Lab 10:

SELinux

Linux Server Security  
 2024-2025

© Daan Pareit

## Introduction

# Lab concept

During this lab, we will explore Security Enhanced Linux (SELinux).

# Learning goals

* Understanding SELinux
* SELinux and a web server
* SELinux and PAM

# Practicalities and prerequisites

You’ll need the following:

* Your RHEL VM and the pfSense VM as installed during previous lab(s)

## SELinux Basics

On RHEL, SELinux is installed by default. Let’s have a look at this machine.

* Perform getenforce. Consult the output. What state is your machine currently in? Consult the man page to recall the other possible states. (And note the author of this tool, don’t make him weep 😉)

A black background with white text

Description automatically generated

It is currently in **Enforcing** state. The other state are permissive, or disabled.

The author is Dan Walsh.

A screenshot of a computer program

Description automatically generated

* Perform setenforce 0. Consult the new state with getenforce. What SELinux state did your machine changed to? Did you disable SELinux this way? Consult the setenforce man page.

A screen shot of a computer code

Description automatically generated

Now we are in **Permissive** mode. A screen shot of a computer

Description automatically generated

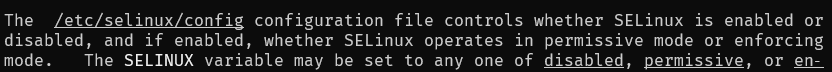
I did not disable it, I just changed the state it is currently running in (so the different mode of operation, which has other functions).

* Execute sestatus. Apart from the current state/mode, you can also find the mode which is specified in the “config file” (and which is thus the state that is applied after reboot). Look in the man page of “selinux” itself to find this main config file of SELinux.

A screen shot of a computer screen

Description automatically generated

As far as I can see, it should be **enforcing** mode every time I boot.



A screenshot of a computer program

Description automatically generated

Looks like I was right.

* The loaded policy is also mentioned with sestatus. Which is the policy? Verify that the ‘mode from config file’ and ‘loaded policy name’ mentioned by sestatus match with the settings in the config file.

The Policy name : **targeted**

Mode from config file : **enforcing**

Looks alright to me

* Have a look at the lab exercises concerning PAM. There, we wanted to know if su and sshd were implementing PAM. This was done by looking at the dynamic libraries which these binaries were using. Similarly, for ‘TCP wrapper’ capabilities we checked if daemons were linked against a certain library. Now, again, use the same approach to identify if su, sshd and ls implement SELinux. Which ones are linked against the libraries of PAM, SELinux and/or TCP wrapper?

Magic command : **ldd $(which <command>)**

For PAM – libpam.so

For SELinux – libselinux.so

For TCP Wrapper – libwrap.so

**su –** can work with PAM

**sshd** – PAM, SELinux

**ls** – SELinux

* Now look in the man page of ls on how to get SELinux context information (‘labels’) of the files in your home directory.

A black background with white text

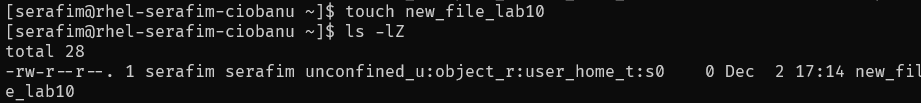
Description automatically generated

A screen shot of a computer

Description automatically generated

**ls -lZ**

* Create a new file in your home directory. Did it get a security context?



Yes it did.

* Remember that in /etc/selinux every security policy has a directory containing all security rules and all rules on how to label files, processes etc. with security contexts. Check the contexts/files/file\_contexts.homedirs file of the policy that is used on your RHEL. Can you find the expression which is responsible for applying the security context to the files in your home folder?

Directory is - /etc/selinux/targeted/contexts/files

What in the mother of god  
It looks confusing, so I will assume this one at least should work:

**/home/[^/]+/.+**

Seems to be right according to the Quiz, btw.



* Let’s see how you can change the SELinux state temporarily on a reboot. Reboot the RHEL machine. In the grub menu, edit the boot entry (follow instructions mentioned at the bottom of the grub screen). Go to the ‘linux /vmlinuz….’ line to boot with SELinux in *permissive* mode (cfr slides). What option do you need to append to this line? After booting, verify with getenforce you are indeed in permissive mode. Also check the /proc/cmdline file to verify you indeed booted your linux kernel with the extra option. Now, reboot once again normally, you’ll see you’re back into RHEL default selinux enforcing mode.

