**An Introduction to SELinux on CentOS 7 – Part 1: Basic Concepts**

**Note**  
The commands, packages, and files shown in this tutorial were tested on CentOS 7. The concepts remain the same for other distributions.

**Why SELinux**

Before we begin, let’s understand a few concepts.

SELinux implements what’s known as **MAC** (Mandatory Access Control). This is implemented on top of what’s already present in every Linux distribution, the **DAC** (Discretionary Access Control).

To understand DAC, let’s first consider how traditional Linux file security works.

In a traditional security model, we have three entities: User, Group, and Other (u,g,o) who can have a combination of Read, Write, and Execute (r,w,x) permissions on a file or directory. If a user **jo** creates a file in their home directory, that user will have read/write access to it, and so will the **jo** group. The “other” entity will possibly have no access to it. In the following code block, we can consider the hypothetical contents of jo’s home directory.

Running a command like this:

ls -l /home/jo/

can show output like the following:

total 4

-rwxrw-r--. 1 jo jo 41 Aug 6 22:45 myscript.sh

Now jo can change this access. jo can grant (and restrict) access to this file to other users and groups or change the owner of the file. These actions can leave critical files exposed to accounts who don’t need this access. jo can also restrict to be more secure, but that’s discretionary: there’s no way for the system administrator to enforce it for every single file in the system.

Consider another case: when a Linux process runs, it may run as the root user or another account with superuser privileges. That means if a black-hat hacker takes control of the application, they can use that application to get access to whatever resource the user account has access to. For processes running as the root user, basically this means everything in the Linux server.

Think about a scenario where you want to restrict users from executing shell scripts from their home directories. This can happen when you have developers working on a production system. You would like them to view log files, but you don’t want them to use su or sudo commands, and you don’t want them to run any scripts from their home directories. How do you do that?

SELinux is a way to fine-tune such access control requirements. With SELinux, you can define what a user or process can do. It confines every process to its own domain so the process can interact with only certain types of files and other processes from allowed domains. This prevents a hacker from hijacking any process to gain system-wide access.

**Setting Up a Test System**

To help us learn the concepts, we will build a test server running both a web and an SFTP server. We will start with a bare installation of CentOS 7 with minimal packages installed and install the Apache and vsftp daemons on that server. However, we will not configure either of these applications.

We will also create a few test user accounts in our cloud server. We will use these accounts in different places throughout the lesson.

Finally, we will install needed SELinux-related packages. This is to ensure we can work with the latest SELinux commands.

**Installing Apache and SFTP Services**

First, let’s log in to the server as the **root** user and run the following command to install Apache:

yum install httpd

The output will show the package being downloaded and ask you for permission to install:

Loaded plugins: fastestmirror, langpacks

...

...

================================================================================

Package Arch Version Repository Size

================================================================================

Installing:

httpd x86\_64 2.4.6-18.el7.centos updates 2.7 M

Transaction Summary

================================================================================

Install 1 Package

Total download size: 2.7 M

Installed size: 9.3 M

Is this ok [y/d/N]:

Pressing **y** will install the Apache web server daemon.

Downloading packages:

httpd-2.4.6-18.el7.centos.x86\_64.rpm | 2.7 MB 00:01

Running transaction check

Running transaction test

Transaction test succeeded

Running transaction

Installing : httpd-2.4.6-18.el7.centos.x86\_64 1/1

Verifying : httpd-2.4.6-18.el7.centos.x86\_64 1/1

Installed:

httpd.x86\_64 0:2.4.6-18.el7.centos

Complete!

Start the daemon manually:

service httpd start

Running the service httpd status command will show the service is now running:

Redirecting to /bin/systemctl status httpd.service

httpd.service - The Apache HTTP Server

Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled)

Active: active (running) since Tue 2014-08-19 13:39:48 EST; 1min 40s ago

Main PID: 339 (httpd)

...

...

Next we will install vsftp:

yum install vsftpd

The output should look similar to the following:

Loaded plugins: fastestmirror, langpacks

...

...

==============================================================================================================

Package Arch Version Repository Size

==============================================================================================================

Installing:

vsftpd x86\_64 3.0.2-9.el7 base 165 k

Transaction Summary

==============================================================================================================

Install 1 Package

Total download size: 165 k

Installed size: 343 k

Is this ok [y/d/N]:

Press **y** to install the package.

Next, we will use the service vsftpd start command to start the vsftpd daemon. The output should show something like the following:

Redirecting to /bin/systemctl status vsftpd.service

vsftpd.service - Vsftpd ftp daemon

Loaded: loaded (/usr/lib/systemd/system/vsftpd.service; disabled)

Active: active (running) since Tue 2014-08-19 13:48:57 EST; 4s ago

Process: 599 ExecStart=/usr/sbin/vsftpd /etc/vsftpd/vsftpd.conf (code=exited, status=0/SUCCESS)

Main PID: 600 (vsftpd)

...

...

**Installing SELinux Packages**

A number of packages are used in SELinux. Some are installed by default. Here is a list for Red Hat-based distributions:

* *policycoreutils* (provides utilities for managing SELinux)
* *policycoreutils-python* (provides utilities for managing SELinux)
* *selinux-policy* (provides SELinux reference policy)
* *selinux-policy-targeted* (provides SELinux targeted policy)
* *libselinux-utils* (provides some tools for managing SELinux)
* *setroubleshoot-server* (provides tools for deciphering audit log messages)
* *setools* (provides tools for audit log monitoring, querying policy, and file context management)
* *setools-console* (provides tools for audit log monitoring, querying policy, and file context management)
* *mcstrans* (tools to translate different levels to easy-to-understand format)

Some of these are installed already. To check what SELinux packages are installed on your CentOS 7 system, you can run a few commands like the one below (with different search terms after grep) as the root user:

rpm -qa | grep selinux

The output should look something like this:

libselinux-utils-2.2.2-6.el7.x86\_64

libselinux-2.2.2-6.el7.x86\_64

selinux-policy-targeted-3.12.1-153.el7.noarch

selinux-policy-3.12.1-153.el7.noarch

libselinux-python-2.2.2-6.el7.x86\_64

You can go ahead and install all the packages with the command below (yum will just update any you already have), or just the ones that you find missing from your system:

yum install policycoreutils policycoreutils-python selinux-policy selinux-policy-targeted libselinux-utils setroubleshoot-server setools setools-console mcstrans

Now we should have a system that’s loaded with all the SELinux packages. We also have Apache and SFTP servers running with their default configurations. We also have four regular user accounts ready for testing in addition to the **root** account.

**SELinux Modes**

It’s time to start playing around with SELinux, so let’s begin with SELinux modes. At any one time, SELinux can be in any of three possible modes:

* Enforcing
* Permissive
* Disabled

In enforcing mode SELinux will *enforce* its policy on the Linux system and make sure any unauthorized access attempts by users and processes are denied. The access denials are also written to relevant log files. We will talk about SELinux policies and audit logs later.

Permissive mode is like a semi-enabled state. SELinux doesn’t apply its policy in permissive mode, so no access is denied. However any policy violation is still logged in the audit logs. It’s a great way to test SELinux before enforcing it.

The disabled mode is self-explanatory – the system won’t be running with enhanced security.

**Checking SELinux Modes and Status**

We can run the getenforce command to check the current SELinux mode.

getenforce

SELinux should currently be disabled, so the output will look like this:

Disabled

We can also run the sestatus command:

sestatus

When SELinux is disabled the output will show:

SELinux status: disabled

**SELinux Configuration File**

The main configuration file for SELinux is /etc/selinux/config. We can run the following command to view its contents:

cat /etc/selinux/config

The output will look something 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=disabled

# SELINUXTYPE= can take one of these two values:

# targeted - Targeted processes are protected,

# minimum - Modification of targeted policy. Only selected processes are protected.

# mls - Multi Level Security protection.

SELINUXTYPE=targeted

There are two directives in this file. The SELINUX directive dictates the SELinux mode and it can have three possible values as we discussed before.

The SELINUXTYPE directive determines the policy that will be used. The default value is targeted. With a targeted policy, SELinux allows you to customize and fine tune access control permissions. The other possible value is “MLS” (multilevel security), an advanced mode of protection. Also with MLS, you need to install an additional package.

**Enabling and Disabling SELinux**

Enabling SELinux is fairly simple; but unlike disabling it, should be done in a two-step process. We assume that SELinux is currently disabled, and that you’ve installed all of the SELinux packages from the earlier section.

As a first step, we need to edit the /etc/selinux/config file to change the SELINUX directive to permissive mode.

vi /etc/sysconfig/selinux

...

SELINUX=permissive

...

Setting the status to **permissive** first is necessary because every file in the system needs to have its context labelled before SELinux can be enforced. Unless all files are properly labelled, processes running in confined domains may fail because they can’t access files with the correct contexts. This can cause the boot process to fail or start with errors. We will introduce *contexts* and *domains* later in the tutorial.

Now issue a system reboot:

reboot

The reboot process will see all the files in the server labelled with an SELinux context. Since the system is running in permissive mode, SELinux errors and access denials will be reported but it won’t stop anything.

