Chapter 1

Managing SELinux

1.1 Understanding the Need for SELinux

SELinux stands for **Security Enhanced Linux**. Let us consider an application that runs on a server, that provides a backdoor to an attacker who can start a shell session on the server. This can be done as the httpd user in the case of a vulnerability on the web server. Let us consider the attacker uses the /tmp directory (which has rwxrwxrwx permissions) for nefarious purposes. Now, we can't take away permissions, since some applications depend on the directory to have universal permissions. We also can't use a firewall, since it'd block access to HTTP services. Finally, we can't mount the file system with a `NOEXEC` flag (which prevents the execution of scripts on that disk) since sometimes applications use the `/tmp` directory to execute scripts.

Under such circumstances, SELinux becomes extremely necessary, since it permits us to set policies that define exactly what kind of access each application has, and on which directories. Thus, it is critical to use SELinux on any server that is connected to the internet.

1.1.1 SELinux and Syscalls

Every operation on the server is occurring via syscalls. When enabled, all of the syscalls are filtered through SELinux. SELinux can be in either *enforcing* or *permissive* mode for this. Each system calls go through an analysis against a policy that check whether the actions are permitted. Let's assume the action is not permitted, and a *avc:denied* is returned. Now, several things will happen.

First, the event will go through **auditd**, and in any case, whenever SELinux is enabled, *auditd* (configurable via `/etc/audit/auditd.conf`) will write the event to the audit log (`/var/log/audit/audit.log`). This is a very important source of information.

From there, if SELinux is set to *enforcing mode*, the syscall will be immediately stopped. However, in *permissive mode*, it'll go on, since in permissive mode, everything is logged by auditd, but nothing is stopped. Thus, the permissive mode allows us to analyse what is going on without stopping syscalls, stopping which might lead to system crashes and other unforeseen events.

Let us consider another example, where we have a webserver running on localhost, which we try to access using `elinks`. Now, let the webserver's DocumentRoot be set to */blah* directory. `ls -Z` prints the security context of every file or directory. On executing this command on */blah*, we will probably find that the directory has the wrong label.

Now when elinks tries to access the index file on the */blah* directory, it'll generate a *getattr* system call. If SELinux is in enforcing mode, that'll be stopped immediately.

1.2 Understanding SELinux Modes and Policy

To configure SELinux at a basic level, there are three things that we need to understand. The first of them is the SELinux Mode.

1.2.1 SELinux Mode

The SELinux mode is obtained from a file called `/etc/sysconfig/selinux`. There are three possible modes for this: Enforcing, Permissive and disabled. The disabled mode can only be specified while booting. This completely disables SELinux by ensuring all the SELinux libraries that are normally loaded by the kernel won't be loaded at all. In fact, the difference is so drastic, it's not possible to switch between disabled and any other mode without rebooting.

However, it is perfectly fine to toggle between enforcing and permissive modes. The current SELinux mode is given by:

```
# getenforce
Enforcing
```

To change the SELinux mode, we use the command:

```
# setenforce Permissive
# getenforce
Permissive
```

Toggling between the permissive and enforcing modes can be extremely useful for troubleshooting. Let us consider a scenario where we're setting up an FTP server, and it doesn't work. This may be due to an error in the FTP server config, or it's being blocked by SELinux. To make sure SELinux is not at fault, we switch to Permissive mode using `setenforce Permissive` and try again. If it starts working, it was being blocked by SELinux. Then we know where to look for the solution. However, under all other circumstances, the SELinux mode should be set to enforcing.

1.2.2 Context and Policies

Everything on RHEL 7 has a context, which can be displayed by the command:

```
# ls -Z
drwxr-xr-x. somu somu unconfined_u:object_r:user_home_t:s0 Desktop
drwxr-xr-x. somu somu unconfined_u:object_r:user_home_t:s0 Documents
drwxr-xr-x. somu somu unconfined_u:object_r:user_home_t:s0 Downloads
drwxr-xr-x. somu somu unconfined_u:object_r:audio_home_t:s0 Music
drwxr-xr-x. somu somu unconfined_u:object_r:user_home_t:s0 Pictures
drwxr-xr-x. somu somu unconfined_u:object_r:user_home_t:s0 Public
drwxr-xr-x. somu somu unconfined_u:object_r:user_home_t:s0 Templates
drwxr-xr-x. somu somu unconfined_u:object_r:user_home_t:s0 Videos
```

There are three parts to a context, with the delimiter : separating them. The first is the *user* part, which is only for advanced SELinux configurations. Next comes the *role* part, which again, is for advanced SELinux configurations. Finally, we have the *type* part. This denotes the kind of access that is allowed to files/directories.