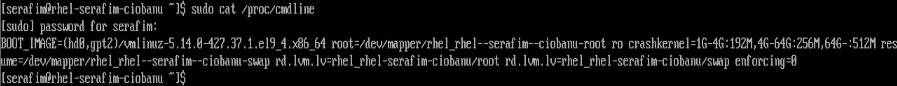
A screen shot of a computer error

Description automatically generated

The command is **enforcing=0**

**A screen shot of a computer

Description automatically generated**

****

**After reboot:**

**A screen shot of a computer

Description automatically generated**

## Running an Apache2 webserver on RHEL with SELinux

Before further looking at SELinux, we’ll quickly install a web server. We can then use this web server as an example use case to configure SELinux for. Supplementary, the web server installation also implies a small recap exercise on systemd and on configuring firewalld, the firewall running on RHEL.

# Installing Apache2

* On RHEL, install the Apache2 web server. (Use dnf search to find the package.)

**sudo dnf install httpd**

* It’s now available on your RHEL as the “httpd” systemd service. Start its service to be active and enable the service so it automatically starts when the VM starts (remember systemctl). What target now includes the httpd service (recall: because of the ‘WantedBy’ in its config file, a symlink will have been created in its “.wants” directory)?

**sudo systemctl enable httpd**

**sudo systemctl start httpd**

**A screenshot of a computer program

Description automatically generated**

**Sudo systemctl cat httpd.service**

**Multi-user.target I believe.**

* Verify your Apache is running, by doing a curl localhost on your RHEL machine, which should print the html start page on your console. Using ss -tulpn should also show that Apache is listening on port 80 (only with sudo the associated process is shown).

A computer screen with white text

Description automatically generated A computer screen with white text

Description automatically generated

Curl to localhost also works (dont show becaus the output is too big)s

* Now, use your web browser on your laptop to surf to your RHEL: http://192.168.11.20 . It wouldn’t work yet, because firewalld, the firewall of RHEL, isn’t allowing this service yet. We’ll solve that in the next section.

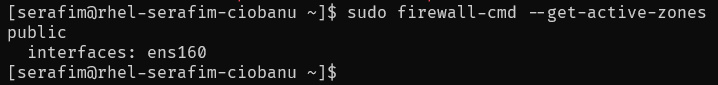
Indeed

A black background with white text

Description automatically generated

# Configuring firewalld for the web server

* What was the user tool to configure firewalld again (cfr lab on ‘Linux firewalling’)? Check what’s the active zone and get the info for this zone. Verify that the http service is indeed not listed.



Firewall-cmd (the most horrible firewalling tool)

A screen shot of a computer

Description automatically generated

sudo firewall-cmd –zone=public –llist-all

* Now use this tool to add the ‘http’ and ‘https’ service to the ‘public’ zone, to allow surfing to your RHEL machine (<https://firewalld.org/documentation/howto/>). Don’t forget (1) to add the service ‘permanently’ to survive reboots and (2) to reload your config, for your changes to take effect immediately.

sudo firewall-cmd --zone=public --add-service=http –permanent

sudo firewall-cmd --zone=public --add-service=https –permanent

sudo firewall-cmd –reload

sudo firewall-cmd –zone=public –list-all

A screen shot of a computer

Description automatically generated

* Verify that now it is possible to surf with your web browser on your laptop to your RHEL machine.

Screens screenshot of a computer

Description automatically generated

It works!

# Inspecting SELinux context for the web server’s process and files

We will now have a look at how the Apache2 web server operates in compliance with the Mandatory Access Control of SELinux. Let’s see how the security context of Apache’s process(es) relate to the security context of the files it has access to.

* We are using the default SELinux ‘targeted’ policy. We want to know if the targeted policy applies to our web server, in other words: if it is ‘confined’. Therefore, let’s first list the unconfined domain(s). This can be obtained by querying all SELinux components with the “unconfined\_domain\_type” attribute with the seinfo tool (install the setools-console package). Look at the seinfo man page on how to do so. Is ‘unconfined\_t’ the only type for unconfined domains or are other types listed as being unconfined as well?