Log in to your server again as **root**. Next, search for the string “SELinux is preventing” from the contents of the /var/log/messages file.

cat /var/log/messages | grep "SELinux is preventing"

If there are no errors reported, we can safely move to the next step. However, it would still be a good idea to search for text containing “SELinux” in /var/log/messages file. In our system, we ran the following command:

cat /var/log/messages | grep "SELinux"

This showed some error messages related to the GNOME Desktop that was running. This was happening when SELInux was either disabled or in permissive mode:

Aug 20 11:31:14 localhost kernel: SELinux: Initializing.

Aug 20 11:31:16 localhost kernel: SELinux: Disabled at runtime.

Aug 20 11:31:21 localhost journal: Unable to lookup SELinux process context: Invalid argument

Aug 20 11:33:20 localhost gnome-session: SELinux Troubleshooter: Applet requires SELinux be enabled to run.

Aug 20 11:37:15 localhost kernel: SELinux: Initializing.

Aug 20 11:37:17 localhost kernel: SELinux: Disabled at runtime.

Aug 20 11:37:23 localhost journal: Unable to lookup SELinux process context: Invalid argument

Aug 20 11:37:44 localhost gnome-session: SELinux Troubleshooter: Applet requires SELinux be enabled to run.

Aug 20 11:39:42 localhost kernel: SELinux: Initializing.

Aug 20 11:39:44 localhost kernel: SELinux: Disabled at runtime.

Aug 20 11:39:50 localhost journal: Unable to lookup SELinux process context: Invalid argument

These types of errors are fine.

In the second phase, we need to edit the config file to change the SELINUX directive from **permissive** to **enforcing** in the /etc/sysconfig/selinux file:

...

SELINUX=enforcing

...

Next, reboot the server again.

reboot

Once the server is back online, we can run the sestatus command to check the SELinux status. It should now show more details about the server:

SELinux status: enabled

SELinuxfs mount: /sys/fs/selinux

SELinux root directory: /etc/selinux

Loaded policy name: targeted

Current mode: permissive

Mode from config file: error (Success)

Policy MLS status: enabled

Policy deny\_unknown status: allowed

Max kernel policy version: 28

Check the /var/log/messages file:

cat /var/log/messages | grep "SELinux"

There should be no errors. The output should look something like this:

Aug 20 11:42:06 localhost kernel: SELinux: Initializing.

Aug 20 11:42:09 localhost systemd[1]: Successfully loaded SELinux policy in 183.302ms.

Aug 20 11:44:25 localhost kernel: SELinux: Initializing.

Aug 20 11:44:28 localhost systemd[1]: Successfully loaded SELinux policy in 169.039ms.

**Checking SELinux Modes and Status (Again)**

We can run the getenforce command to check the current SELinux mode.

getenforce

If our system is running in enforcing mode the output will look like this:

Enforcing

The output will be different if SELinux is disabled:

Disabled

We can alo run the sestatus command to get a better picture.

sestatus

If SELinux isn’t disabled, the output will show its current status, its current mode, the mode defined in the configuration file, and the policy type.

SELinux status: enabled

SELinuxfs mount: /sys/fs/selinux

SELinux root directory: /etc/selinux

Loaded policy name: targeted

Current mode: enforcing

Mode from config file: enforcing

Policy MLS status: enabled

Policy deny\_unknown status: allowed

Max kernel policy version: 28

When SELinux is disabled the output will show:

SELinux status: disabled

We can also temporarily switch between enforcing and permissive modes using the setenforce command. (Note that we can’t run setenforce when SELinux is disabled.)

First change the SELinux mode from enforcing to permissive in our CentOS 7 system:

setenforce permissive

Running the sestatus command now shows the current mode is different from the mode defined in config file:

SELinux status: enabled

SELinuxfs mount: /sys/fs/selinux

SELinux root directory: /etc/selinux

Loaded policy name: targeted

Current mode: permissive

Mode from config file: enforcing

Policy MLS status: enabled

Policy deny\_unknown status: allowed

Max kernel policy version: 28

Switch back to **enforcing**:

setenforce enforcing

**SELinux Policy**

At the heart of SELinux’ security engine is its *policy*. A policy is what the name implies: a set of rules that define the security and access rights for everything in the system. And when we say *everything*, we mean users, roles, processes, and files. The policy defines how each of these entities are related to one another.

**Some Basic Terminology**

To understand policy, we have to learn some basic terminology. We will go into the details later, but here is a brief introduction. An SELinux policy defines user access to roles, role access to domains, and domain access to types.

**Users**

SELinux has a set of pre-built users. Every regular Linux user account is mapped to one or more SELinux users.

In Linux, a user runs a process. This can be as simple as the user **jo** opening a document in the vi editor (it will be jo’s account running the vi process) or a service account running the httpd daemon. In the SELinux world, a process (a daemon or a running program) is called a *subject*.

**Roles**

A *role* is like a gateway that sits between a user and a process. A role defines which users can access that process. Roles are not like groups, but more like filters: a user may enter or assume a role at any time provided the role grants it. The definition of a role in SELinux policy defines which users have access to that role. It also defines what process domains the role itself has access to. Roles come into play because part of SELinux implements what’s known as **Role Based Access Control** (RBAC).

**Subjects and Objects**

A *subject* is a process and can potentially affect an *object*.

An *object* in SELinux is anything that can be acted upon. This can be a file, a directory, a port, a tcp socket, the cursor, or perhaps an X server. The actions that a subject can perform on an object are the subject’s *permissions*.

**Domains are for Subjects**

A *domain* is the context within which an SELinux subject (process) can run. That context is like a wrapper around the subject. It tells the process what it can and can’t do. For example, the domain will define what files, directories, links, devices, or ports are accessible to the subject.

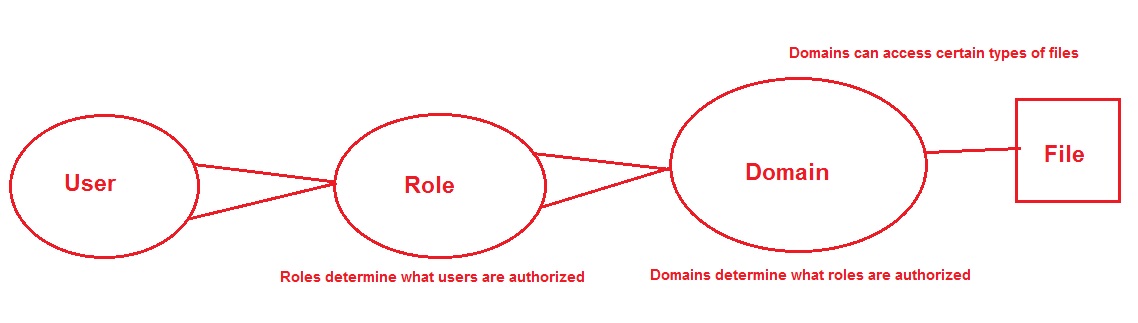
**Types are for Objects**

A *type* is the context for a file’s context that stipulates the file’s purpose. For example, the context of a file may dictate that it’s a web page, or that the file belongs to the /etc directory, or that the file’s owner is a specific SELinux user. A file’s context is called its *type* in SELinux lingo.

**So what is SELinux policy?**

SELinux policy defines user access to roles, role access to domains, and domain access to types. First the user has to be authorized to enter a role, and then the role has to be authorized to access the domain. The domain in turn is restricted to access only certain types of files.

The policy itself is a bunch of rules that say that so-and-so users can assume only so-and-so roles, and those roles will be authorized to access only so-and-so domains. The domains in turn can access only so-and-so file types. The following image shows the concept:



Terminology tip: The last bit, where a process running within a particular domain can perform only certain operations on certain types of objects, is called *Type Enforcement* (TE).

Coming back to the topic of policies, SELinux policy implementations are also typically *targeted* by default. If you remember the SELinux config file that we saw before, the SELINUXTYPE directive is set to be targeted. What this means is that, by default, SELinux will restrict only certain processes in the system (i.e. only certain processes are targeted). The ones that are not targeted will run in unconfined domains.

The alternative is a deny-by-default model where every access is denied unless approved by the policy. It would be a very secure implementation, but this also means that developers have to anticipate every single possible permission every single process may need on every single possible object. The default behaviour sees SELinux concerned with only certain processes.

**SELinux Policy Behavior**

SELinux policy is not something that replaces traditional DAC security. If a DAC rule prohibits a user access to a file, SELinux policy rules won’t be evaluated because the first line of defense has already blocked access. SELinux security decisions come into play *after* DAC security has been evaluated.

When an SELinux-enabled system starts, the policy is loaded into memory. SELinux policy comes in modular format, much like the kernel modules loaded at boot time. And just like the kernel modules, they can be dynamically added and removed from memory at run time. The *policy store* used by SELinux keeps track of the modules that have been loaded. The sestatus command shows the policy store name. The semodule -l command lists the SELinux policy modules currently loaded into memory.

To get a feeling for this, let’s run the semodule command:

semodule -l | less

The output will look something like this:

abrt 1.2.0

accountsd 1.0.6

acct 1.5.1

afs 1.8.2

aiccu 1.0.2

aide 1.6.1

ajaxterm 1.0.0

alsa 1.11.4

amanda 1.14.2

amtu 1.2.3

anaconda 1.6.1

antivirus 1.0.0

apache 2.4.0

...

...

semodule can be used for a number other tasks like installing, removing, reloading, upgrading, enabling and disabling SELinux policy modules.

