section 20.4.1, “MariaDB Changing Database Location” for instructions on how to change the default database location and how to label the new section appropriately.

`mysqld_etc_t`

> This type is used for the MariaDB main configuration file **`/etc/my.cnf`** and any other configuration files in the **`/etc/mysql/`** directory.

`mysqld_exec_t`

> This type is used for the **`mysqld`** binary located at **`/usr/libexec/mysqld`**, which is the default location for the MariaDB binary on Red Hat Enterprise Linux. Other systems may locate this binary at **`/usr/sbin/mysqld`** which should also be labeled with this type.

`mysqld_unit_file_t`

> This type is used for executable MariaDB-related files located in the **`/usr/lib/systemd/system/`** directory by default in Red Hat Enterprise Linux.

`mysqld_log_t`

> Logs for MariaDB need to be labeled with this type for proper operation. All log files in the **`/var/log/`** directory matching the **`mysql.*`** wildcard must be labeled with this type.

`mysqld_var_run_t`

> This type is used by files in the **`/var/run/mariadb/`** directory, specifically the process id (PID) named **`/var/run/mariadb/mariadb.pid`** which is created by the **`mysqld`** daemon when it runs. This type is also used for related socket files such as **`/var/lib/mysql/mysql.sock`**. Files such as these must be labeled correctly for proper operation as a confined service.

20.3. Booleans

SELinux is based on the least level of access required for a service to run. Services can be run in a variety of ways; therefore, you need to specify how you run your services. Use the following Booleans to set up SELinux:

`selinuxuser_mysql_connect_enabled`

> When enabled, this Boolean allows users to connect to the local MariaDB server.

`exim_can_connect_db`

When enabled, this Boolean allows the **`exim`** mailer to initiate connections to a database server.

`ftpd_connect_db`

When enabled, this Boolean allows **`ftp`** daemons to initiate connections to a database server.

`httpd_can_network_connect_db`

Enabling this Boolean is required for a web server to communicate with a database server.

Note

Due to the continuous development of the SELinux policy, the list above might not contain all Booleans related to the service at all times. To list them, run the following command:

```
~]$ getsebool -a | grep service_name
```

Run the following command to view description of a particular Boolean:

```
~]$ sepolicy booleans -b boolean_name
```

Note that the additional *policycoreutils-devel* package providing the **`sepolicy`** utility is required.

20.4. Configuration Examples

20.4.1. MariaDB Changing Database Location

When using Red Hat Enterprise Linux, the default location for MariaDB to store its database is **`/var/lib/mysql/`**. This is where SELinux expects it to be by default, and hence this area is already labeled appropriately for you, using the **`mysqld_db_t`** type.

The location where the database is stored can be changed depending on individual environment requirements or preferences, however it is important that SELinux is aware of this new location; that it is labeled accordingly. This example explains how to change the location of a MariaDB database and then how to label the new location so that SELinux can still provide its protection mechanisms to the new area based on its contents.

Note that this is an example only and demonstrates how SELinux can affect MariaDB. Comprehensive documentation of MariaDB is beyond the scope of this document. Refer to the official MariaDB documentation for further details. This example assumes that the *mariadb-server* and *setroubleshoot-server* packages are installed, that the **`auditd`** service is running, and that there is a valid database in the default location of **`/var/lib/mysql/`**.

1. View the SELinux context of the default database location for **`mysql`**:

```
~]# ls -lZ /var/lib/mysql
drwx------. mysql mysql system_u:object_r:mysqld_db_t:s0 mysql
```

This shows **mysqld_db_t** which is the default context element for the location of database files. This context will have to be manually applied to the new database location that will be used in this example in order for it to function properly.

2. Run the following command and enter the **mysqld** root password to show the available databases:

```
~]# mysqlshow -u root -p
Enter password: *******
+--------------------+
|     Databases      |
+--------------------+
| information_schema |
| mysql              |
| test               |
| wikidb             |
+--------------------+
```

3. Stop the **mysqld** daemon:

```
~]# systemctl stop mariadb.service
```

4. Create a new directory for the new location of the database(s). In this example, **/mysql/** is used:

```
~]# mkdir -p /mysql
```

5. Copy the database files from the old location to the new location:

```
~]# cp -R /var/lib/mysql/* /mysql/
```

6. Change the ownership of this location to allow access by the mysql user and group. This sets the traditional Unix permissions which SELinux will still observe:

```
~]# chown -R mysql:mysql /mysql
```

7. Run the following command to see the initial context of the new directory:

```
~]# ls -lZ /mysql
drwxr-xr-x. mysql mysql unconfined_u:object_r:usr_t:s0   mysql
```

The context **usr_t** of this newly created directory is not currently suitable to SELinux as a location for MariaDB database files. Once the context has been changed, MariaDB will be able to function properly in this area.

8. Open the main MariaDB configuration file **/etc/my.cnf** with a text editor and modify the **datadir** option so that it refers to the new location. In this example the value that should be entered is **/mysql**:

```
[mysqld]
datadir=/mysql
```

Save this file and exit.

9. Start **mysqld**. The service should fail to start, and a denial message will be logged to the **/var/log/messages** file:

```
~]# systemctl start mariadb.service
Job for mariadb.service failed. See 'systemctl status postgresql.service' and
'journalctl -xn' for details.
```

However, if the **audit** daemon is running and with him the **setroubleshoot** service, the denial will be logged to the **/var/log/audit/audit.log** file instead:

```
SELinux is preventing /usr/libexec/mysqld "write" access on /mysql. For
complete SELinux messages. run sealert -l b3f01aff-7fa6-4ebe-ad46-abaef6f8ad71
```

The reason for this denial is that **/mysql/** is not labeled correctly for MariaDB data files. SELinux is stopping MariaDB from having access to the content labeled as **usr_t**. Perform the following steps to resolve this problem:

10. Run the following command to add a context mapping for **/mysql/**. Note that the **semanage** utility is not installed by default. If it missing on your system, install the *policycoreutils-python* package.

