An Introduction to SELinux on CentOS 7 – Part 3: Users

Posted September 5, 2014 62.1k SECURITY SYSTEM TOOLS CENTOS

By: Sadequl Hussain

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.

> **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.

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

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:

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
staff_u         user       s0         s0-s0:c0.c1023                 staff_r sysadm_r
sysadm_u        user       s0         s0-s0:c0.c1023                 sysadm_r
system_u        user       s0         s0-s0:c0.c1023                 system_r unconfi
unconfined_u    user       s0         s0-s0:c0.c1023                 system_r unconfi
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 sysadm*r role, but guest*u 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 unconfined*u 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

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, 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.

- **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:

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.

```
[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
```


ime, the

```
[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 exe
Aug 23 12:59:42 localhost python: SELinux is preventing /usr/bin/bash from execute acc
```

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   ******************
```



```
tent' boolean.
```

```
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 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

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:
```

```
[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
```

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 s not one of

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: [illegible] ift through

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=
```

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.

lt_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
```

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="s


type=SYSCALL msg=audit(1408931985.387:850): arch=x86_64 syscall=setresuid success=no e

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

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

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.

It will all come together with time and practice. :)

By: Sadequl Hussain

Editor:
Sharon Campbell

Tutorial Series

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

8 Comments

Leave a comment...

Log In to Comment

defensora *April 1, 2015*

0 Excellent introduction to a difficult to explain topic! This cleared up a lot of questions I had and I was able to actually successfully configure some of the options explained, with no previous experience with SELinux.

amellya *April 14, 2015*

0 May I ask you a question,
I'm currently studying about bell lapadula model for my research and I need to implementation it as an example. Can BLP model implement in SELinux? If it can, can you explain to me, how to do it, because I'm not really sure about how to implement it in SELinux?

Thank you so much..
If you can help me, I really appreciate it..

SadequlHussain *July 12, 2015*

0 Hi amellya, Unfortunately I haven't worked with Bell Lapadula Model and won't be able to comment on this. However, as they say, there's only one way to find out when you don't know the answer - do it. I would recommend you roll out a non-prod dedicated environment and tighten its security with SELinux first and then see if BLP can be implemented there

SadequlHussain *July 12, 2015*

0 Hi again, amellya, just to correct my previous post in answer to your query, I believe I was wrong when saying I haven't worked with BLP - because SELinux *is* built on top of BLP and hence it's implemented by SELinux by default. You can have a look at this knowledgebase article from CentOS documentation on MLS:

https://www.centos.org/docs/5/html/Deployment_Guide-en-US/sec-mls-ov.html

---

nycJacob *September 11, 2015*

0 I think there is a small step left out. I could start httpd as restricted user even after the se user change. I had to log out and then log in for the restriction to take effect and deny sudo access to httpd start

---

BilalArif *December 5, 2015*

0 Well I have question.
I want to use this for monitoring **Processes Running inside Docker Containers**, I search about this, this is possible and already in use. But unfortunately I could not get clear image or direction how to make this in action. I would be greatly thankful if you can explain how to do this.

---

oogway *August 5, 2016*

0 Hi Sadequl,

Thank you very much for such a good write up!

I noticed an issue here though. I replicated your steps to try to restrict a user from invoking a script. One thing I noticed though is that whenever a restricted user (guestuser) runs the script via relative/absolute path, it gets denied. That's good. But if I try to invoke it directly via bash or sh, it gets executed which seems unexpected :(. It is on CentOS-7. By the way, here's the output:

[root@puppetmaster ~]# semanage user -l | grep '<guest'
guest*u user s0 s0 guestr*
[root@puppetmaster ~]#

[root@puppetmaster ~]# semanage login -l | grep '<guestuser'
guestuser guest_u s0 *
[root@puppetmaster ~]#

[root@puppetmaster ~]# getsebool guest*exec*content
guest*exec*content --> off
[root@puppetmaster ~]#

```
[guestuser@puppetmaster ~]$ id -Z
guestu:guestr:guest_t:s0
[guestuser@puppetmaster ~]$ cat test.sh

!/bin/bash
echo "This is a test"
[guestuser@puppetmaster ~]$
[guestuser@puppetmaster ~]$ ls -lZ test.sh
-rwxrwxr-x. guestuser guestuser unconfinedu:objectr:userhomet:s0 test.sh
[guestuser@puppetmaster ~]$
[guestuser@puppetmaster ~]$ ./test.sh
-bash: ./test.sh: Permission denied
[guestuser@puppetmaster ~]$
[guestuser@puppetmaster ~]$

---Here's the troubling part ---

[guestuser@puppetmaster ~]$ bash test.sh
This is a test
[guestuser@puppetmaster ~]$
[guestuser@puppetmaster ~]$ sh test.sh
This is a test
[guestuser@puppetmaster ~]$

[guestuser@puppetmaster ~]$ ~/test.sh
-bash: /home/guestuser/test.sh: Permission denied
[guestuser@puppetmaster ~]$
```

Thank you very much!

---

digitalis *February 2, 2018*

0 Hi

Thanks for the useful tutorial on a tricky subject.

I'm having the same problem as oogway:

```
[guestuser@centos7-restore-001 ~]$ ./myscript.sh
-bash: ./myscript.sh: 許可がありません
[guestuser@centos7-restore-001 ~]$ bash myscript.sh
This is a test script
```

Please can you tell us why this is?