By now you would probably be interested to know where the module files are located. Most modern distributions include binary versions of the modules as part of the SELinux packages. The policy files have a .pp extension. For CentOS 7, we can run the following command:

ls -l /etc/selinux/targeted/modules/active/modules/

The listing shows a number of files with the .pp extension. If you look closely, they will relate to different applications:

...

-rw-r--r--. 1 root root 10692 Aug 20 11:41 anaconda.pp

-rw-r--r--. 1 root root 11680 Aug 20 11:41 antivirus.pp

-rw-r--r--. 1 root root 24190 Aug 20 11:41 apache.pp

-rw-r--r--. 1 root root 11043 Aug 20 11:41 apcupsd.pp

...

The .pp files are not human readable though.

The way SELinux modularization works is that when the system boots, policy modules are combined into what’s known as the *active policy*. This policy is then loaded into memory. The combined binary version of this loaded policy can be found under the /etc/selinux/targeted/policy directory.

ls -l /etc/selinux/targeted/policy/

will show the active policy.

total 3428

-rw-r--r--. 1 root root 3510001 Aug 20 11:41 policy.29

**Changing SELinux Boolean Settings**

Although you can’t read the policy module files, there’s a simple way to tweak their settings. That’s done through SELinux *booleans*.

To see how it works, let’s run the semanage boolean -l command.

semanage boolean -l | less

This shows the different switches that can be turned on or off, what they do, and their current statuses:

ftp\_home\_dir (off , off) Allow ftp to home dir

smartmon\_3ware (off , off) Allow smartmon to 3ware

mpd\_enable\_homedirs (off , off) Allow mpd to enable homedirs

xdm\_sysadm\_login (off , off) Allow xdm to sysadm login

xen\_use\_nfs (off , off) Allow xen to use nfs

mozilla\_read\_content (off , off) Allow mozilla to read content

ssh\_chroot\_rw\_homedirs (off , off) Allow ssh to chroot rw homedirs

mount\_anyfile (on , on) Allow mount to anyfile

...

...

We can see the first option allows the FTP daemon to access users’ home directories. The setting is turned off at the moment.

To change any of the settings, we can use the setsebool command. As an example, let’s consider the anonymous FTP write access:

getsebool ftpd\_anon\_write

This shows us the switch is off at the moment:

ftpd\_anon\_write --> off

Next we change the boolean to enable it:

setsebool ftpd\_anon\_write on

Checking the value again should show the change:

ftpd\_anon\_write --> on

Changed booleans are not permanent. They revert to their old values after a reboot. To make things permanent, we can use the -P switch with the setsebool command.

**Conclusion**

In the first part of this tutorial we have tried to understand a few basic concepts around SELinux. We have seen how SELinux can secure a system, how we can enable it and what modes it can be running in. We have also touched on the topic of SELinux policy. Next, we will learn [how to use SELinux to restrict access to files and processes](https://www.digitalocean.com/community/tutorials/an-introduction-to-selinux-on-centos-7-part-2-files-and-processes).

# Part 2

### Introduction

In the [first part of our SELinux series](https://www.digitalocean.com/community/tutorials/an-introduction-to-selinux-on-centos-7-part-1-basic-concepts), we saw how to enable and disable SELinux and how to change some of the policy settings using boolean values. In this second part, we will talk about file and process security contexts.

To refresh your memory from the previous tutorial, a file security context is a type and a process security context is a domain.

## Creating Test User Accounts

First, let’s create four user accounts to demonstrate SELinux capabilities as we go along.

* regularuser
* switcheduser
* guestuser
* restricteduser

You should currently be the **root** user. Let’s run the following command to add the **regularuser** account:

useradd -c "Regular User" regularuser

Then we run the passwd command to change its password:

passwd regularuser

The output will ask us for new password. Once supplied, the account will be ready for login:

Changing password for user regularuser.

New password:

Retype new password:

passwd: all authentication tokens updated successfully.

Let’s create the other accounts too:

useradd -c "Switched User" switcheduser

passwd switcheduser

useradd -c "Guest User" guestuser

passwd guestuser

useradd -c "Restricted Role User" restricteduser

passwd restricteduser

## SELinux for Processes and Files

The purpose of SELinux is to secure how processes access files in a Linux environment. Without SELinux, a process or application like the Apache daemon will run under the context of the user that started it. So if your system is compromised by a rogue application that’s running under the root user, the app can do whatever it wants because root has all-encompassing rights on every file.

SELinux tries to go one step further and eliminate this risk. With SELinux, a process or application will have only the rights it needs to function and NOTHING more. The SELinux policy for the application will determine what types of files it needs access to and what processes it can transition to. SELinux policies are written by app developers and shipped with the Linux distribution that supports it. A policy is basically a set of rules that maps processes and users to their rights.

We begin the discussion of this part of the tutorial by understanding what SELinux contexts and domains mean.

The first part of security puts a label on each entity in the Linux system. A label is like any other file or process attribute (owner, group, date created etc.); it shows the context of the resource. So what’s a context? Put simply, a context is a collection of security related information that helps SELinux make access control decisions. Everything in a Linux system can have a security context: a user account, a file, a directory, a daemon, or a port can all have their security contexts. However, security context will mean different things for different types of objects.  
​

### SELinux File Contexts

Let’s start by understanding SELinux file contexts. Let’s look at the output of a regular ls -l command against the /etc directory.

ls -l /etc/\*.conf

This will show us a familiar output:

...

-rw-r--r--. 1 root root 19 Aug 19 21:42 /etc/locale.conf

-rw-r--r--. 1 root root 662 Jul 31 2013 /etc/logrotate.conf

-rw-r--r--. 1 root root 5171 Jun 10 07:35 /etc/man\_db.conf

-rw-r--r--. 1 root root 936 Jun 10 05:59 /etc/mke2fs.conf

...

Simple, right? Let’s now add the -Z flag:

ls -Z /etc/\*.conf

We now have an extra column of information after the user and group ownership:

...

-rw-r--r--. root root system\_u:object\_r:locale\_t:s0 /etc/locale.conf

-rw-r--r--. root root system\_u:object\_r:etc\_t:s0 /etc/logrotate.conf

-rw-r--r--. root root system\_u:object\_r:etc\_t:s0 /etc/man\_db.conf

-rw-r--r--. root root system\_u:object\_r:etc\_t:s0 /etc/mke2fs.conf

...

This column shows the security contexts of the files. A file is said to have been labelled with its security context when you have this information available for it. Let’s take a closer look at one of the security contexts.

-rw-r--r--. root root system\_u:object\_r:etc\_t:s0 /etc/logrotate.conf

The security context is this part:

system\_u:object\_r:etc\_t:s0

There are four parts and each part of the security context is separated by a colon (:). The first part is the SELinux user context for the file. We will discuss SELinux users later, but for now, we can see that it’s **system\_u**. Each Linux user account maps to an SELinux user, and in this case, the **root** user that owns the file is mapped to the **system\_u** SELinux user. This mapping is done by the SELinux policy.

The second part specifies the SELinux role, which is **object\_r**. To brush up on SELinux roles, look back at the first SELinux article.

What’s most important here is the third part, the type of the file that’s listed here as **etc\_t**. This is the part that defines what type the file or directory belongs to. We can see that most files belong to the **etc\_t** type in the /etc directory. Hypothetically, you can think of type as a sort of “group” or attribute for the file: it’s a way of classifying the file.

We can also see some files may belong to other types, like locale.conf which has a **locale\_t** type. Even when all the files listed here have the same user and group owners, their types could be different.

As another example, let’s check the type contexts for user home directories:

ls -Z /home

Home directories will have a different context type: **user\_home\_dir\_t**

drwx------. guestuser guestuser unconfined\_u:object\_r:user\_home\_dir\_t:s0 guestuser

drwx------. root root system\_u:object\_r:lost\_found\_t:s0 lost+found

drwx------. regularuser regularuser unconfined\_u:object\_r:user\_home\_dir\_t:s0 regularuser

drwx------. restricteduser restricteduser unconfined\_u:object\_r:user\_home\_dir\_t:s0 restricteduser

drwx------. switcheduser switcheduser unconfined\_u:object\_r:user\_home\_dir\_t:s0 switcheduser

drwx------. sysadmin sysadmin unconfined\_u:object\_r:user\_home\_dir\_t:s0 sysadmin

The fourth part of the security context, **s0**, has to do with multilevel security or MLS. Basically this is another way of enforcing SELinux security policy, and this part shows the sensitivity of the resource (**s0**). We will briefly talk about sensitivity and categories later. For most vanilla setups of SELinux, the first three security contexts are more important.

### SELinux Process Contexts

Let’s now talk about process security contexts.