    ```
    ~]# semanage fcontext -a -t mysqld_db_t "/mysql(/.*)?"
    ```

11. This mapping is written to the **/etc/selinux/targeted/contexts/files/file_contexts.local** file:

    ```
    ~]# grep -i mysql /etc/selinux/targeted/contexts/files/file_contexts.local

    /mysql(/.*)?    system_u:object_r:mysqld_db_t:s0
    ```

12. Now use the **restorecon** utility to apply this context mapping to the running system:

    ```
    ~]# restorecon -R -v /mysql
    ```

13. Now that the **/mysql/** location has been labeled with the correct context for MariaDB, **mysqld** starts:

    ```
    ~]# systemctl start mariadb.service
    ```

14. Confirm the context has changed for **/mysql/**:

    ```
    ~]$ ls -lZ /mysql
    drwxr-xr-x. mysql mysql system_u:object_r:mysqld_db_t:s0 mysql
    ```

15. The location has been changed and labeled, and **mysqld** has started successfully. At this point all running services should be tested to confirm normal operation.

[22] Refer to the MariaDB project page for more information.

Chapter 21. PostgreSQL

PostgreSQL is an Object-Relational database management system (DBMS). [23]

In Red Hat Enterprise Linux, the *postgresql-server* package provides PostgreSQL. Run the following command to see if the *postgresql-server* package is installed:

```
~]# rpm -q postgresql-server
```

If it is not installed, use the **yum** utility as root to install it:

```
~]# yum install postgresql-server
```

21.1. PostgreSQL and SELinux

When PostgreSQL is enabled, it runs confined by default. Confined processes run in their own domains, and are separated from other confined processes. If a confined process is compromised by an attacker, depending on SELinux policy configuration, an attacker's access to resources and the possible damage they can do is limited. The following example demonstrates the PostgreSQL processes running in their own domain. This example assumes the *postgresql-server* package is installed:

1. Run the **getenforce** command to confirm SELinux is running in enforcing mode:

   ```
   ~]$ getenforce
   Enforcing
   ```

   The command returns **Enforcing** when SELinux is running in enforcing mode.

2. Run the following command as the root user to start **postgresql**:

   ```
   ~]# systemctl start postgresql.service
   ```

   Confirm that the service is running. The output should include the information below (only the time stamp will differ):

   ```
   ~]# systemctl start postgresql.service
   postgresql.service - PostgreSQL database server
      Loaded: loaded (/usr/lib/systemd/system/postgresql.service; disabled)
      Active: active (running) since Mon 2013-08-05 14:57:49 CEST; 12s
   ```

3. Run the following command to view the **postgresql** processes:

   ```
   ~]$ ps -eZ | grep postgres
   system_u:system_r:postgresql_t:s0 395 ?    00:00:00 postmaster
   system_u:system_r:postgresql_t:s0 397 ?    00:00:00 postmaster
   system_u:system_r:postgresql_t:s0 399 ?    00:00:00 postmaster
   system_u:system_r:postgresql_t:s0 400 ?    00:00:00 postmaster
   system_u:system_r:postgresql_t:s0 401 ?    00:00:00 postmaster
   system_u:system_r:postgresql_t:s0 402 ?    00:00:00 postmaster
   ```

   The SELinux context associated with the **postgresql** processes is **system_u:system_r:postgresql_t:s0**. The second last part of the context, **postgresql_t**, is the type. A type defines a domain for processes and a type for files. In this case, the **postgresql** processes are running in the **postgresql_t** domain.

21.2. Types

Type Enforcement is the main permission control used in SELinux targeted policy. All files and processes are labeled with a type: types define a domain for processes and a type for files. SELinux policy rules define how types access each other, whether it be a domain accessing a type, or a domain accessing another domain. Access is only allowed if a specific SELinux policy rule exists that allows it.

The following types are used with **`postgresql`**. Different types allow you to configure flexible access. Note that in the list below are used several regular expression to match the whole possible locations:

`postgresql_db_t`

This type is used for several locations. The locations labeled with this type are used for data files for PostgreSQL:

- **`/usr/lib/pgsql/test/regres`**
- **`/usr/share/jonas/pgsql`**
- **`/var/lib/pgsql/data`**
- **`/var/lib/postgres(ql)?`**

`postgresql_etc_t`

This type is used for configuration files in the **`/etc/postgresql/`** directory.

`postgresql_exec_t`

This type is used for several locations. The locations labeled with this type are used for binaries for PostgreSQL:

- **`/usr/bin/initdb(.sepgsql)?`**
- **`/usr/bin/(se)?postgres`**
- **`/usr/lib(64)?/postgresql/bin/.*`**
- **`/usr/lib(64)?/pgsql/test/regress/pg_regress`**

`systemd_unit_file_t`

This type is used for the executable PostgreSQL-related files located in the **`/usr/lib/systemd/system/`** directory.

`postgresql_log_t`

This type is used for several locations. The locations labeled with this type are used for log files:

- **`/var/lib/pgsql/logfile`**
- **`/var/lib/pgsql/pgstartup.log`**
- **`/var/lib/sepgsql/pgstartup.log`**
- **`/var/log/postgresql`**

- **/var/log/postgres.log.***
- **/var/log/rhdb/rhdb**
- **/var/log/sepostgresql.log.***

postgresql_var_run_t

This type is used for run-time files for PostgreSQL, such as the process id (PID) in the **/var/run/postgresql/** directory.