Not only are there contexts on files, there are contexts on processes as well, which can be viewed using `ps Zau`. Even ports have context labels, viewed by using `netstat Ztulpen`. So, the idea is that every file/process/port's context is matched to a policy to grant/deny access.

1.2.3 Booleans

Booleans are easy switches to enable or disable functionalities in a policy. A list of all available booleans can be obtained with:

```
# getsebool -a
abrt_anon_write --> off
abrt_handle_event --> off
abrt_upload_watch_anon_write --> on
antivirus_can_scan_system --> off
...
zebra_write_config --> off
zoneminder_anon_write --> off
zoneminder_run_sudo --> off
```

We can filter the list and find only booleans that have 'ftp' in their boolean name using:

```
# getsebool -a | grep ftp
ftpd_anon_write --> off
ftpd_connect_all_unreserved --> off
ftpd_connect_db --> off
ftpd_full_access --> off
ftpd_use_cifs --> off
ftpd_use_fusefs --> off
ftpd_use_nfs --> off
ftpd_use_passive_mode --> off
httpd_can_connect_ftp --> off
httpd_enable_ftp_server --> off
tftp_anon_write --> off
tftp_home_dir --> off
```

For example, let us consider the boolean `ftpd_full_access --> off` which doesn't allows ftp servers to login to local user accounts and have read/write access to all files subject to Discretionary Access Control (DAC) mechanisms (permissions, ACLs, etc.). Another such boolean is `ftp_home_dir --> off` which doesn't allow users to login to their home directories.

When certain functionalities are turned off, we should always check if some boolean is turned off. In this case, since `ftp_home_dir` is off, the users won't be able to login to their home directories even though *vsftpd* may be configured to do so, since SELinux will prevent it.

1.3 Understanding SELinux Lables and Booleans

To manage SELinux, we need to be able to manage context. The context of the *httpd* process can be viewed with:

```
# ps Zaux | grep httpd
system_u:system_r:httpd_t:s0    root       1249  0.1  0.1 226240  5156 ?        Ss
 ↪  10:32   0:00 /usr/sbin/httpd -DFOREGROUND
system_u:system_r:httpd_t:s0    apache     1435  0.0  0.0 228324  3160 ?        S
 ↪  10:32   0:00 /usr/sbin/httpd -DFOREGROUND
system_u:system_r:httpd_t:s0    apache     1436  0.0  0.0 228324  3160 ?        S
 ↪  10:32   0:00 /usr/sbin/httpd -DFOREGROUND
system_u:system_r:httpd_t:s0    apache     1438  0.0  0.0 228324  3160 ?        S
 ↪  10:32   0:00 /usr/sbin/httpd -DFOREGROUND
system_u:system_r:httpd_t:s0    apache     1441  0.0  0.0 228324  3160 ?        S
 ↪  10:32   0:00 /usr/sbin/httpd -DFOREGROUND
system_u:system_r:httpd_t:s0    apache     1443  0.0  0.0 228324  3160 ?        S
 ↪  10:32   0:00 /usr/sbin/httpd -DFOREGROUND
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 root 2738 0.0  0.0 112664 968 pts/0
 ↪  S+ 10:34   0:00 grep --color=auto httpd
```

Here we can see that the current context of the httpd process is *httpd_t*. Now, the default document root for the httpd process is `/var/www` and we can see its context using:

```
# ls -Z /var/www
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
```

We can see that the context for `/var/www` has been set correctly. The policy will state that the source context *httpd_t* is allowed to get through the target context *httpd_sys_content_t*.

Now if an attacker finds a vulnerability on a web server script, and tries to access the `/tmp` directory, SELinux would prevent that because the context of `/tmp` has been set to:

```
# ls -dZ /tmp
drwxrwxrwt. root root system_u:object_r:tmp_t:s0       /tmp
```

Thus, the process with the source context *httpd_t* won't be allowed to access a directory with a target context of *tmp_t*.

There are primarily two situations where administrators may need to manage context:

- A file has been moved instead of copied, or
- We want to do something that doesn't correspond to the defaults.