I can barely understand what the question wants from me, but I will do my best to answer it:

A screen shot of a computer

Description automatically generated

It looks like it is not only **unconfined\_t** for unconfined domains, and it looks like there are others?

Or maybe it is safer to say that it is the only one, because then we have unrelated services. and unconfined\_service\_t

* Search for the httpd processes of apache2 (combine ps and grep) on your system. What is the security context (label) of these processes? What is the *domain (type)* of the security context (label) of these processes (cfr lecture slides)?

A screen shot of numbers and symbols

Description automatically generated

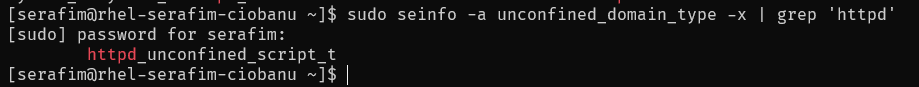
The label is apparently this whole thing - **system\_u:system\_r:httpd\_t:s0**

Domain (type) – **httpd\_t**

It is not included into the unconfined domains

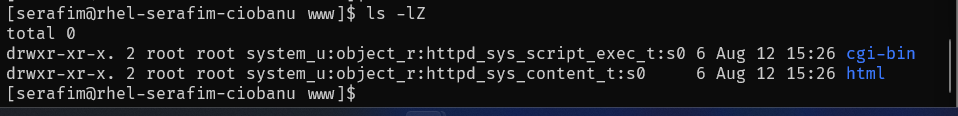
* We want to know if this domain (of the httpd processes) is ‘confined’ and thus subject to the MAC of SELinux. Figure this out by looking whether the domain of our apache2 processes is not included in the list of the unconfined domains, which you obtained earlier with seinfo. Check if this is the case.

It does not look like it is there (If it would be there, then it would be the name as it is, right?)



* HTML-pages which are being served by the Apache2 web server are located at the /var/www/html folder. What is the type of the security context (label) of this folder?

**httpd\_sys\_content\_t (?)**

****

**Adding a file there gives the same type to it**

****

* You’ve found the type/domain of the security context for (1) the httpd process and (2) the /var/www/html folder. How does (the targeted policy of) SELinux define that httpd has access to those files? Use sesearch to find the policy rule which allows the (domain of the) httpd process to access the (security context type of the) /var/www directory .

**sesearch --allow -s httpd\_t -t httpd\_sys\_content\_t**

This command searches for **allow** rules that permit the **httpd\_t** domain to access the **httpd\_sys\_content\_t type**.

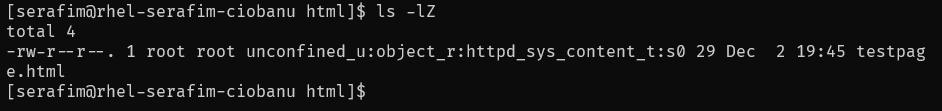


In a nutshell, it just creates rules that says “for this type, please provide this access rules”.

# Creating/manipulating web content at the default location

So, according to what you found in the previous section, you now know that the httpd process is allowed to access the /var/www/html folder. Let’s verify if this works as it should.

* Create a “testpage.html” file within /var/www/html with simple HTML code, e.g.:  
   "<h1>This is a Test Page</h1>"  
  What is the SELinux type this file was automatically labeled with?



**httpd\_sys\_content\_t**

* Look at the /etc/selinux/targeted/contexts/files/file\_contexts config file of the targeted policy and verify that this type is indeed set to be automatically applied to the /var/www subfolders (on what line did you find this?). (Hint: nl tool for line numbering)

For god sake this lab is so exhausting ☹



Looks like this is the line that applies the security context to all the /var/www subfolders

* Surf with your web browser on your laptop to this ‘testpage.html’ page, to find out it works.

A screenshot of a computer

Description automatically generated

* Now let’s change the SELinux type of testpage.html with the chcon tool to ‘default\_t’ .