21.3. Booleans

SELinux is based on the least level of access required for a service to run. Services can be run in a variety of ways; therefore, you need to specify how you run your services. Use the following Booleans to set up SELinux:

selinuxuser_postgresql_connect_enabled

Having this Boolean enabled allows any user domain (as defined by PostgreSQL) to make connections to the database server.

Note

Due to the continuous development of the SELinux policy, the list above might not contain all Booleans related to the service at all times. To list them, run the following command:

```
~]$ getsebool -a | grep service_name
```

Run the following command to view description of a particular Boolean:

```
~]$ sepolicy booleans -b boolean_name
```

Note that the additional *policycoreutils-devel* package providing the **sepolicy** utility is required.

21.4. Configuration Examples

21.4.1. PostgreSQL Changing Database Location

When using Red Hat Enterprise Linux, the default location for PostgreSQL to store its database is **/var/lib/pgsql/data/**. This is where SELinux expects it to be by default, and hence this area is already labeled appropriately for you, using the **postgresql_db_t** type.

The area where the database is located can be changed depending on individual environment requirements or preferences, however it is important that SELinux is aware of this new location; that it is labeled accordingly. This example explains how to change the location of a PostgreSQL database and then how to label the new location so that SELinux can still provide its protection mechanisms to the new area based on its contents.

Note that this is an example only and demonstrates how SELinux can affect PostgreSQL. Comprehensive documentation of PostgreSQL is beyond the scope of this document. Refer to the official PostgreSQL documentation for further details. This example assumes that the *postgresql-server* package is installed.

1. View the SELinux context of the default database location for **postgresql**:

   ```
   ~]# ls -lZ /var/lib/pgsql
   drwx------. postgres postgres system_u:object_r:postgresql_db_t:s0 data
   ```

   This shows **postgresql_db_t** which is the default context element for the location of database files. This context will have to be manually applied to the new database location that will be used in this example in order for it to function properly.

2. Create a new directory for the new location of the database(s). In this example, **/opt/postgresql/data/** is used. If you use a different location, replace the text in the following steps with your location:

   ```
   ~]# mkdir -p /opt/postgresql/data
   ```

3. Perform a directory listing of the new location. Note that the initial context of the new directory is **usr_t**. This context is not sufficient for SELinux to offer its protection mechanisms to PostgreSQL. Once the context has been changed, it will be able to function properly in the new area.

   ```
   ~]# ls -lZ /opt/postgresql/
   drwxr-xr-x. root root unconfined_u:object_r:usr_t:s0   data
   ```

4. Change the ownership of the new location to allow access by the postgres user and group. This sets the traditional Unix permissions which SELinux will still observe.

   ```
   ~]# chown -R postgres:postgres /opt/postgresql
   ```

5. Open the PostgreSQL init file **/etc/rc.d/init.d/postgresql** with a text editor and modify the **PGDATA** and **PGLOG** variables to point to the new location:

   ```
   ~]# vi /etc/rc.d/init.d/postgresql
   PGDATA=/opt/postgresql/data
   PGLOG=/opt/postgresql/data/pgstartup.log
   ```

   Save this file and exit the text editor.

6. Initialize the database in the new location:

   ```
   ~]$ su - postgres -c "initdb -D /opt/postgresql/data"
   ```

7. Having changed the database location, starting the service will fail at this point:

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

   SELinux has caused the service to not start. This is because the new location is not properly labeled. The following steps explain how to label the new location (**/opt/postgresql/**) and start the postgresql service properly:

8. Use the **semanage** utility to add a context mapping for **/opt/postgresql/** and any other directories/files within it:

```
~]# semanage fcontext -a -t postgresql_db_t "/opt/postgresql(/.*)?"
```

9. This mapping is written to the **/etc/selinux/targeted/contexts/files/file_contexts.local** file:

```
~]# grep -i postgresql
/etc/selinux/targeted/contexts/files/file_contexts.local

/opt/postgresql(/.*)?    system_u:object_r:postgresql_db_t:s0
```

10. Now use the **restorecon** utility to apply this context mapping to the running system:

```
~]# restorecon -R -v /opt/postgresql
```

11. Now that the **/opt/postgresql/** location has been labeled with the correct context for PostgreSQL, the **postgresql** service will start successfully:

```
~]# systemctl start postgresql.service
```

12. Confirm the context is correct for **/opt/postgresql/**:

```
~]$ ls -lZ /opt
drwxr-xr-x. root root system_u:object_r:postgresql_db_t:s0 postgresql
```

13. Check with the **ps** command that the **postgresql** process displays the new location:

```
~]# ps aux | grep -i postmaster

postgres 21564  0.3  0.3  42308  4032 ?        S    10:13   0:00
/usr/bin/postmaster -p 5432 -D /opt/postgresql/data/
```

14. The location has been changed and labeled, and **postgresql** has started successfully. At this point all running services should be tested to confirm normal operation.

[23] Refer to the PostgreSQL project page for more information.