1.3.1 File being moved instead of copied

Let us consider a file *myFile* in our home directory. In that case, it'd have the context of:

```
# touch myFile
# ls -Z
-rw-------. root root system_u:object_r:admin_home_t:s0 anaconda-ks.cfg
-rw-r--r--. root root system_u:object_r:admin_home_t:s0 initial-setup-ks.cfg
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 myFile
```

```
# ls -dZ
dr-xr-x---. root root system_u:object_r:admin_home_t:s0 .
```

We can see that the file we created has a context of `admin_home_t`. This is because the current directory `/root` also has a context of `admin_home_t`.

Now, let us make a copy of the `/etc/hosts` file and name it `/etc/hosts2`. If we move that file, instead of copy it to the home directory of the home user, it'll have a context of:

```
# ls -Z hosts2
-rw-r--r--. root root unconfined_u:object_r:etc_t:s0   hosts2
```

The context of *hosts2* is set to *etc_t* because while moving a file, the original context moves with it. When copying a file, however, a new file is created and it normally inherits the context of the parent (target) directory.

1.3.2 semanage

The **semanage** utility is used to set context. It works with a set of arguments, and a specific argument defines what it's actions will be. Some of the important arguments are:

Options	Description
fcontext	Manages the fcontext of the object.
boolean	Used to change the value of a boolean
port	Changes the port type definition.

The documentation for `semanage` has been arranged in such a way that a separate man page exists for each of the arguments. Thus, there's a man page for `man semanage-fcontext`, `man semanage-boolean`, etc. The examples in the man page for *semanage-fcontext* has examples for setting the context for everything under the `/web` directory:

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

The `-t` flag sets the type of `httpd_sys_content_t` for all items that match the regular expression `/web(/.*)?`. This matches everything in the web directory, and any files/sub-directories contained within it.

Note that `semanage fcontext` doesn't write to the file system directly, but to the policy. This is because all the default policies should be set in the policy and not the file system. To apply these changes from the policy to the file system, we need to use the command:

```
# restorecon -R -v /web
```

The `-R` flag makes it recursive and the `-v` flag makes it verbose. The `restorecon` utility is also very useful when something goes wrong with a context, because it checks the policy and ensures that the context of every file in a directory matches their context as described in the policy.

The file we moved from the `/etc/hosts` directory has the wrong context of *etc_t*, instead of *admin_home_t*. This can be fixed using:

```
# restorecon -v hosts2
restorecon reset /root/hosts2 context
↪  unconfined_u:object_r:etc_t:s0->unconfined_u:object_r:admin_home_t:s0
```

```
# ls -Z hosts2
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 hosts2
```

We can see that the file *hosts2* now has the correct context for the directory `/root`. This could also have been done directly on the `/root` directory to fix all the wrong contexts in the directory at once, using `restorecon -R -v /root`.

1.4 Understanding File System Labels

If we want to change the context using `semanage fcontext`, we should know which context to use. There are many contexts to choose from. One possible solution is to go to the target directory and view which context the files use. For example, the contents of `/var/www` directory use:

```
# ls -Z
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
```

The available contexts are: *httpd_sys_script_exec_t* and *httpd_sys_content_t*. A list of all possible contexts can be displayed using `semanage fcontext -l`. However, it's a long list and grepping does help, but the filtered contents are still long:

```
# semanage fcontext -l | grep http
/usr/.*\.cgi                                        regular file
    ↪  system_u:object_r:httpd_sys_script_exec_t:s0
/opt/.*\.cgi                                        regular file
    ↪  system_u:object_r:httpd_sys_script_exec_t:s0
/srv/([^/]*/)?www(/.*)?                             all files
    ↪  system_u:object_r:httpd_sys_content_t:s0
/srv/([^/]*/)?www/logs(/.*)?                        all files
    ↪  system_u:object_r:httpd_log_t:s0
/var/www(/.*)?                                      all files
    ↪  system_u:object_r:httpd_sys_content_t:s0
/var/www(/.*)?/logs(/.*)?                           all files
    ↪  system_u:object_r:httpd_log_t:s0
...
/usr/share/wordpress-mu/wp-config\.php              regular file
    ↪  system_u:object_r:httpd_sys_script_exec_t:s0
/usr/share/munin/plugins/http_loadtime              regular file
    ↪  system_u:object_r:services_munin_plugin_exec_t:s0
/usr/share/system-config-httpd/system-config-httpd regular file
    ↪  system_u:object_r:bin_t:s0
```

