An Introduction to SELinux on CentOS 7 – Part 2: Files and Processes

17

Posted September 5, 2014 101.3k SECURITY CENTOS

By: Sadequl Hussain

Introduction

In the first part of our SELinux series, 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*.

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

In this tutorial, we will be running the commands as the root user unless otherwise stated. If you don't have access to the root account and use another account with sudo privileges, you need to precede the commands with the `sudo` keyword.

Creating Test User Accounts

First, let's create four user accounts to demonstrate SELinux capabilities as we gı

SCROLL TO TOP

- guestuser

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

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.

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:

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.

```
$ ls -Z /home
```

```
drwx------. guestuser     guestuser     unconfined_u:object_r:user_home_dir_t:s0
drwx------. root          root          system_u:object_r:lost_found_t:s0 lost+founc
drwx------. regularuser   regularuser   unconfined_u:object_r:user_home_dir_t:s0
drwx------. restricteduser restricteduser unconfined_u:object_r:user_home_dir_t:s0
drwx------. switcheduser  switcheduser  unconfined_u:object_r:user_home_dir_t:s0
drwx------. sysadmin      sysadmin      unconfined_u:object_r:user_home_dir_t:s0
```

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 ?        (
system_u:system_r:httpd_t:s0            apache      7127    7126    0 16:50 ?        (
system_u:system_r:httpd_t:s0           apache      7131    7126    0 16:50 ?        0
```

```
6:54 ?        0
16:57 pts/0 00
```

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

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,

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/*
```

```
-rw-r--r--. root root unconfined_u:object_r:httpd_sys_content_t:s0 /var/www/html/inde
```

od 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 ap
```

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.

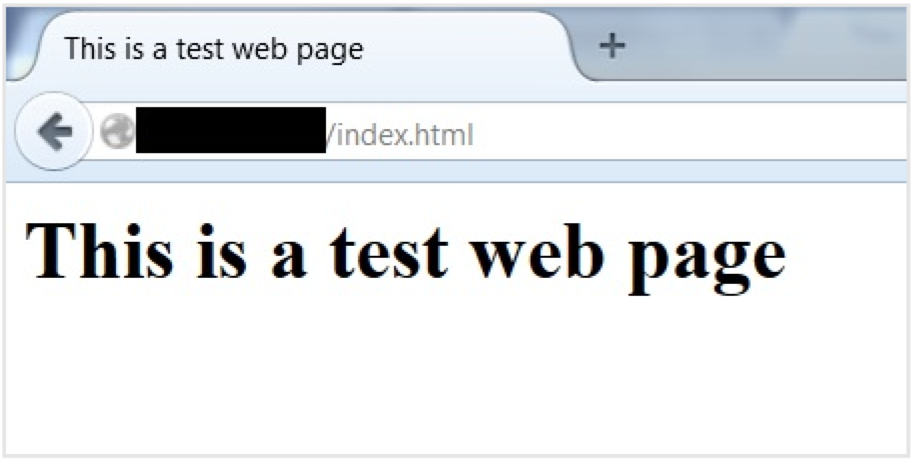
Going one step further, let's first modify the web page (`/var/www/html/index.html`). Edit the file to contain this content:

```
<html>
    <title>
        This is a test web page
    </title>
    <body>
        <h1>This is a test web page</h1>
```

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