Chapter 22. rsync

The **rsync** utility performs fast file transfer and it is used for synchronizing data between systems. [24]

When using Red Hat Enterprise Linux, the *rsync* package provides rsync. Run the following command to see if the *rsync* package is installed:

```
~]$ rpm -q rsync
package rsync is not installed
```

If it is not installed, use the **yum** utility as root to install it:

```
~]# yum install rsync
```

22.1. rsync and SELinux

SELinux requires files to have an extended attribute to define the file type. Policy governs the access daemons have to these files. If you want to share files using the **rsync** daemon, you must label the files and directories with the **public_content_t** type. Like most services, correct labeling is required for SELinux to perform its protection mechanisms over **rsync**. [25]

22.2. Types

Type Enforcement is the main permission control used in SELinux targeted policy. All files and processes are labeled with a type: types define a domain for processes and a type for files. SELinux policy rules define how types access each other, whether it be a domain accessing a type, or a domain accessing another domain. Access is only allowed if a specific SELinux policy rule exists that allows it.

The following types are used with **rsync**. Different types all you to configure flexible access:

public_content_t

This is a generic type used for the location of files (and the actual files) to be shared using **rsync**. If a special directory is created to house files to be shared with **rsync**, the directory and its contents need to have this label applied to them.

rsync_exec_t

This type is used for the **/usr/bin/rsync** system binary.

rsync_log_t

This type is used for the **rsync** log file, located at **/var/log/rsync.log** by default. To change the location of the file rsync logs to, use the **--log-file=FILE** option to the **rsync** command at run-time.

rsync_var_run_t

This type is used for the **rsyncd** lock file, located at **/var/run/rsyncd.lock**. This lock file is used by the **rsync** server to manage connection limits.

rsync_data_t

This type is used for files and directories which you want to use as rsync domains and isolate them from the access scope of other services. Also, the **`public_content_t`** is a general SELinux context type, which can be used when a file or a directory interacts with multiple services (for example, FTP and NFS directory as an rsync domain).

`rsync_etc_t`

This type is used for rsync-related files in the **`/etc/`** directory.

22.3. Booleans

SELinux is based on the least level of access required for a service to run. Services can be run in a variety of ways; therefore, you need to specify how you run your services. Use the following Booleans to set up SELinux:

`rsync_anon_write`

Having this Boolean enabled allows **`rsync`** in the **`rsync_t`** domain to manage files, links and directories that have a type of **`public_content_rw_t`**. Often these are public files used for public file transfer services. Files and directories must be labeled this type.

`rsync_client`

Having this Boolean enabled allows **`rsync`** to initiate connections to ports defined as **`rsync_port_t`**, as well as allowing the daemon to manage files, links, and directories that have a type of **`rsync_data_t`**. Note that **`rsync`** must be in the **`rsync_t`** domain in order for SELinux to enact its control over it. The configuration example in this chapter demonstrates **`rsync`** running in the **`rsync_t`** domain.

`rsync_export_all_ro`

Having this Boolean enabled allows **`rsync`** in the **`rsync_t`** domain to export NFS and CIFS volumes with read-only access to clients.

Note

Due to the continuous development of the SELinux policy, the list above might not contain all Booleans related to the service at all times. To list them, run the following command:

```
~]$ getsebool -a | grep service_name
```

Run the following command to view description of a particular Boolean:

```
~]$ sepolicy booleans -b boolean_name
```

Note that the additional *policycoreutils-devel* package providing the **`sepolicy`** utility is required.

22.4. Configuration Examples

22.4.1. Rsync as a daemon

When using Red Hat Enterprise Linux, rsync can be used as a daemon so that multiple clients can directly communicate with it as a central server, in order to house centralized files and keep them synchronized. The following example will demonstrate running rsync as a daemon over a network socket in the correct domain, and how SELinux expects this daemon to be running on a pre-defined (in SELinux policy) TCP port. This example will then show how to modify SELinux policy to allow the **rsync** daemon to run normally on a non-standard port.

This example will be performed on a single system to demonstrate SELinux policy and its control over local daemons and processes. Note that this is an example only and demonstrates how SELinux can affect rsync. Comprehensive documentation of rsync is beyond the scope of this document. Refer to the official rsync documentation for further details. This example assumes that the *rsync*, *setroubleshoot-server* and *audit* packages are installed, that the SELinux targeted policy is used and that SELinux is running in enforcing mode.

Procedure 22.1. Getting rsync to launch as rsync_t

1. Run the **getenforce** command to confirm SELinux is running in enforcing mode:

   ```
   ~]$ getenforce
   Enforcing
   ```

   The command returns **Enforcing** when SELinux is running in enforcing mode.

2. Run the **which** command to confirm that the rsync binary is in the system path:

   ```
   ~]$ which rsync
   /usr/bin/rsync
   ```

3. When running rsync as a daemon, a configuration file should be used and saved as **/etc/rsyncd.conf**. Note that the following configuration file used in this example is very simple and is not indicative of all the possible options that are available, rather it is just enough to demonstrate the **rsync** daemon:

   ```
   log file = /var/log/rsync.log
   pid file = /var/run/rsyncd.pid
   lock file = /var/run/rsync.lock
   [files]
    path = /srv/rsync
           comment = file area
           read only = false
           timeout = 300
   ```

4. Now that a simple configuration file exists for rsync to operate in daemon mode, you can start it by running the following command:

   ```
   ~]# systemctl start rsyncd.service
   ```

   Ensure that **rsyncd** was successfully started (the output is supposed to look similar to the one below, only the time stamp will differ):

   ```
   ~]# systemctl status rsyncd.service
   rsyncd.service - fast remote file copy program daemon
      Loaded: loaded (/usr/lib/systemd/system/rsyncd.service; disabled)
      Active: active (running) since Thu 2014-02-27 09:46:24 CET; 2s ago
    Main PID: 3220 (rsync)
      CGroup: /system.slice/rsyncd.service
              └─3220 /usr/bin/rsync --daemon --no-detach
   ```

SELinux can now enforce its protection mechanisms over the **rsync** daemon as it is now running in the **rsync_t** domain:

```
~]$ ps -eZ | grep rsync
system_u:system_r:rsync_t:s0     3220 ?        00:00:00 rsync
```

This example demonstrated how to get **rsyncd** running in the **rsync_t** domain. The next example shows how to get this daemon successfully running on a non-default port. TCP port 10000 is used in the next example.