What do help are the SELinux man pages. On previous versions of RHEL, the man pages were available through the command `man -k _selinux`. However, on RHEL 7 these need to be generated by us using the `sepolicy` utility, which isn't installed by default. We can find which package provides it using:

```
# yum provides */sepolicy
Loaded plugins: fastestmirror, langpacks
Loading mirror speeds from cached hostfile
* base: centos.mirror.net.in
* extras: centos.mirror.net.in
* updates: centos.mirror.net.in
policycoreutils-devel-2.5-17.1.el7.i686 : SELinux policy core policy devel utilities
```

```
Repo        : base
Matched from:
Filename    : /usr/share/bash-completion/completions/sepolicy
Filename    : /usr/bin/sepolicy

policycoreutils-devel-2.5-17.1.el7.x86_64 : SELinux policy core policy devel utilities
Repo        : base
Matched from:
Filename    : /usr/share/bash-completion/completions/sepolicy
Filename    : /usr/bin/sepolicy

policycoreutils-python-2.5-17.1.el7.x86_64 : SELinux policy core python utilities
Repo        : base
Matched from:
Filename    : /usr/lib64/python2.7/site-packages/sepolicy
```

So, we need the policycoreutils development version. So, we install it using the command `yum install -y policycoreutils-devel`.

Once installed, we need to run a command that helps us find the correct man page for a SELinux command, which is:

```
# sepolicy manpage -a -p /usr/share/man/man8
/usr/share/man/man8/NetworkManager_selinux.8
/usr/share/man/man8/abrt_selinux.8
/usr/share/man/man8/abrt_dump_oops_selinux.8
...
/usr/share/man/man8/zoneminder_script_selinux.8
/usr/share/man/man8/zos_remote_selinux.8
# mandb
```

The command generates a list of manpages. Every service available on SELinux has its own manpage, created by running this command. Once the manpages have been generated, we should run the `mandb`, which updates the index of the manpages, making them searchable using `man -k`.

So, to search for the SELinux manpages for anything concerning httpd, we use:

```
# man -k _selinux | grep http
apache_selinux (8)   - Security Enhanced Linux Policy for the httpd processes
httpd_helper_selinux (8) - Security Enhanced Linux Policy for the httpd_helper processes
httpd_passwd_selinux (8) - Security Enhanced Linux Policy for the httpd_passwd processes
httpd_php_selinux (8) - Security Enhanced Linux Policy for the httpd_php processes
httpd_rotatelogs_selinux (8) - Security Enhanced Linux Policy for the httpd_rotatelogs
 ↪  processes
httpd_selinux (8)    - Security Enhanced Linux Policy for the httpd processes
httpd_suexec_selinux (8) - Security Enhanced Linux Policy for the httpd_suexec processes
httpd_sys_script_selinux (8) - Security Enhanced Linux Policy for the httpd_sys_script
 ↪  processes
httpd_unconfined_script_selinux (8) - Security Enhanced Linux Policy for the
 ↪  httpd_unconfined_script processes
httpd_user_script_selinux (8) - Security Enhanced Linux Policy for the httpd_user_script
 ↪  processes
```

Inside these manpages, all booleans and contexts are defined. So, we have a place to look up the appropriate context for the kind of access that our files/processes need.

1.5 Understanding semanage fcontext and chcon differences

In certain man page entries, we might come across the command `chcon`, which is a *bad* program, and shouldn't be used. For this, we need to understand the difference between `semanage`, `fcontext` and `chcon`.

Let us consider a scenario where we need to change the context of a file `/blah/index.html`. Suppose we want to set its context to *httpd_sys_content_t*. To do this using `chcon`, we would need to use the command:

```
# chcon -R --type=httpd_sys_content_t /blah
```

What this command does is set the given context type to the inode, i.e., applies the change to the file system. The corresponding entry for it in the policy still remains *default_t*. This is bad because whenever a relabel operation occurs (typically on the entire root file system [relabel of /]), the context for the `/blah` directory would be overwritten to *default_t*, because during a relabel everything in the policy overwrites everything in the file system. Thus, it is absolutely critical that SELinux information is always written to the policy first! This is why to set the context of a file/directory, we use:

```
# semanage fcontext -a -t httpd_sys_content_t "/blah(/.*)?"
```

This sets the context in the policy and thus the change will survive the relabel activity.