```
$ chmod -R 755 /var/www
$ service httpd restart
```

We will then try to access it from a browser:



This is a test web page

> **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 on the topic which you can use.

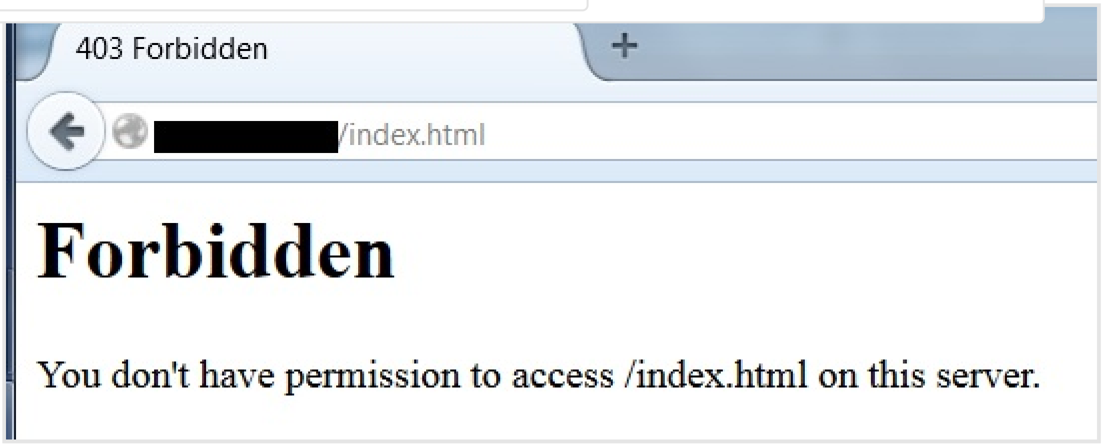
So far so good. The httpd daemon is authorized to access a particular type of file and we can see it when accessing via the browser. Next, let's make things a little different by changing the context of the file. We will use the `chcon` command for it. The `--type` flag for the command allows us to specify 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/
```

Next, when we try to access the web page (i.e. the httpd daemon tries to read the file), you may get



Forbidden

You don't have permission to access /index.html on this server.

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->uncc
```

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.

Similarly, if we have a directory with a type of "some_context_t", any file or directory created under it

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/
```

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
```

```
-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:

When we check the moved file's context, we see that the **var_t** context has been preserved under

```
$ ls -Z /etc/index.html
```

```
-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
```

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

```
$ ls -Z /www/
```

```
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/
```

also have to relax the access rights for this directory.

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
```

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.

But how do we do that?

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
```

```
...
/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.

ne `/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(/.*)?"
```

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
```

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
```

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_
restorecon reset /www/html context unconfined_u:object_r:default_t:s0->unconfined_u:ob
restorecon reset /www/html/index.html context unconfined_u:object_r:default_t:s0->unco
```

There is a nifty tool called `matchpathcon` that can help troubleshoot context-related problems.

s listed under

st this with the

`/www/html/index.html` file. We will use the `-V` flag that verifies the context:

```
$ matchpathcon -V /www/html/index.html
```

The `matchpathcon` output should show that the context is verified.

```
/www/html/index.html verified.
```

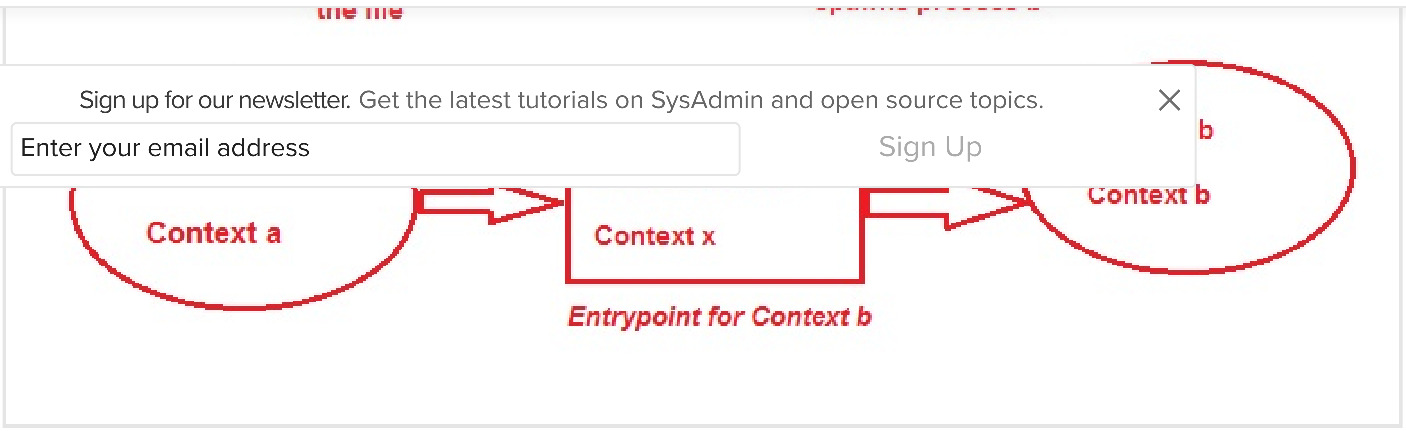
For an incorrectly labelled file, the message will say what the context should be:

```
/www/html/index.html has context unconfined_u:object_r:default_t:s0, should be system_
```

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
```

```
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
```

Shows us:

```
-rwxr-xr-x. root root system_u:object_r:ftpd_exec_t:s0 /usr/sbin/vsftpd
```

Checking the process:

Shows us:

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
```

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 } ;
```

```
sesearch -s ftpd_t -t ftpd_exec_t -c file -p entrypoint -Ad
```

```
Found 1 semantic av rules:
   allow ftpd_t ftpd_exec_t : file { ioctl read getattr lock execute execute_no_trans
```

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
```

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

Share

Editor:
Sharon Campbell

Tutorial Series

An Introduction to SELinux on CentOS 7

SELinux is a Linux kernel security module that brings heightened security for Linux systems. This series introduces basic SELinux terms and concepts, demonstrating how to enable SELinux, change security settings, check logs, and resolve errors. After completing all three steps, you will have a working CentOS 7 system with SELinux enabled, with four users added with differing degrees of access.

Show Tutorials

Related Tutorials

How To Protect Your Server Against the Meltdown and Spectre Vulnerabilities

How To Protect Your Linux Server Against the GHOST Vulnerability

How to Protect Your Server Against the Shellshock Bash Vulnerability

How to Protect Your Server Against the Heartbleed OpenSSL Vulnerability

How to Install TrueCrypt (CLI) on Linux

7 Comments

Leave a comment...

Log In to Comment

0 Great article! Since there seems to be no possibility for me to edit this article: A few spelling mistakes I noticed: 1) vart should probably replaced with vart *2) anoher => another 3) simplify you backup =>* *tpdexect 7)*

SadequlHussain *July 12, 2015*
0 Thanks, pheanex! yes indeed they were spelling mistakes.

lizz *May 22, 2015*
0 Your introduction articles on SElinux have been a huge help -- really enjoying them. Can you help me understand the difference between "open" and "read" in SElinux rule permissions? I'm sure it's very nuanced but I'm having trouble understanding what one allows that the other wouldn't.

SadequlHussain *July 12, 2015*
0 Hi lizz, "read" would mean a permission where the directory contents can be listed/parsed and "open" would mean actually being able to load the file in that directory in memory and read its contents.

iamhere64 *December 17, 2015*
0 Thank you! Now this is an example of taking something difficult and making it easy!

vstoykov *December 11, 2016*
0 You forgot to escape the "_" symbol many times in this article. It looks like you are using markup language that uses the "_" symbol for *italic* formatting.

You should put the "\" symbol before the "_" symbol.

For example: ftpd*exec*t and "ftpd_exec_t" should be: "ftpd_exec_t" (and it looks in your source code: "ftpd_exec_t").

andrewryan *October 11, 2017*
0

Sign up for our newsletter. Get the latest tutorials on SysAdmin and open source topics.

Enter your email address

Sign Up

Community Tutorials Questions Projects Tags Newsletter RSS

Distros & One-Click Apps Terms, Privacy, & Copyright Security Report a Bug Write for DOnations Shop