Procedure 22.2. Running the rsync daemon on a non-default port

1. Modify the **/etc/rsyncd.conf** file and add the **port = 10000** line at the top of the file in the global configuration area (that is, before any file areas are defined). The new configuration file will look like:

```
log file = /var/log/rsyncd.log
pid file = /var/run/rsyncd.pid
lock file = /var/run/rsync.lock
port = 10000
[files]
        path = /srv/rsync
        comment = file area
        read only = false
 timeout = 300
```

2. After launching the **rsync** daemon with this new setting, a denial message similar to the following is logged by SELinux:

```
Jul 22 10:46:59 localhost setroubleshoot: SELinux is preventing the rsync
(rsync_t) from binding to port 10000. For complete SELinux messages, run
sealert -l c371ab34-639e-45ae-9e42-18855b5c2de8
```

3. Use the **semanage** utility to add TCP port 10000 to the SELinux policy in **rsync_port_t**:

```
~]# semanage port -a -t rsync_port_t -p tcp 10000
```

4. Now that TCP port 10000 has been added to the SELinux policy for **rsync_port_t**, **rsyncd** will start and operate normally on this port:

```
~]# systemctl start rsyncd.service
```

```
~]# netstat -lnp | grep 10000
tcp        0      0 0.0.0.0:10000   0.0.0.0:*      LISTEN      9910/rsync
```

SELinux has had its policy modified and is now permitting **rsyncd** to operate on TCP port 10000.

[24] Refer to the Rsync project page for more information.

[25] Refer to the rsync_selinux(8) manual page for more information about rsync and SELinux.

Chapter 23. Postfix

Postfix is an open-source Mail Transport Agent (MTA), which supports protocols like LDAP, SMTP AUTH (SASL), and TLS. [26]

In Red Hat Enterprise Linux, the *postfix* package provides Postfix. Run the following command to see if the *postfix* package is installed:

```
~]$ rpm -q postfix
package postfix is not installed
```

If it is not installed, use the **yum** utility root to install it:

```
~]# yum install postfix
```

23.1. Postfix and SELinux

When Postfix is enabled, it runs confined by default. Confined processes run in their own domains, and are separated from other confined processes. If a confined process is compromised by an attacker, depending on SELinux policy configuration, an attacker's access to resources and the possible damage they can do is limited. The following example demonstrates the Postfix and related processes running in their own domain. This example assumes the *postfix* package is installed and that the Postfix service has been started:

1. Run the **getenforce** command to confirm SELinux is running in enforcing mode:

   ```
   ~]$ getenforce
   Enforcing
   ```

   The command returns **Enforcing** when SELinux is running in enforcing mode.

2. Run the following command as the root user to start **postfix**:

   ```
   ~]# systemctl start postfix.service
   ```

   Confirm that the service is running. The output should include the information below (only the time stamp will differ):

   ```
   ~]# systemctl status postfix.service
   postfix.service - Postfix Mail Transport Agent
      Loaded: loaded (/usr/lib/systemd/system/postfix.service; disabled)
      Active: active (running) since Mon 2013-08-05 11:38:48 CEST; 3h 25min ago
   ```

3. Run following command to view the **postfix** processes:

   ```
   ~]$ ps -eZ | grep postfix
   system_u:system_r:postfix_master_t:s0 1651 ?   00:00:00 master
   system_u:system_r:postfix_pickup_t:s0 1662 ?   00:00:00 pickup
   system_u:system_r:postfix_qmgr_t:s0 1663 ?     00:00:00 qmgr
   ```

   In the output above, the SELinux context associated with the Postfix **master** process is **system_u:system_r:postfix_master_t:s0**. The second last part of the context, **postfix_master_t**, is the type for this process. A type defines a domain for processes and a type for files. In this case, the **master** process is running in the **postfix_master_t** domain.

23.2. Types

Type Enforcement is the main permission control used in SELinux targeted policy. All files and processes are labeled with a type: types define a domain for processes and a type for files. SELinux policy rules define how types access each other, whether it be a domain accessing a type, or a domain accessing another domain. Access is only allowed if a specific SELinux policy rule exists that allows it.

The following types are used with Postfix. Different types all you to configure flexible access:

`postfix_etc_t`

This type is used for configuration files for Postfix in the **`/etc/postfix/`** directory.

`postfix_data_t`

This type is used for Postfix data files in the **`/var/lib/postfix/`** directory.

`postfix_var_run_t`

This type is used for Postfix files stored in the **`/run/`** directory.

`postfix_initrc_exec_t`

The Postfix executable files are labeled with the **`postfix_initrc_exec_t`** type. When executed, they transition to the **`postfix_initrc_t`** domain.

`postfix_spool_t`

This type is used for Postfix files stored in the **`/var/spool/`** directory.

Note

To see the full list of files and their types for Postfix, run the following command:

```
~]$ grep postfix /etc/selinux/targeted/contexts/files/file_contexts
```

23.3. Booleans

SELinux is based on the least level of access required for a service to run. Services can be run in a variety of ways; therefore, you need to specify how you run your services. Use the following Booleans to set up SELinux:

`postfix_local_write_mail_spool`

Having this Boolean enabled allows Postfix to write to the local mail spool on the system. Postfix requires this Boolean to be enabled for normal operation when local spools are used.