Start the Apache and SFTP services. We installed these services in the first SELinux tutorial.

service httpd start

service vsftpd start

We can run the ps command with a few flags to show the Apache and SFTP processes running on our server:

ps -efZ | grep 'httpd\|vsftpd'

Once again the -Z flag is used for displaying SELinux contexts. The output shows the user running the process, the process ID, and the parent process ID:

system\_u:system\_r:httpd\_t:s0 root 7126 1 0 16:50 ? 00:00:00 /usr/sbin/httpd -DFOREGROUND

system\_u:system\_r:httpd\_t:s0 apache 7127 7126 0 16:50 ? 00:00:00 /usr/sbin/httpd -DFOREGROUND

system\_u:system\_r:httpd\_t:s0 apache 7128 7126 0 16:50 ? 00:00:00 /usr/sbin/httpd -DFOREGROUND

system\_u:system\_r:httpd\_t:s0 apache 7129 7126 0 16:50 ? 00:00:00 /usr/sbin/httpd -DFOREGROUND

system\_u:system\_r:httpd\_t:s0 apache 7130 7126 0 16:50 ? 00:00:00 /usr/sbin/httpd -DFOREGROUND

system\_u:system\_r:httpd\_t:s0 apache 7131 7126 0 16:50 ? 00:00:00 /usr/sbin/httpd -DFOREGROUND

system\_u:system\_r:ftpd\_t:s0-s0:c0.c1023 root 7209 1 0 16:54 ? 00:00:00 /usr/sbin/vsftpd /etc/vsftpd/vsftpd.conf

unconfined\_u:unconfined\_r:unconfined\_t:s0-s0:c0.c1023 root 7252 2636 0 16:57 pts/0 00:00:00 grep --color=auto httpd\|vsftpd

The security context is this part:

system\_u:system\_r:httpd\_t:s0

The security context has four parts: user, role, domain, and sensitivity. The user, role, and sensitivity work just like the same contexts for files (explained in the previous section). The domain is unique to processes.

In the example above, we can see that a few processes are running within the **httpd\_t** domain, while one is running within the **ftpd\_t** domain.

So what’s the domain doing for processes? It gives the process a context to run within. It’s like a bubble around the process that confines it. It tells the process what it can do and what it can’t do. This confinement makes sure each process domain can act on only certain types of files and nothing more.

Using this model, even if a process is hijacked by another malicious process or user, the worst it can do is to damage the files it has access to. For example, the vsftp daemon will not have access to files used by say, sendmail or samba. This restriction is implemented from the kernel level: it’s enforced as the SELinux policy loads into memory, and thus the access control becomes mandatory.

## Naming Conventions

Before we go any further, here is a note about SELinux naming convention. SELinux Users are suffixed by “\_u”, roles are suffixed by “\_r” and types (for files) or domains (for processes) are suffixed by “\_t”.

## How Processes Access Resources

So far we have seen that files and processes can have different contexts, and that they are restricted to their own types or domains. So how does a process run? To run, a process needs to access its files and perform some actions on them (open, read, modify, or execute). We have also learned that each process can have access to only certain types of resources (files, directories, ports, etc.).

SELinux stipulates these access rules in a policy. The access rules follow a standard allow statement structure:

allow <domain> <type>:<class> { <permissions> };

We have already talked about domains and types. **Class** defines what the resource actually represents (file, directory, symbolic link, device, ports, cursor etc.)

Here’s what this generic allow statement means:

* If a process is of certain domain
* And the resource object it’s trying to access is of certain class and type
* Then allow the access
* Else deny access

To see how this works, let’s consider the security contexts of the httpd daemon running on our CentOS 7 system:

system\_u:system\_r:httpd\_t:s0 7126 ? 00:00:00 httpd

system\_u:system\_r:httpd\_t:s0 7127 ? 00:00:00 httpd

system\_u:system\_r:httpd\_t:s0 7128 ? 00:00:00 httpd

system\_u:system\_r:httpd\_t:s0 7129 ? 00:00:00 httpd

system\_u:system\_r:httpd\_t:s0 7130 ? 00:00:00 httpd

system\_u:system\_r:httpd\_t:s0 7131 ? 00:00:00 httpd

The default home directory for the web server is /var/www/html. Let’s create a file within that directory and check its context:

touch /var/www/html/index.html

ls -Z /var/www/html/\*

The file context for our web content will be **httpd\_sys\_content\_t**:

-rw-r--r--. root root unconfined\_u:object\_r:httpd\_sys\_content\_t:s0 /var/www/html/index.html

We can use the sesearch command to check the type of access allowed for the httpd daemon:

sesearch --allow --source httpd\_t --target httpd\_sys\_content\_t --class file

The flags used with the command are fairly self-explanatory: the source domain is **httpd\_t**, the same domain Apache is running in. We are interested about target resources that are files and have a type context of **httpd\_sys\_content\_t**. Your output should look like this:

Found 4 semantic av rules:

allow httpd\_t httpd\_sys\_content\_t : file { ioctl read getattr lock open } ;

allow httpd\_t httpd\_content\_type : file { ioctl read getattr lock open } ;

allow httpd\_t httpd\_content\_type : file { ioctl read getattr lock open } ;

allow httpd\_t httpdcontent : file { ioctl read write create getattr setattr lock append unlink link rename execute open } ;

Notice the first line:

allow httpd\_t httpd\_sys\_content\_t : file { ioctl read getattr lock open } ;

This says that the httpd daemon (the Apache web server) has I/O control, read, get attribute, lock, and open access to files of the **httpd\_sys\_content** type. In this case our index.html file has the same type.

Going one step further, let’s first modify the web page (/var/www/html/index.html). Edit the file to contain this content:

<html>

<title>

This is a test web page

</title>

<body>

<h1>This is a test web page</h1>

</body>

</html>

Next, we will change the permission of the /var/www/ folder and its contents, followed by a restart of the httpd daemon:

chmod -R 755 /var/www

service httpd restart

We will then try to access it from a browser:



**Note**  
Depending on how your server is set up, you may have to enable port 80 in the IPTables firewall for allowing incoming HTTP traffic from outside the server. We won’t go into the details of enabling ports in IPTables here. There are some excellent DigitalOcean [articles](https://www.digitalocean.com/community/tutorials/how-to-set-up-a-firewall-using-iptables-on-ubuntu-14-04) on the topic which you can use.

So far so good. The httpd daemon is authorized to access a particular type of file and we can see it when accessing via the browser. Next, let’s make things a little different by changing the context of the file. We will use the chcon command for it. The --type flag for the command allows us to specify a new type for the target resource. Here, we are changing the file type to **var\_t**.

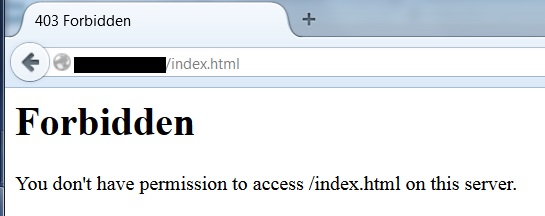
chcon --type var\_t /var/www/html/index.html

We can confirm the type change:

ls -Z /var/www/html/

-rwxr-xr-x. root root unconfined\_u:object\_r:var\_t:s0 index.html

Next, when we try to access the web page (i.e. the httpd daemon tries to read the file), you may get a **Forbidden** error, or you may see the generic CentOS “Testing 123” page:



So what’s happening here? Obviously some access is now being denied, but whose access is it? As far as SELinux is concerned, the web server is authorized to access only certain types of files and var\_t is not one of those contexts. Since we changed the context of the index.html file to var\_t, Apache can no longer read it and we get an error.

To make things work again, let’s change the file type with the restorecon command. The -v switch shows the change of context labels:

restorecon -v /var/www/html/index.html

restorecon reset /var/www/html/index.html context unconfined\_u:object\_r:var\_t:s0->unconfined\_u:object\_r:httpd\_sys\_content\_t:s0

If we try to access the page now, it will show our “This is a test web page” text again.

This is an important concept to understand: making sure files and directories have the correct context is pivotal to making sure SELinux is behaving as it should. We will see a practical use case at the end of this section, but before that, let’s talk about a few more things.

## Context Inheritance for Files and Directories

SELinux enforces something we can term as “context inheritance”. What this means is that unless specified by the policy, processes and files are created with the contexts of their parents.

So if we have a process called “proc\_a” spawning another process called “proc\_b”, the spawned process will run in the same domain as “proc\_a” unless specified otherwise by the SELinux policy.

Similarly, if we have a directory with a type of “some\_context\_t”, any file or directory created under it will have the same type context unless the policy says otherwise.

To illustrate this, let’s check the contexts of the /var/www/ directory:

ls -Z /var/www

The html directory within /var/www/ has the **httpd\_sys\_content\_t** type context. As we saw before, the index.html file within it has the same context (i.e., the context of the parent):

drwxr-xr-x. root root system\_u:object\_r:httpd\_sys\_script\_exec\_t:s0 cgi-bin

drwxr-xr-x. root root system\_u:object\_r:httpd\_sys\_content\_t:s0 html

This inheritance is not preserved when files are copied to another location. In a copy operation, the copied file or directory will assume the type context of the target location. In the code snippet below, we are copying the index.html file (with “httpd\_sys\_content\_t” type context) to the /var/ directory:

* cp /var/www/html/index.html /var/

Copy

If we check the copied file’s context, we will see it has changed to **var\_t**, the context of its current parent directory:

* ls -Z /var/index.html

Copy

-rwxr-xr-x. root root unconfined\_u:object\_r:var\_t:s0 /var/index.html

This change of context can be overridden by the --preserver=context clause in the cp command.