**sudo chcon -t default\_t testpage.html** (?)

A black screen with white text

Description automatically generated

* The ‘default\_t’ type is not allowed to be accessed by the domain of the httpd processes (verify this again with sesearch). Surf again to testpage.html and see what happens.

A screenshot of a computer

Description automatically generated

I actually expected it to be forbidden

* Set SELinux to permissive mode and surf to testpage.html again. That should work again. However, the permissive mode is a ‘what if’ mode and should log that you’ll have no access in enforcing mode. Verify this by inspecting the audit log:  
  sudo grep permissive /var/log/audit/audit.log

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

Indeed it is.

* Now, set SELinux in enforcing mode again.

A black background with white text

Description automatically generated

* Use restorecon to fix the context of testpage.html to its default value, according to labeling rules. Surf again to testpage.html and see it works again in enforcing mode.

A screenshot of a computer screen

Description automatically generated

# Creating/manipulating web content at a different location

Now, we want our web server also to serve some web pages, located in a different folder (for whatever reason). Let’s see how to make SELinux comply.

* In the root of your file system tree, create a directory ‘/mywebstuff’. (We’ll want files in this directory to be served by our web server.)

sudo mkdir /mywebstuff

* In /var/www/html create a symbolic link ‘personal’ to this directory. (Use ln with the appropriate options and arguments.)

sudo ln -s /mywebstuff /var/www/html/personal

* Create an ‘index.html’ with some text in the ‘/mywebstuff’ directory. What is the security context of this index.html file?’

A computer screen with white text

Description automatically generated

default\_t

* Surf to <http://192.168.11.20/personal> . Does it work?

A screenshot of a computer

Description automatically generated

Nope, it will not

* Now, we want to create a labelling rule to label all files within ‘/mywebstuff’ with the same type as the /var/www/html directory instead, to allow our Apache2 process to access these files. We use the semanage command for this purpose to add a file context rule.  
  (Note the command can take a while to execute.)  
  sudo semanage fcontext -a -t httpd\_sys\_content\_t "/mywebstuff(/.\*)?"
* Verify that this labeling rule is now indeed created in the following file:  
  /etc/selinux/targeted/contexts/files/file\_contexts.local

A screen shot of a computer

Description automatically generated

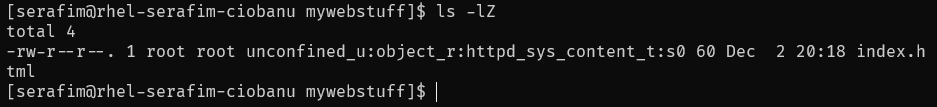
* Note that the labeling rules are written in different files. We have ‘file\_contexts’, ‘file\_contexts.homedirs’ and ‘file\_contexts.local’. The semanage tool can parse all these together without us having to remember the specific paths:  
  sudo semanage fcontext -l
* Check with ls -Z the security context of /mywebstuff/index.html . Is it changed yet? Use restorecon to (recursively) apply the new rule to existing files within ‘mywebstuff’.

A screen shot of a computer program

Description automatically generated

sudo restorecon -R mywebstuff/





* Now surf again to <http://192.168.11.20/personal> and verify it works.

A screenshot of a computer

Description automatically generated

# Creating/manipulating web content at user home folders

Now, we want our web server to serve some web pages, located at the user home folders (this is also what ISPs traditionally did to provide you with some limited personal web space).

We’ll explore the available SELinux ‘booleans’ here to tweak a loaded policy to allow Apache to access home directories, rather than manually manipulating the security ruleset.

* Let’s set SELinux to permissive mode for a moment, to ensure you’ll get things working without SELinux first.

sudo setenforce 0

A black screen with white text

Description automatically generated

* Read what is written in the following file:  
  /etc/httpd/conf.d/userdir.conf  
  and make the appropriate changes in this file to enable UserDir to work as expected (i.e. requests to the “~<user>” URL will be directed by Apache to the /home/<user>/public\_html )

A screenshot of a computer

Description automatically generated

* Restart your Apache web server
* Change the permissions of your home folder:   
  chmod 711 /home/<user>
* Create a public\_html subfolder in your home folder and give it world readable permissions:  
  chmod 755 /home/<user>/public\_html
* Create an index.html file in that folder with some random text
* Verify that surfing to [http://192.168.11.20/~<user>/](http://192.168.11.20/~%3cuser%3e/) works.