Note

Due to the continuous development of the SELinux policy, the list above might not contain all Booleans related to the service at all times. To list them, run the following command:

```
~]$ getsebool -a | grep service_name
```

Run the following command to view description of a particular Boolean:

```
~]$ sepolicy booleans -b boolean_name
```

Note that the additional *policycoreutils-devel* package providing the **sepolicy** utility is required.

23.4. Configuration Examples

23.4.1. SpamAssassin and Postfix

SpamAssasin is an open-source mail filter that provides a way to filter unsolicited email (spam messages) from incoming email. [27]

When using Red Hat Enterprise Linux, the *spamassassin* package provides SpamAssassin. Run the following command to see if the *spamassassin* package is installed:

```
~]$ rpm -q spamassassin
package spamassassin is not installed
```

If it is not installed, use the **yum** utility as root to install it:

```
~]# yum install spamassassin
```

SpamAssassin operates in tandem with a mailer such as Postfix to provide spam-filtering capabilities. In order for SpamAssassin to effectively intercept, analyze and filter mail, it must listen on a network interface. The default port for SpamAssassin is TCP/783, however this can be changed. The following example provides a real-world demonstration of how SELinux complements SpamAssassin by only allowing it access to a certain port by default. This example will then demonstrate how to change the port and have SpamAssassin operate on a non-default port.

Note that this is an example only and demonstrates how SELinux can affect a simple configuration of SpamAssassin. Comprehensive documentation of SpamAssassin is beyond the scope of this document. Refer to the official SpamAssassin documentation for further details. This example assumes the *spamassassin* is installed, that any firewall has been configured to allow access on the ports in use, that the SELinux targeted policy is used, and that SELinux is running in enforcing mode:

Procedure 23.1. Running SpamAssassin on a non-default port

1. Use the **semanage** utility as root to show the port that SELinux allows the **spamd** daemon to listen on by default:

   ```
   ~]# semanage port -l | grep spamd
   spamd_port_t  tcp 783
   ```

This output shows that TCP/783 is defined in **spamd_port_t** as the port for SpamAssassin to operate on.

2. Edit the **/etc/sysconfig/spamassassin** configuration file and modify it so that it will start SpamAssassin on the example port TCP/10000:

   ```
   # Options to spamd
   SPAMDOPTIONS="-d -p 10000 -c m5 -H"
   ```

   This line now specifies that SpamAssassin will operate on port 10000. The rest of this example will show how to modify the SELinux policy to allow this socket to be opened.

3. Start SpamAssassin and an error message similar to the following will appear:

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

   This output means that SELinux has blocked access to this port.

4. A denial message similar to the following will be logged by SELinux:

   ```
   SELinux is preventing the spamd (spamd_t) from binding to port 10000.
   ```

5. As root, run **semanage** to modify the SELinux policy in order to allow SpamAssassin to operate on the example port (TCP/10000):

   ```
   ~]# semanage port -a -t spamd_port_t -p tcp 10000
   ```

6. Confirm that SpamAssassin will now start and is operating on TCP port 10000:

   ```
   ~]# systemctl start spamassassin.service

   ~]# netstat -lnp | grep 10000
   tcp 0 0 127.0.0.1:10000 0.0.0.0:* LISTEN 2224/spamd.pid
   ```

7. At this point, **spamd** is properly operating on TCP port 10000 as it has been allowed access to that port by the SELinux policy.

[26] For more information, see the appropriate chapter in the System Administrator's Guide.

[27] For more information, see the appropriate chapter in the System Administrator's Guide.

Chapter 24. DHCP

The **dhcpd** daemon is used in Red Hat Enterprise Linux to dynamically deliver and configure Layer 3 TCP/IP details for clients.

The *dhcp* package provides the DHCP server and the **dhcpd** daemon. Run the following command to see if the *dhcp* package is installed:

```
~]# rpm -q dhcp
package dhcp is not installed
```

If it is not installed, use the **yum** utility as root to install it:

```
~]# yum install dhcp
```

24.1. DHCP and SELinux

When **dhcpd** is enabled, it runs confined by default. Confined processes run in their own domains, and are separated from other confined processes. If a confined process is compromised by an attacker, depending on SELinux policy configuration, an attacker's access to resources and the possible damage they can do is limited. The following example demonstrates **dhcpd** and related processes running in their own domain. This example assumes the *dhcp* package is installed and that the **dhcpd** service has been started:

1. Run the **getenforce** command to confirm SELinux is running in enforcing mode:

   ```
   ~]$ getenforce
   Enforcing
   ```

   The command returns **Enforcing** when SELinux is running in enforcing mode.

2. Run the following command as the root user to start **dhcpd**:

   ```
   ~]# systemctl start dhcpd.service
   ```

   Confirm that the service is running. The output should include the information below (only the time stamp will differ):

   ```
   ~]# systemctl status dhcpd.service
   dhcpd.service - DHCPv4 Server Daemon
      Loaded: loaded (/usr/lib/systemd/system/dhcpd.service; disabled)
      Active: active (running) since Mon 2013-08-05 11:49:07 CEST; 3h 20min ago
   ```

3. Run following command to view the **dhcpd** processes:

   ```
   ~]$ ps -eZ | grep dhcpd
   system_u:system_r:dhcpd_t:s0 5483 ?        00:00:00 dhcpd
   ```

   The SELinux context associated with the dhcpd process is **system_u:system_r:dhcpd_t:s0**.