When files or directories are moved, original contexts are preserved. In the following command, we are moving the /var/index.html to the /etc/ directory:

* mv /var/index.html /etc/

Copy

When we check the moved file’s context, we see that the **var\_t** context has been preserved under the /etc/ directory:

* ls -Z /etc/index.html

Copy

-rwxr-xr-x. root root unconfined\_u:object\_r:var\_t:s0 /etc/index.html

So why are we so concerned with file contexts? Why is this copy and move concept important? Think about it: maybe you decided to copy all your web server’s HTML files to a separate directory under the root folder. You have done this to simplify you backup process and also to tighten security: you don’t want any hacker to easily guess where your website’s files are. You have updated the directory’s access control, changed the web config file to point to the new location, restarted the service, but it still doesn’t work. Perhaps you can then look at the contexts of the directory and its files as the next troubleshooting step. Let’s run it as a practical example.

### SELinux in Action: Testing a File Context Error

First, let’s create a directory named www under the root. We will also create a folder called html under www.

* mkdir -p /www/html

Copy

If we run the ls -Z command, we will see these directories have been created with the **default\_t** context:

* ls -Z /www/

Copy

drwxr-xr-x. root root unconfined\_u:object\_r:default\_t:s0 html

Next we copy the contents of the /var/www/html directory to /www/html:

* cp /var/www/html/index.html /www/html/

Copy

The copied file will have a context of **default\_t**. That’s the context of the parent directory.

We now edit the httpd.conf file to point to this new directory as the web site’s root folder. We will also have to relax the access rights for this directory.

* vi /etc/httpd/conf/httpd.conf

Copy

First we comment out the existing location for document root and add a new DocumentRoot directive to /www/html:

# DocumentRoot "/var/www/html"

DocumentRoot "/www/html"

We also comment out the access rights section for the existing document root and add a new section:

#<Directory "/var/www">

# AllowOverride None

# Allow open access:

# Require all granted

#</Directory>

<Directory "/www">

AllowOverride None

# Allow open access:

Require all granted

</Directory>

We leave the location of the cgi-bin directory as it is. We are not getting into detailed Apache configuration here; we just want our site to work for SELinux purposes.

Finally, restart the httpd daemon:

* service httpd restart

Copy

Once the server has been restarted, accessing the web page will give us the same “403 Forbidden” error (or default “Testing 123” page) we saw before.

The error is happening because the index.html file’s context changed during the copy operation. It needs to be changed back to its original context (httpd\_sys\_content\_t).

But how do we do that?

## Changing and Restoring SELinux File Contexts

In a previous code sample we saw two commands for changing file contents: chcon and restorecon. Running chcon is a temporary measure. You can use it to temporarily change file or directory contexts for troubleshooting access denial errors. However, this method is only temporary: a file system relabel or running the restorecon command will revert the file back to its original context.

Also, running chcon requires you to know the correct context for the file; the --type flag specifies the context for the target. restorecon doesn’t need this specified. If you run restorecon, the file will have the correct context re-applied and the changes will be made permanent.

But if you don’t know the file’s correct context, how does the system know which context to apply when it runs restorecon?

Conveniently, SELinux “remembers” the context of every file or directory in the server. In CentOS 7, contexts of files already existing in the system are listed in the /etc/selinux/targeted/contexts/files/file\_contexts file. It’s a large file and it lists every file type associated with every application supported by the Linux distribution. Contexts of new directories and files are recorded in the /etc/selinux/targeted/contexts/files/file\_contexts.local file. So when we run the restorecon command, SELinux will look up the correct context from one of these two files and apply it to the target.

The code snippet below shows an extract from one of the files:

* cat /etc/selinux/targeted/contexts/files/file\_contexts

Copy

...

/usr/(.\*/)?lib(/.\*)? system\_u:object\_r:lib\_t:s0

/opt/(.\*/)?man(/.\*)? system\_u:object\_r:man\_t:s0

/dev/(misc/)?agpgart -c system\_u:object\_r:agp\_device\_t:s0

/usr/(.\*/)?sbin(/.\*)? system\_u:object\_r:bin\_t:s0

/opt/(.\*/)?sbin(/.\*)? system\_u:object\_r:bin\_t:s0

/etc/(open)?afs(/.\*)? system\_u:object\_r:afs\_config\_t:s0

...

To permanently change the context of our index.html file under /www/html, we have to follow a two-step process.

* First we run the semanage fcontext command. This will write the new context to the /etc/selinux/targeted/contexts/files/file\_contexts.local file. But it won’t relabel the file itself. We’ll do this for both directories.
* semanage fcontext --add --type httpd\_sys\_content\_t "/www(/.\*)?"
* semanage fcontext --add --type httpd\_sys\_content\_t "/www/html(/.\*)?"

Copy

To make sure, we can check the file context database (note that we are using the file\_contexts.local file):

* cat /etc/selinux/targeted/contexts/files/file\_contexts.local

Copy

You should see the updated contexts:

# This file is auto-generated by libsemanage

# Do not edit directly.

/www(/.\*)? system\_u:object\_r:httpd\_sys\_content\_t:s0

/www/html(/.\*)? system\_u:object\_r:httpd\_sys\_content\_t:s0

Next, we will run the restorecon command. This will relabel the file or directory with what’s been recorded in the previous step:

* restorecon -Rv /www

Copy

This should reset the context in three levels: the top level /www directory, the /www/html directory under it and the index.html file under /www/html:

restorecon reset /www context unconfined\_u:object\_r:default\_t:s0->unconfined\_u:object\_r:httpd\_sys\_content\_t:s0

restorecon reset /www/html context unconfined\_u:object\_r:default\_t:s0->unconfined\_u:object\_r:httpd\_sys\_content\_t:s0

restorecon reset /www/html/index.html context unconfined\_u:object\_r:default\_t:s0->unconfined\_u:object\_r:httpd\_sys\_content\_t:s0

If we now try to access the web page, it should work.

There is a nifty tool called matchpathcon that can help troubleshoot context-related problems. This command will look at the current context of a resource and compare it with what’s listed under the SELinux context database. If different, it will suggest the change required. Let’s test this with the /www/html/index.html file. We will use the -V flag that verifies the context:

* matchpathcon -V /www/html/index.html

Copy

The matchpathcon output should show that the context is verified.

/www/html/index.html verified.

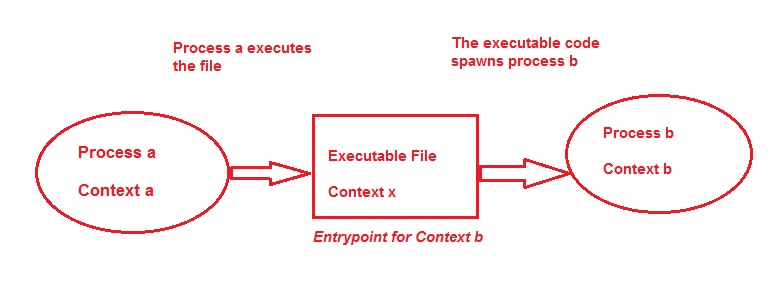
For an incorrectly labelled file, the message will say what the context should be:

/www/html/index.html has context unconfined\_u:object\_r:default\_t:s0, should be system\_u:object\_r:httpd\_sys\_content\_t:s0

## Domain Transition

So far we have seen how processes access file system resources. We will now see how processes access other processes.

Domain transition is the method where a process changes its context from one domain to another. To understand it, let’s say you have a process called proc\_a running within a context of contexta\_t. With domain transition, proc\_a can run an application (a program or an executable script) called app\_x that would spawn another process. This new process could be called proc\_b and it could be running within the contextb\_t domain. So effectively, contexta\_t is transitioning to contextb\_t through app\_x. The app\_x executable is working as an entrypoint to contextb\_t. The flow can be illustrated below:



The case of domain transition is fairly common in SELinux. Let’s consider the vsftpd process running on our server. If it is not running, we can run the service vsftpd start command to start the daemon.

Next we consider the systemd process. This is the ancestor of all processes. This is the replacement of the System V init process and runs within a context of **init\_t**. :

* ps -eZ | grep init

Copy

system\_u:system\_r:init\_t:s0 1 ? 00:00:02 systemd

system\_u:system\_r:mdadm\_t:s0 773 ? 00:00:00 iprinit

The process running within the **init\_t** domain is a short-lived one: it will invoke the binary executable /usr/sbin/vsftpd, which has a type context of **ftpd\_exec\_t**. When the binary executable starts, it becomes the vsftpd daemon itself and runs within the **ftpd\_t** domain.

We can check the domain contexts of the files and processes:

* ls -Z /usr/sbin/vsftpd

Copy

Shows us:

-rwxr-xr-x. root root system\_u:object\_r:ftpd\_exec\_t:s0 /usr/sbin/vsftpd

Checking the process:

* ps -eZ | grep vsftpd

Copy

Shows us:

system\_u:system\_r:ftpd\_t:s0-s0:c0.c1023 7708 ? 00:00:00 vsftpd

So here the process running in the **init\_t** domain is executing a binary file with the **ftpd\_exec\_t** type. That file starts a daemon within the **ftpd\_t** domain.

This transition is not something the application or the user can control. This has been stipulated in the SELinux policy that loads into memory as the system boots. In a non-SELinux server a user can start a process by switching to a more powerful account (provided she or he has the right to do so). In SELinux, such access is controlled by pre-written policies. And that’s another reason SELinux is said to implement Mandatory Access Control.