1.6 Using Booleans

To handle booleans, we need two commands: `getsebool` and `setsebool`. The command to list all possible SELinux Boolean Switches on a particular system is given by:

```
# getsebool -a
abrt_anon_write --> off
abrt_handle_event --> off
abrt_upload_watch_anon_write --> on
...
zoneminder_anon_write --> off
zoneminder_run_sudo --> off
```

To find an appropriate boolean for something (e.g., FTP), we use grepping:

```
# getsebool -a | grep ftp
ftpd_anon_write --> off
ftpd_connect_all_unreserved --> off
ftpd_connect_db --> off
ftpd_full_access --> off
ftpd_use_cifs --> off
ftpd_use_fusefs --> off
ftpd_use_nfs --> off
ftpd_use_passive_mode --> off
httpd_can_connect_ftp --> off
httpd_enable_ftp_server --> off
tftp_anon_write --> off
tftp_home_dir --> off
```

For example, if we want to turn on the switch for `ftpd_use_nfs --> off`, all we need to do is:

```
# setsebool ftpd_use_nfs on
# getsebool ftpd_use_nfs
ftpd_use_nfs --> on
```

These changes are however temporary in nature, and thus lost after a restart. To make these changes permanent, we need to use:

```
# setsebool -P ftpd_use_nfs on
# getsebool ftpd_use_nfs
ftpd_use_nfs --> on
```

This particular operation takes considerably more time since the policy has to be modified.

1.7 Analyzing SELinux Log Files

Understanding what is going wrong in a SELinux enabled environment isn't always easy, even though SELinux logs each occurrence of requests coming to it. To help us there are the **setroubleshoot** packages. Whether they're installed or not can be checked with:

```
# yum list installed | grep setrouble
setroubleshoot.x86_64                  3.2.28-3.el7               @anaconda
setroubleshoot-plugins.noarch          3.0.65-1.el7               @anaconda
setroubleshoot-server.x86_64           3.2.28-3.el7               @anaconda
```

All the events that have been logged by SELinux go to the *audit log*. In order for the audit log to be working, the *auditd* process needs to be started. We can confirm that it's working using `systemctl status auditd`, and if it is, we can view the log using:

```
# grep AVC /var/log/audit/audit.log
type=AVC msg=audit(1513680230.189:22): avc:  denied  { write } for  pid=709
 ↪  comm="accounts-daemon" name="root" dev="dm-0" ino=33574977
 ↪  scontext=system_u:system_r:accountsd_t:s0 tcontext=system_u:object_r:admin_home_t:s0
 ↪  tclass=dir
type=USER_AVC msg=audit(1513760499.935:10): pid=1 uid=0 auid=4294967295 ses=4294967295
 ↪  subj=system_u:system_r:init_t:s0 msg='avc:  received setenforce notice (enforcing=0)
 ↪  exe="/usr/lib/systemd/systemd" sauid=0 hostname=? addr=? terminal=?'
...
type=USER_AVC msg=audit(1513848001.387:320): pid=1 uid=0 auid=4294967295 ses=4294967295
 ↪  subj=system_u:system_r:init_t:s0 msg='avc:  received policyload notice (seqno=2)
 ↪  exe="/usr/lib/systemd/systemd" sauid=0 hostname=? addr=? terminal=?'
type=USER_AVC msg=audit(1513848601.536:329): pid=1 uid=0 auid=4294967295 ses=4294967295
 ↪  subj=system_u:system_r:init_t:s0 msg='avc:  received policyload notice (seqno=3)
 ↪  exe="/usr/lib/systemd/systemd" sauid=0 hostname=? addr=? terminal=?'
```

All SELinux messages start with the header **AVC**. Once such case where some action was denied by SELinux is:

```
# grep 'type=AVC' /var/log/audit/audit.log
type=AVC msg=audit(1513680230.189:22): avc:  denied  { write } for  pid=709
 ↪  comm="accounts-daemon" name="root" dev="dm-0" ino=33574977
 ↪  scontext=system_u:system_r:accountsd_t:s0 tcontext=system_u:object_r:admin_home_t:s0
 ↪  tclass=dir
```

The above incident tells us a file write system call was denied by SELinux on the directory `/root` as the context noted in the policy (*accountsd_t*) didn't match the context for the directory being accessed (*admin_home_t*). In the `/var/log/messages` file, more detail can be found on the event. If we check the `/var/log/messages` file, we can see the corresponding entry in it by searching for the term **sealert**:

```
# less /var/log/messages
Dec 19 16:13:56 vmPrime setroubleshoot: SELinux is preventing
↪  /usr/libexec/accounts-daemon from write access on the directory root. For complete
↪  SELinux messages run: sealert -l e277d205-f3b0-4ef7-a6c2-178a813da2e0
```