24.2. Types

The following types are used with DHCP:

`dhcp_etc_t`

This type is mainly used for files in the **`/etc/`** directory, including configuration files.

`dhcpd_var_run_t`

This type is used for the PID file for **`dhcpd`**, in the **`/var/run/`** directory.

`dhcpd_exec_t`

This type is used for transition of DHCP executable files to the **`dhcpd_t`** domain.

`dhcpd_initrc_exec_t`

This type is used for transition of DHCP executable files to the **`dhcpd_initrc_t`** domain.

Note

To see the full list of files and their types for **`dhcpd`**, run the following command:

```
~]$ grep dhcp /etc/selinux/targeted/contexts/files/file_contexts
```

Chapter 25. OpenShift

OpenShift is a Platform as a Service (PaaS) that enables developers to build and deploy web applications. OpenShift provides a wide selection of programming languages and frameworks including Java, Ruby, and PHP. It also provides integrated developer tools to support the application life cycle, including Eclipse integration, JBoss Developer Studio, and Jenkins. OpenShift uses an open source ecosystem to provide a platform for mobile applications, database services, and more. [28]

In Red Hat Enterprise Linux, the *rhc* package provides the OpenShift client tools. Run the following command to see if it is installed:

```
~]$ rpm -q rhc
package rhc is not installed
```

If *rhc* is not installed, see the OpenShift Enterprise Client Tools Installation Guide and OpenShift Online Client Tools Installation Guide for detailed information on the OpenShift client tools installation process.

25.1. OpenShift and SELinux

SELinux provides better security control over applications that use OpenShift because all processes are labeled according to the SELinux policy. Therefore, SELinux protects OpenShift from possible malicious attacks within different gears running on the same node.

See Dan Walsh's presentation for more information about SELinux and OpenShift.

25.2. Types

Type Enforcement is the main permission control used in SELinux targeted policy. All files and processes are labeled with a type: types define a domain for processes and a type for files. SELinux policy rules define how types access each other, whether it be a domain accessing a type, or a domain accessing another domain. Access is only allowed if a specific SELinux policy rule exists that allows it.

The following types are used with OpenShift. Different types allow you configure flexible access:

Process types

`openshift_t`

The OpenShift process is associated with the **`openshift_t`** SELinux type.

`openshift_mail_t`

This type allows the OpenShift process to execute a mail program and it is used to control the amount of data that can be sent by a gear. Note that this type is used only by OpenShift Online.

Types on executables

`openshift_cgroup_read_exec_t`

SELinux allows files with this type to transition an executable to the **`openshift_cgroup_read_t`** domain.

`openshift_cron_exec_t`

SELinux allows files with this type to transition an executable to the **`openshift_cron_t`** domain.

`openshift_initrc_exec_t`

SELinux allows files with this type to transition an executable to the **`openshift_initrc_t`** domain.

Writable types

`openshift_cgroup_read_tmp_t`

This type allows OpenShift control groups (cgroup) read and access temporary files in the **`/tmp/`** directory.

`openshift_cron_tmp_t`

This type allows storing temporary files of the OpenShift cron jobs in **`/tmp/`**.

`openshift_initrc_tmp_t`

This type allows storing the OpenShift **`initrc`** temporary files in **`/tmp/`**.

`openshift_log_t`

Files with this type are treated as OpenShift log data, usually stored under the **`/var/log/`** directory.

`openshift_rw_file_t`

OpenShift have permission to read and to write to files labeled with this type.

`openshift_tmp_t`

This type is used for storing the OpenShift temporary files in **`/tmp/`**.

`openshift_tmpfs_t`

This type allows storing the OpenShift data on a tmpfs file system.

`openshift_var_lib_t`

This type allows storing the OpenShift files in the **`/var/lib/`** directory.

`openshift_var_run_t`

This type allows storing the OpenShift files in the **`/run/`** or **`/var/run/`** directory.

25.3. Booleans

SELinux is based on the least level of access required for a service to run. Services can be run in a variety of ways; therefore, you need to specify how you run your services. Use the following Booleans to set up SELinux:

`openshift_use_nfs`

Having this Boolean enabled allows installing OpenShift on an NFS share.

Note

Due to the continuous development of the SELinux policy, the list above might not contain all Booleans related to the service at all times. To list them, run the following command:

```
~]$ getsebool -a | grep service_name
```

Run the following command to view description of a particular Boolean:

```
~]$ sepolicy booleans -b boolean_name
```

Note that the additional *policycoreutils-devel* package providing the **sepolicy** utility is required.

25.4. Configuration Examples

25.4.1. Changing the Default OpenShift Directory

By default, OpenShift stores its data in the **/var/lib/openshift/** directory, which is labeled with the **openshift_var_lib_t** SELinux type. To allow OpenShift to store data in a different directory, label the new directory with the proper SELinux context.

The following procedure shows how to change the default OpenShift directory for storing data to **/srv/openshift/**:

Procedure 25.1. Changing the Default OpenShift Directory for Storing Data

1. As root, create a new **/openshift/** directory within the **/srv/** directory. The new directory is labeled with the **var_t** type:

```
~]# mkdir /srv/openshift
```

```
~]$ ls -Zd /srv/openshift
drwxr-xr-x. root root unconfined_u:object_r:var_t:s0   openshift/
```

2. As root, use the **semanage** utility to map **/srv/openshift/** to the proper SELinux context:

```
~]# semanage fcontext -a -e /var/lib/openshift /srv/openshift
```

3. Then, use the **restorecon** utility as root to apply the changes:

```
~]# restorecon -R -v /srv/openshift
```

4. The **/srv/openshift/** directory is now labeled with the correct **openshift_var_lib_t** type:

```
~]$ls -Zd /srv/openshift
drwxr-xr-x. root root unconfined_u:object_r:openshift_var_lib_t:s0
openshift/
```

[28] To learn more about OpenShift, see OpenShift Enterprise documentation and OpenShift Online documentation.