Domain transition is subject to three strict rules:

* The parent process of the source domain must have the execute permission for the application sitting between both the domains (this is the entrypoint).
* The file context for the application must be identified as an entrypoint for the target domain.
* The original domain must be allowed to transition to the target domain.

Taking the vsftpd daemon example above, let’s run the sesearch command with different switches to see if the daemon conforms to these three rules.

First, the source domain init\_t needs to have execute permission on the entrypoint application with the ftpd\_exec\_t context. So if we run the following command:

* sesearch -s init\_t -t ftpd\_exec\_t -c file -p execute -Ad

Copy

The result shows that processes within init\_t domain can read, get attribute, execute, and open files of ftpd\_exec\_t context:

Found 1 semantic av rules:

allow init\_t ftpd\_exec\_t : file { read getattr execute open } ;

Next, we check if the binary file is the entrypoint for the target domain ftpd\_t:

sesearch -s ftpd\_t -t ftpd\_exec\_t -c file -p entrypoint -Ad

And indeed it is so:

Found 1 semantic av rules:

allow ftpd\_t ftpd\_exec\_t : file { ioctl read getattr lock execute execute\_no\_trans entrypoint open } ;

And finally, the source domain init\_t needs to have permission to transition to the target domain ftpd\_t:

* sesearch -s init\_t -t ftpd\_t -c process -p transition -Ad

Copy

As we can see below, the source domain has that permission:

Found 1 semantic av rules:

allow init\_t ftpd\_t : process transition ;

## Unconfined Domains

When we introduced the concept of domains, we compared it to a hypothetical bubble around the process: something that stipulates what the process can and can’t do. This is what confines the process.

SELinux also has processes that run within unconfined domains. As you can imagine, unconfined processes would have all types of access in the system. Even then, this full access is not arbitrary: full access is also specified in the SELinux policy.

Example of an unconfined process domain would be unconfined\_t. This is the same domain logged in users run their processes by default. We will talk about users and their accesses to process domains in subsequent sections.

## Conclusion

We have covered some very important SELinux concepts here today. Managing file and process context is at the heart of a successful SELinux implementation. As we will see in the next and [final part of this series](https://www.digitalocean.com/community/tutorials/an-introduction-to-selinux-on-centos-7-part-3-users), there’s another piece of the puzzle remaining: the SELinux user.

Part 3

### Introduction

In this final part of our SELinux tutorial, we will talk about SELinux users and how to fine-tune their access. We will also learn about SELinux error logs and how to make sense of the error messages.

## SELinux Users

SELinux users are different entities from normal Linux user accounts, including the root account. An SELinux user is not something you create with a special command, nor does it have its own login access to the server. Instead, SELinux users are defined in the policy that’s loaded into memory at boot time, and there are only a few of these users. The user names end with \_u, just like types or domain names end with \_t and roles end with \_r. Different SELinux users have different rights in the system and that’s what makes them useful.

The SELinux user listed in the first part of a file’s security context is the user that owns that file. This is just like you would see a file’s owner from a regular ls -l command output. A user label in a process context shows the SELinux user’s privilege the process is running with.

When SELinux is enforced, each regular Linux user account is mapped to an SELinux user account. There can be multiple user accounts mapped to the same SELinux user. This mapping enables a regular account to inherit the permission of its SELinux counterpart.

To view this mapping, we can run the semanage login -l command:

semanage login -l

In CentOS 7, this is what we may see:

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 \*

The first column in this table, “Login Name”, represents the local Linux user accounts. But there are only three listed here, you may ask, didn’t we create a few accounts in the second part of this tutorial? Yes, and they are represented by the entry shown as **default**. Any regular Linux user account is first mapped to the **default** login. This is then mapped to the SELinux user called unconfined\_u. In our case, this is the second column of the first row. The third column shows the multilevel security / Multi Category Security (MLS / MCS) class for the user. For now, let’s ignore that part and also the column after that (Service).

Next, we have the **root** user. Note that it’s not mapped to the “**default**” login, rather it has been given its own entry. Once again, root is also mapped to the unconfined\_u SELinux user.

system\_u is a different class of user, meant for running processes or daemons.

To see what SELinux users are available in the system, we can run the semanage user command:

semanage user -l

The output in our CentOS 7 system should look like this:

Labeling MLS/ MLS/

SELinux User Prefix MCS Level MCS Range SELinux Roles

guest\_u user s0 s0 guest\_r

root user s0 s0-s0:c0.c1023 staff\_r sysadm\_r system\_r unconfined\_r

staff\_u user s0 s0-s0:c0.c1023 staff\_r sysadm\_r system\_r unconfined\_r

sysadm\_u user s0 s0-s0:c0.c1023 sysadm\_r

system\_u user s0 s0-s0:c0.c1023 system\_r unconfined\_r

unconfined\_u user s0 s0-s0:c0.c1023 system\_r unconfined\_r

user\_u user s0 s0 user\_r

xguest\_u user s0 s0 xguest\_r

What does this bigger table mean? First of all, it shows the different SELinux users defined by the policy. We had seen users like unconfined\_u and system\_u before, but we are now seeing other types of users like guest\_u, staff\_u, sysadm\_u, user\_u and so on. The names are somewhat indicative of the rights associated with them. For example, we can perhaps assume that the sysadm\_u user would have more access rights than guest\_u.

To verify our guest, let’s look at the fifth column, SELinux Roles. If you remember from the first part of this tutorial, SELinux roles are like gateways between a user and a process. We also compared them to filters: a user may enter a role, provided the role grants it. If a role is authorized to access a process domain, the users associated with that role will be able to enter that process domain.

Now from this table we can see the unconfined\_u user is mapped to the system\_r and unconfined\_r roles. Although not evident here, SELinux policy actually allows these roles to run processes in the unconfined\_t domain. Similarly, user sysadm\_u is authorized for the sysadmr role, but guestu is mapped to guest\_r role. Each of these roles will have different domains authorized for them.

Now if we take a step back, we also saw from the first code snippet that the **default** login maps to the unconfinedu user, just like the root user maps to the unconfined\_u user. Since the \*\*default\_\*\* login represents any regular Linux user account, those accounts will be authorized for system\_r and unconfined\_r roles as well.

So what this really means is that any Linux user that maps to the unconfined\_u user will have the privileges to run any app that runs within the unconfined\_t domain.

To demonstrate this, let’s run the id -Z command as the root user:

id -Z

This shows the SELinux security context for **root**:

unconfined\_u:unconfined\_r:unconfined\_t:s0-s0:c0.c1023

So the root account is mapped to the unconfined\_u SELinux user, and unconfined\_u is authorized for the unconfined\_r role, which in turn is authorized to run processes in the unconfined\_t domain.

We suggest that you take the time now to start four new SSH sessions with the four users you created from separate terminal windows. This will help us switch between different accounts when needed.

* regularuser
* switcheduser
* guestuser
* restricteduser

Next, we switch to the terminal session logged in as the regularuser. If you remember, we created a number of user accounts in the [second tutorial](https://www.digitalocean.com/community/tutorials/an-introduction-to-selinux-on-centos-7-part-2-files-and-processes), and regularuser was one of them. If you have not already done so, open a separate terminal window to connect to your CentOS 7 system as regularuser. If we execute the same id -Z command from there, the output will look like this:

[regularuser@localhost ~]$ id -Z

unconfined\_u:unconfined\_r:unconfined\_t:s0-s0:c0.c1023

In this case, regulauser account is mapped to the unconfined\_u SELinux user account and it can assume the unconfined\_r role. The role can run processes in an unconfined domain. This is the same SELinux user/role/domain the root account also maps to. That’s because SELinux targeted policy allows logged in users to run in unconfined domains.

We had seen the list of a number of SELinux users before:

* **guest\_u**: This user doesn’t have access to X-Window system (GUI) or networking and can’t execute su / sudo command.
* **xguest\_u**: This user has access to GUI tools and networking is available via Firefox browser.
* **user\_u**: This user has more access than the guest accounts (GUI and networking), but can’t switch users by running su or sudo.
* **staff\_u**: Same rights as user\_u, except it can execute sudo command to have root privileges.
* **system\_u**: This user is meant for running system services and not to be mapped to regular user accounts.

## SELinux in Action 1: Restricting Switched User Access

To see how SELinux can enforce security for user accounts, let’s think about the regularuser account. As a system administrator, you now know the user has the same unrestricted SELinux privileges as the root account and you would like to change that. Specifically, you don’t want the user to be able to switch to other accounts, including the root account.

Let’s first check the user’s ability to switch to another account. In the following code snippet, the regularuser switches to the switcheduser account. We assume he knows the password for switcheduser:

[regularuser@localhost ~]$ su - switcheduser

Password:

[switcheduser@localhost ~]$

Next, we go back to the terminal window logged in as the **root** user and change regularuser’s SELinux user mapping. We will map regularuser to user\_u.

semanage login -a -s user\_u regularuser

So what are we doing here? We are adding (-a) the regularuser account to the SELinux (-s) user account user\_u. The change won’t take effect until regularuser logs out and logs back in.