Finally, the noted command, `sealert -l e277d205-f3b0-4ef7-a6c2-178a813da2e0` explains the event in very great detail. **sealert** consults a database on the system to analyse what went wrong.

```
SELinux is preventing /usr/libexec/accounts-daemon from write access on the directory
↪  root.
*****  Plugin catchall (100. confidence) suggests   *************************
...
Additional Information:
Source Context                system_u:system_r:accountsd_t:s0
Target Context                system_u:object_r:admin_home_t:s0
Target Objects                root [ dir ]
Source                        accounts-daemon
Source Path                   /usr/libexec/accounts-daemon
Port                          <Unknown>
Host                          vmPrime.somuVMnet.com
Source RPM Packages           accountsservice-0.6.45-2.el7.x86_64
Target RPM Packages           filesystem-3.2-21.el7.x86_64
Policy RPM                    selinux-policy-3.13.1-166.el7.noarch
Selinux Enabled               True
Policy Type                   targeted
Enforcing Mode                Enforcing
Host Name                     vmPrime.somuVMnet.com
Platform                      Linux vmPrime.somuVMnet.com 3.10.0-693.el7.x86_64
.#1 SMP Tue Aug 22 21:09:27 UTC 2017 x86_64 x86_64
Alert Count                   1
First Seen                    2017-12-19 16:13:50 IST
Last Seen                     2017-12-19 16:13:50 IST
Local ID                      e277d205-f3b0-4ef7-a6c2-178a813da2e0
...
```

The confidence score suggests how likely a suggestion is to work. Note that these are automated attempts to solve whatever is wrong, and might not always be correct, and the SysAdmin must consider if it's a valid option and if the solution meets his/her requirements.

1.8 Configuring SELinux for Apache

This is how we deal with SELinux during real-life scenarios such as while configuring the apache web server. Let us consider we want to use a new document root at `/web`. We put an index.html file in the directory, and configure the `/etc/http/conf/httpd.conf` file with the new document root at `/web`, by adding the lines below:

```
DocumentRoot "/web"

```

```
<Directory "/web">
AllowOverride None
# Allow open access:
Require all granted
</Directory>

#<Directory "/var/www/html"> --> Manually commenting out the old tag start, and copying the original config with our own.
<Directory "/web">
...
```

The lines 3-7 help provide access to the new document root. Note that this is modelled on the original `<Directory>` tag for the document root `/var/www`, which itself must not be edited or commented out (to stop other functionality from being disabled).

Once the *httpd.conf* file has been edited, we need to restart the apache service, with `systemctl restart httpd`. Now, the index.html should be available on the address `http://localhost`. Instead of the index.html page, we see an error page from the apache web server that reads *"The website you just visited is either experiencing problems or undergoing routine maintenance"*.

This generates the following SELinux notifications in the logs:

```
# In /var/log/audit/audit.log: [grep AVC /var/log/audit/audit.log]
type=AVC msg=audit(1513869456.604:322): avc:  denied  { getattr } for  pid=3683 comm="httpd" path="/web/index.html" dev="dm-0" ino=2556901 scontext=system_u:system_r:httpd_t:s0 tcontext=unconfined_u:object_r:default_t:s0 tclass=file

# In /var/log/messages: [less /var/log/messages ?sealert]
Dec 21 20:49:37 vmPrime setroubleshoot: SELinux is preventing httpd from getattr access on the file /web/index.html. For complete SELinux messages run: sealert -l 001da822-04b5-498d-8344-d78de9014597
```

It is clear from the message in the audit log that this is a case of context type mismatch. The source has a context label of *httpd_t* while the target directory and file (`/web/index.html`) have the context label of *default_t*. We can fix this using the command:

```
# semanage fcontext -a -t httpd_sys_content_t '/web(/.*)?'
# restorecon -R -v /web
restorecon reset /web context unconfined_u:object_r:default_t:s0->unconfined_u:object_r:httpd_sys_content_t:s0
restorecon reset /web/index.html context unconfined_u:object_r:default_t:s0->unconfined_u:object_r:httpd_sys_content_t:s0
```

Now when we visit the webpage at `http://localhost/index.html`, SELinux won't block us anymore.