Chapter 26. Identity Management

Identity Management (IdM) provides a unifying environment for standards-defined, common network services, including PAM, LDAP, Kerberos, DNS, NTP, and certificate services. IdM allows Red Hat Enterprise Linux systems to serve as domain controllers. [29]

In Red Hat Enterprise Linux, the *ipa-server* package provides the IdM server. Run the following command to see if the *ipa-server* package is installed:

```
~]$ rpm -q ipa-server
package ipa-server is not installed
```

If it is not installed, run the following command as the root user to install it:

```
~]# yum install ipa-server
```

26.1. Identity Management and SELinux

Identity Management can map IdM users to configured SELinux roles per host so that it is possible to specify SELinux context for IdM access rights. During the user login process, the System Security Services Daemon (**SSSD**) queries the access rights defined for a particular IdM user. Then the **pam_selinux** module sends a request to the kernel to launch the user process with the proper SELinux context according to the IdM access rights, for example **guest_u:guest_r:guest_t:s0**.

For more information about Identity Management and SELinux, see the Linux Domain, Identity, Authentication, and Policy Guide for Red Hat Enterprise Linux 7.

26.1.1. Trust to Active Directory Domains

In previous versions of Red Hat Enterprise Linux, Identity Management used the **WinSync** utility to allow users from Active Directory (AD) domains to access data stored on IdM domains. To do that, **WinSync** had to replicate the user and group data from the AD server to the local server and kept the data synchronized.

In Red Hat Enterprise Linux 7, the **SSSD** daemon has been enhanced to work with AD and users are able to create a trusted relationship between IdM and AD domains. The user and group data are read directly from the AD server. Additionally, Kerberos cross-realm trust allowing single sign-on (SSO) authentication between the AD and IdM domains is provided. If SSO is set, users from the AD domains can access data protected by Kerberos that is stored on the IdM domains without requiring a password.

This feature is not installed by default. To use it, install the additional *ipa-server-trust-ad* package.

26.2. Configuration Examples

26.2.1. Mapping SELinux users to IdM users

The following procedure shows how to create a new SELinux mapping and how to add a new IdM user to this mapping.

Procedure 26.1. How to Add a User to an SELinux Mapping

1. To create a new SELinux mapping, run the following command where **`SELinux_mapping`** is the name of the new SELinux mapping and the **`--selinuxuser`** option specifies a particular SELinux user:

   ```
   ~]$ ipa selinuxusermap-add SELinux_mapping --selinuxuser=staff_u:s0-
   s0:c0.c1023
   ```

2. Run the following command to add an IdM user with the **`tuser`** user name to the SELinux mapping:

   ```
   ~]$ ipa selinuxusermap-add-user --users=tuser SELinux_mapping
   ```

3. To add a new host named **`ipaclient.example.com`** to the SELinux mapping, run the following command:

   ```
   ~]$ ipa selinuxusermap-add-host --hosts=ipaclient.example.com SELinux_mapping
   ```

4. The **`tuser`** user gets the **`staff_u:s0-s0:c0.c1023`** label when logged in to the *ipaclient.example.com* host:

   ```
   [tuser@ipa-client]$ id -Z
   staff_u:staff_r:staff_t:s0-s0:c0.c1023
   ```

[29] For more information about Identity Management, see the Linux Domain, Identity, Authentication, and Policy Guide for Red Hat Enterprise Linux 7.

Chapter 27. References

The following references are pointers to additional information that is relevant to SELinux but beyond the scope of this guide. Note that due to the rapid development of SELinux, some of this material may only apply to specific releases of Red Hat Enterprise Linux.

Books

SELinux by Example

Mayer, MacMillan, and Caplan

Prentice Hall, 2007

SELinux: NSA's Open Source Security Enhanced Linux

Bill McCarty

O'Reilly Media Inc., 2004

Tutorials and Help

Tutorials and talks from Russell Coker

http://www.coker.com.au/selinux/talks/ibmtu-2004/

Dan Walsh's Journal

http://danwalsh.livejournal.com/

Red Hat Knowledgebase

https://access.redhat.com/site/

General Information

NSA SELinux main website

http://www.nsa.gov/research/selinux/index.shtml

NSA SELinux FAQ

http://www.nsa.gov/research/selinux/faqs.shtml

Mailing Lists

NSA SELinux mailing list

http://www.nsa.gov/research/selinux/list.shtml

Fedora SELinux mailing list

http://www.redhat.com/mailman/listinfo/fedora-selinux-list http://www.redhat.com/mailman/listinfo/fedora-selinux-list

Community

SELinux Project Wiki

http://selinuxproject.org/page/Main_Page

SELinux community page

http://selinux.sourceforge.net/

IRC

irc.freenode.net, #selinux

Revision History

Revision 0.2-2	**Thu Sep 11 2014**	**Barbora Ančincová**

Red Hat Enterprise Linux 7.0 GA release of the book.

Revision 0.1-41	**Tue May 20 2014**	**Tomáš Čapek**

Rebuild for style changes.

Revision 0.1-1	**Tue Jan 17 2013**	**Tomáš Čapek**

Initial creation of the book for Red Hat Enterprise Linux 7