Going back to regularuser’s terminal window, we first switch back from switcheduser:

[switcheduser@localhost ~]$ logout

Next the regularuser also logs out:

[regularuser@localhost ~]$ logout

We then open a new terminal window to connect as regularuser. Next, we try to change to switcheduser again:

[regularuser@localhost ~]$ su - switcheduser

Password:

This is what we see now:

su: Authentication failure

If we now run the id -Z command again to see the SELinux context for regularuser, we will see the output is quite different from what we saw before: regularuser is now mapped to user\_u.

[regularuser@localhost ~]$ id -Z

user\_u:user\_r:user\_t:s0

So where would you use such restrictions? You can think of an application development team within your IT organization. You may have a number of developers and testers in that team coding and testing the latest app for your company. As a system administrator you know developers are switching from their account to some of the high-privileged accounts to make ad-hoc changes to your server. You can stop this from happening by restricting their ability to switch accounts. (Mind you though, it still doesn’t stop them from logging in directly as the high-privileged user).

## SELinux in Action 2: Restricting Permissions to Run Scripts

Let’s see another example of restricting user access through SELinux. Run these commands from the **root** session.

By default, SELinux allows users mapped to the guest\_t account to execute scripts from their home directories. We can run the getsebool command to check the boolean value:

getsebool allow\_guest\_exec\_content

The output shows the flag is on.

guest\_exec\_content --> on

To verify its effect, let’s first change the SELinux user mapping for the guestuser account we created at the beginning of this tutorial. We will do it as the root user.

semanage login -a -s guest\_u guestuser

We can verify the action by running the semanage login -l command again:

semanage login -l

As we can see, guestuser is now mapped to the guest\_u SELinux user account.

Login Name SELinux User MLS/MCS Range Service

\_\_default\_\_ unconfined\_u s0-s0:c0.c1023 \*

guestuser guest\_u s0 \*

regularuser user\_u s0 \*

root unconfined\_u s0-s0:c0.c1023 \*

system\_u system\_u s0-s0:c0.c1023 \*

If we have a terminal window open as guestuser, we will log out from it and log back in a new terminal window as guestuser.

Next we will create an extremely simple bash script in the user’s home directory. The following code blocks first checks the home directory, then creates the file and reads it on console. Finally the execute permission is changed.

Verify that you are in the guestuser home directory:

[guestuser@localhost ~]$ pwd

/home/guestuser

Create the script:

[guestuser@localhost ~]$ vi myscript.sh

Script contents:

#!/bin/bash

echo "This is a test script"

Make the script executable:

chmod u+x myscript.sh

When we try to execute the script as guestuser, it works as expected:

[guestuser@localhost ~]$ ~/myscript.sh

This is a test script

Next we go back to the root terminal window and change the boolean setting allow\_guest\_exec\_content to off and verify it:

setsebool allow\_guest\_exec\_content off

getsebool allow\_guest\_exec\_content

guest\\_exec\\_content --> off

Going back to the console logged in as guestuser we try to run the script again. This time, the access is denied:

[guestuser@localhost ~]$ ~/myscript.sh

-bash: /home/guestuser/myscript.sh: Permission denied

So this is how SELinux can apply an additional layer of security on top of DAC. Even when the user has full read, write, execute access to the script created in their own home directory, they can still be stopped from executing it. Where would you need it? Well, think about a production system. You know developers have access to it as do some of the contractors working for your company. You would like them to access the server for viewing error messages and log files, but you don’t want them to execute any shell scripts. To do this, you can first enable SELinux and then ensure the corresponding boolean value is set.

We will talk about SELinux error messages shortly, but for now, if we are eager to see where this denial was logged we can look at the /var/log/messages file. Execute this from the root session:

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

The last two messages in the file in our CentOS 7 server show the access denial:

Aug 23 12:59:42 localhost setroubleshoot: SELinux is preventing /usr/bin/bash from execute access on the file . For complete SELinux messages. run sealert -l 8343a9d2-ca9d-49db-9281-3bb03a76b71a

Aug 23 12:59:42 localhost python: SELinux is preventing /usr/bin/bash from execute access on the file .

The message also shows a long ID value and suggests we run the sealert command with this ID for more information. The following command shows this (use your own alert ID):

sealert -l 8343a9d2-ca9d-49db-9281-3bb03a76b71a

And indeed, the output shows us greater detail about the error:

SELinux is preventing /usr/bin/bash from execute access on the file .

\*\*\*\*\* Plugin catchall\_boolean (89.3 confidence) suggests \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

If you want to allow guest to exec content

Then you must tell SELinux about this by enabling the 'guest\\_exec\\_content' boolean.

You can read 'None' man page for more details.

Do

setsebool -P guest\\_exec\\_content 1

\*\*\*\*\* Plugin catchall (11.6 confidence) suggests \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

...

It’s a large amount of output, but note the few lines at the beginning:

**SELinux is preventing /usr/bin/bash from execute access on the file .**

That gives us a pretty good idea where the error is coming from.

The next few lines also tell you how to fix the error:

If you want to allow guest to exec content

Then you must tell SELinux about this by enabling the 'guest\\_exec\\_content' boolean.

...

setsebool -P guest\\_exec\\_content 1

## SELinux in Action 3: Restricting Access to Services

In the [first part of this series](https://www.digitalocean.com/community/tutorials/an-introduction-to-selinux-on-centos-7-part-1-basic-concepts) we talked about SELinux roles when we introduced the basic terminology of users, roles, domains, and types. Let’s now see how roles also play a part in restricting user access. As we said before, a role in SELinux sits between the user and the process domain and controls what domains the user’s process can get into. Roles are not that important when we see them in file security contexts. For files, it’s listed with a generic value of object\_r. Roles become important when dealing with users and processes.

Let’s first make sure that the httpd daemon is not running in the system. As the root user, you can run the following command to make sure the process is stopped:

service httpd stop

Next, we switch to the terminal window we had logged in as restricteduser and try to see the SELinux security context for it. If you don’t have the terminal window open, start a new terminal session against the system and log in as the restricteduser account we had created at the beginning of this tutorial.

[restricteduser@localhost ~]$ id -Z

unconfined\_u:unconfined\_r:unconfined\_t:s0-s0:c0.c1023

So the account has the default behaviour of running as unconfined\_u user and having access to unconfined\_r role. However, this account does not have the right to start any processes within the system. The following code block shows that restricteduser is trying to start the httpd daemon and getting an access denied error:

[restricteduser@localhost ~]$ service httpd start

Redirecting to /bin/systemctl start httpd.service

Failed to issue method call: Access denied

Next we move back to the root user terminal window and make sure the restricteduser account has been added to the /etc/sudoers file. This action will enable the restricteduser account to use root privileges.

visudo

And then in the file, add the following line, save and exit:

restricteduser ALL=(ALL) ALL

If we now log out of the restricteduser terminal window and log back in again, we can start and stop the httpd service with sudo privileges:

[restricteduser@localhost ~]$ sudo service httpd start

We trust you have received the usual lecture from the local System

Administrator. It usually boils down to these three things:

#1) Respect the privacy of others.

#2) Think before you type.

#3) With great power comes great responsibility.

[sudo] password for restricteduser:

Redirecting to /bin/systemctl start httpd.service

The user can also stop the service now:

[restricteduser@localhost ~]$ sudo service httpd stop

Redirecting to /bin/systemctl stop httpd.service

That’s all very normal: system administrators give sudo access to user accounts they trust. But what if you want to stop this particular user from starting the httpd service even when the user’s account is listed in the sudoers file?

To see how this can be achieved, let’s switch back to the root user’s terminal window and map the restricteduser to the SELinux user\_r account. This is what we did for the regularuser account in another example.

semanage login -a -s user\_u restricteduser

Going back to restricteduser’s terminal window, we log out and log back in again in a new terminal session as restricteduser.

Now that restricteduser has been restricted to user\_u (and that means to role user\_r and domain user\_t), we can verify its access using the seinfo command from our root user’s window:

seinfo -uuser\_u -x

The output shows the roles user\_u can assume. These are object\_r and user\_r:

user\_u

default level: s0

range: s0

roles:

object\_r

user\_r

Taking it one step further, we can run the seinfo command to check what domains the user\_r role is authorized to enter:

seinfo -ruser\_r -x

There are a number of domains user\_r is authorized to enter:

user\_r

Dominated Roles:

user\_r

Types:

git\_session\_t

sandbox\_x\_client\_t

git\_user\_content\_t

virt\_content\_t

policykit\_grant\_t

httpd\_user\_htaccess\_t

telepathy\_mission\_control\_home\_t

qmail\_inject\_t

gnome\_home\_t

...

...

But does this list show httpd\_t as one of the domains? Let’s try the same command with a filter:

seinfo -ruser\_r -x | grep httpd

There are a number of httpd related domains the role has access to, but httpd\_t is not one of them:

httpd\_user\_htaccess\_t

httpd\_user\_script\_exec\_t

httpd\_user\_ra\_content\_t

httpd\_user\_rw\_content\_t

httpd\_user\_script\_t

httpd\_user\_content\_t

Taking this example then, if the restricteduser account tries to start the httpd daemon, the access should be denied because the httpd process runs within the httpd\_t domain and that’s not one of the domains the user\_r role is authorized to access. And we know user\_u (mapped to restricteduser) can assume user\_r role. This should fail even if the restricteduser account has been granted sudo privilege.