A screenshot of a computer

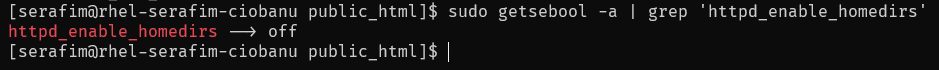
Description automatically generated

* Now, let’s turn on SELinux again and refresh the web page. You’ll get a 403 Forbidden again.

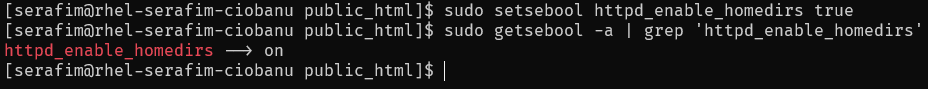
A screenshot of a computer

Description automatically generated

* With getsebool -a you’ll get a list of available Booleans for SELinux and their current state. Look at the different Booleans related to httpd. The one we’re interested in here, is ‘httpd\_enable\_homedirs’



* Now, use setsebool with appropriate arguments to turn on that Boolean. Verify that surfing to [http://192.168.11.20/~<user>/](http://192.168.11.20/~%3cuser%3e/) works again on a SELinux enforcing system now. Congrats 😉



sudo setsebool httpd\_enable\_homedirs true

sudo getsebool -a | grep 'httpd\_enable\_homedirs'

A screenshot of a computer

Description automatically generated

* The Boolean is enabled for the current running system only. It will not be applied after reboot. Let’s make a persistent, but first have a look at the timestamp of /etc/selinux/targeted/policy/policy.<XX>

A computer screen with white text

Description automatically generated

* Now, make the Boolean change persistent by using the -P flag when using your setsebool command. Look again at /etc/selinux/targeted/policy/policy.<XX> . You’ll notice that it has been recompiled to include the changed Boolean default value. A computer screen shot of a computer program

  Description automatically generated

## SELinux and PAM

In a previous lab, we’ve dealt with PAM. Using PAM, you’ve seen that you can (amongst others) make authentication decisions based on keys, fingerprints, time of day, etc. You can also make decisions based on SELinux information. Let’s explore that.

* There are already PAM modules available for SELinux functionality on a default RHEL system. Which are these PAM modules (simply based on their names)?

Hint: if you forgot the directory which contains these modules (dynamic libraries), search for ‘pam\_unix.so’, the module for traditional password authentication.

A screen shot of a computer

Description automatically generated

* Look at the man pages of these modules. Recall that there are 4 different function types for PAM modules. Which are the function types for these specific PAM modules?

pam\_linux.so

A black background with white text

Description automatically generated

pam\_sepermit.so

A black background with white text

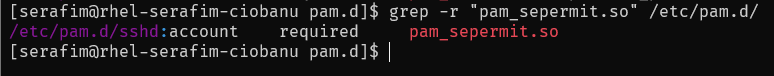
Description automatically generated

* Which PAM configuration files are using these modules? Hint: You can use grep with the recursive option to find patterns within files, instead of having to open every single file

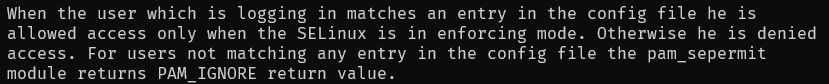
Directory **- /etc/pam.d**

A black screen with white text

Description automatically generated



* Let’s focus on the pam\_sepermit.so PAM module, which has its entry in the sshd PAM config file. Now look again at its man page. When will this module allow access and when will it deny access? What is the configuration file of this PAM module? (The location of this file makes sense, if you recall the lesson about PAM.)



User logging in matches an entry in the config file = allowed access. If only enforcing mode. Otherwise = denied access.

A black background with white text

Description automatically generated

* Now add your regular user account in this config file. Try ssh’ing to your machine once in the selinux enforcing state and once in the selinux permissive state. Do you notice pam\_sepermit’s impact?

A black screen with white text

Description automatically generated

In permissive state it does not allow me to log in. Maybe it should be different, but I think there really IS supposed to be a difference.