Going back to the restricteduser account’s terminal window, we try to start the httpd daemon now (we were able to stop it before because the account was granted sudo privilege):

[restricteduser@localhost ~]$ sudo service httpd start

The access is denied:

sudo: PERM\_SUDOERS: setresuid(-1, 1, -1): Operation not permitted

So there is another example of how SELinux can work like a gatekeeper.

## SELinux Audit Logs

As a system administrator, you would be interested to look at the error messages logged by SELinux. These messages are logged in specific files and they can provide detailed information about access denials. In a CentOS 7 system you can look at two files:

* /var/log/audit/audit.log
* /var/log/messages

These files are populated by the auditd daemon and the rsyslogd daemon respectively. So what do these daemons do? The man pages say the auditd daemon is the userspace component of the Linux auditing system and rsyslogd is the system utility providing support for message logging. Put simply, these daemons log error messages in these two files.

The /var/log/audit/audit.log file will be used if the auditd daemon is running. The /var/log/messages file is used if auditd is stopped and rsyslogd is running. If both the daemons are running, both the files are used: /var/log/audit/audit.log records detailed information while an easy-to-read version is kept in /var/log/messages.

### Deciphering SELinux Error Messages

We looked at one SELinux error message in an earlier section (refer to “SELinux in Action 2: Restricting Permissions to Run Scripts”). We were then using the grep command to sift through /var/log/messages file. Fortunately SELinux comes with a few tools to make life a bit easier than that. These tools are not installed by default and require installing a few packages, which you should have installed in the first part of this tutorial.

The first command is ausearch. We can make use of this command if the auditd daemon is running. In the following code snippet we are trying to look at all the error messages related to the httpd daemon. Make sure you are in your root account:

ausearch -m avc -c httpd

In our system a number of entries were listed, but we will concentrate on the last one:

----

time->Thu Aug 21 16:42:17 2014

...

type=AVC msg=audit(1408603337.115:914): avc: denied { getattr } for pid=10204 comm="httpd" path="/www/html/index.html" dev="dm-0" ino=8445484 scontext=system\_u:system\_r:httpd\_t:s0 tcontext=unconfined\_u:object\_r:default\_t:s0 tclass=file

Even experienced system administrators can get confused by messages like this unless they know what they are looking for. To understand it, let’s take apart each of the fields:

* type=AVC and avc: AVC stands for Access Vector Cache. SELinux caches access control decisions for resource and processes. This cache is known as the Access Vector Cache (AVC). That’s why SELinux access denial messages are also known as “AVC denials”. These two fields of information are saying the entry is coming from an AVC log and it’s an AVC event.
* denied { getattr }: The permission that was attempted and the result it got. In this case the get attribute operation was denied.
* pid=10204. This is the process id of the process that attempted the access.
* comm: The process id by itself doesn’t mean much. The comm attribute shows the process command. In this case it’s httpd. Immediately we know the error is coming from the web server.
* path: The location of the resource that was accessed. In this case it’s a file under /www/html/index.html.
* dev and ino: The device where the target resource resides and its inode address.
* scontext: The security context of the process. We can see the source is running under the httpd\_t domain.
* tcontext: The security context of the target resource. In this case the file type is default\_t.
* tclass: The class of the target resource. In this case it’s a file.

If you look closely, the process domain is httpd\_t and the file’s type context is default\_t. Since the httpd daemon runs within a confined domain and SELinux policy stipulates this domain doesn’t have any access to files with default\_t type, the access was denied.

We have already seen the sealert tool. This command can be used with the id value of the error message logged in the /var/log/messages file.

In the following code snippet we again grep through the the /var/log/message file for SELinux related errors:

cat /var/log/messages | grep "SELinux is preventing"

In our system, we look at the very last error. This is the error that was logged when our restricteduser tried to run the httpd daemon:

...

Aug 25 11:59:46 localhost setroubleshoot: SELinux is preventing /usr/bin/su from using the setuid capability. For complete SELinux messages. run sealert -l e9e6c6d8-f217-414c-a14e-4bccb70cfbce

As suggested, we ran sealert with the ID value and were able to see the details (your ID value should be unique to your system):

sealert -l e9e6c6d8-f217-414c-a14e-4bccb70cfbce

SELinux is preventing /usr/bin/su from using the setuid capability.

...

Raw Audit Messages

type=AVC msg=audit(1408931985.387:850): avc: denied { setuid } for pid=5855 comm="sudo" capability=7 scontext=user\_u:user\_r:user\_t:s0 tcontext=user\_u:user\_r:user\_t:s0 tclass=capability

type=SYSCALL msg=audit(1408931985.387:850): arch=x86\_64 syscall=setresuid success=no exit=EPERM a0=ffffffff a1=1 a2=ffffffff a3=7fae591b92e0 items=0 ppid=5739 pid=5855 auid=1008 uid=0 gid=1008 euid=0 suid=0 fsuid=0 egid=0 sgid=1008 fsgid=0 tty=pts2 ses=22 comm=sudo exe=/usr/bin/sudo subj=user\_u:user\_r:user\_t:s0 key=(null)

Hash: su,user\_t,user\_t,capability,setuid

We have seen how the first few lines of the output of sealert tell us about the remediation steps. However, if we now look near the end of the output stream, we can see the “Raw Audit Messages” section. The entry here is coming from the audit.log file, which we discussed earlier, so you can use that section to help you interpret the output here.

## Multilevel Security

Multilevel security or **MLS** is the fine-grained part of an SELinux security context.

So far in our discussion about security contexts for processes, users, or resources we have been talking about three attributes: SELinux user, SELinux role, and SELinux type or domain. The fourth field of the security context shows the sensitivity and optionally, the category of the resource.

To understand it, let’s consider the security context of the FTP daemon’s configuration file:

ls -Z /etc/vsftpd/vsftpd.conf

The fourth field of the security context shows a sensitivity of s0.

-rw-------. root root system\_u:object\_r:etc\_t:s0 /etc/vsftpd/vsftpd.conf

The sensitivity is part of the hierarchical multilevel security mechanism. By hierarchy, we mean the levels of sensitivity can go deeper and deeper for more secured content in the file system. Level 0 (depicted by s0) is the lowest sensitivity level, comparable to say, “public.” There can be other sensitivity levels with higher s values: for example, internal, confidential, or regulatory can be depicted by s1, s2, and s3 respectively. This mapping is not stipulated by the policy: system administrators can configure what each sensitivity level mean.

When a SELinux enabled system uses MLS for its policy type (configured in the /etc/selinux/config file), it can mark certain files and processes with certain levels of sensitivity. The lowest level is called “current sensitivity” and the highest level is called “clearance sensitivity”.

Going hand-in-hand with sensitivity is the category of the resource, depicted by c. Categories can be considered as labels assigned to a resource. Examples of categories can be department names, customer names, projects etc. The purpose of categorization is to further fine-tune access control. For example, you can mark certain files with confidential sensitivity for users from two different internal departments.

For SELinux security contexts, sensitivity and category work together when a category is implemented. When using a range of sensitivity levels, the format is to show sensitivity levels separated by a hyphen (for example, s0-s2). When using a category, a range is shown with a dot in between. Sensitivity and category values are separated by a colon (:).

Here is an example of sensitivity / category pair:

user\_u:object\_r:etc\_t:s0:c0.c2

There is only one sensitivity level here and that’s s0. The category level could also be written as c0-c2.

So where do you assign your category levels? Let’s find the details from the /etc/selinux/targeted/setrans.conf file:

cat /etc/selinux/targeted/setrans.conf

#

# Multi-Category Security translation table for SELinux

#

#

# Objects can be categorized with 0-1023 categories defined by the admin.

# Objects can be in more than one category at a time.

# Categories are stored in the system as c0-c1023. Users can use this

# table to translate the categories into a more meaningful output.

# Examples:

# s0:c0=CompanyConfidential

# s0:c1=PatientRecord

# s0:c2=Unclassified

# s0:c3=TopSecret

# s0:c1,c3=CompanyConfidentialRedHat

s0=SystemLow

s0-s0:c0.c1023=SystemLow-SystemHigh

s0:c0.c1023=SystemHigh

We won’t go into the details of sensitivities and categories here. Just know that a process is allowed read access to a resource only when its sensitivity and category level is higher than that of the resource (i.e. the process domain dominates the resource type). The process can write to the resource when its sensitivity/category level is less than that of the resource.

### Conclusion

We have tried to cover a broad topic on Linux security in the short span of this three-part-series. If we look at our system now, we have a simple Apache web server installed with its content being served from a custom directory. We also have an FTP daemon running in our server. There were a few users created whose access have been restricted. As we went along, we used SELinux packages, files, and commands to cater to our security needs. Along the way we also learned how to look at SELinux error messages and make sense of them.

Entire books have been written on the SELinux topic and you can spend hours trying to figure out different packages, configuration files, commands, and their effects on security. So where do you go from here?

One thing I would do is caution you not to test anything on a production system. Once you have mastered the basics, start playing with SELinux by enabling it on a test replica of your production box. Make sure the audit daemons are running and keep an eye on the error messages. Check any denials preventing services from starting. Play around with the boolean settings. Make a list of possible steps for securing your system, like creating new users mapped to least-privilged SELinux accounts or applying the right context to non-standard file locations. Understand how to decipher an error log. Check the ports for various daemons: if non-standard ports are used, make sure they are correctly assigned to the policy.