Troubleshooting SELinux

Hal Pomeranz, Deer Run Associates

For most people SELinux is nothing more than "that annoying security feature I need to remember to turn off during the install." This is not entirely surprising, since the SELinux documentation has always been a little sketchy and is frequently out of date. And in many environments, enabling SELinux leads to strange failures with mysterious error messages. This article will shed some light on the basic principles behind SELinux and help you deal with some of the typical problems you will encounter when enabling SELinux on your systems.

Why should you even care about SELinux? When implemented properly, SELinux is an effective *application whitelisting* tool that restricts critical applications to only the specific set of functionality they need to accomplish their mission. If an attacker were to subvert the application via a buffer overflow or other exploit, SELinux would very likely prevent the attacker from using the compromised application from accessing critical files and directories in the operating system. In other words, SELinux can prevent exploits that could compromise your system and steal your data and other computing resources. This is powerful.

Is It Turned On? What's It Doing?

The first question for most sites is, "Is SELinux turned on?" The current state of SELinux on your system is visible via the `sestatus` command:

```
# sestatus
SELinux status:                 enabled
SELinuxfs mount:                /selinux
Current mode:                   permissive
Mode from config file:          permissive
Policy version:                 21
Policy from config file:        targeted
```

The first item to point out in the `sestatus` output is that this system is using the `targeted` policy. The standard targeted policy is designed to only affect specific daemons running on the system--typically those operating services that are listening on network sockets and therefore potential entry points for an external attacker. Normal interactive user sessions and additional third-party software packages you've added to the system will generally not be constrained by SELinux. The `targeted` policy was a compromise measure designed to make it easier to adopt SELinux.

We can also see from the above output that SELinux is `enabled`, however it is currently in `permissive` mode. Being `enabled` means that SELinux is currently active and monitoring/logging security events on the system, but `permissive` mode means that violations of the current security policy are being allowed to happen. This mode is often used for testing new services: you can allow the service to run and collect all of the violations reported by SELinux and then later use that audit trail to create a working SELinux policy for the new service. But many sites run their systems in this mode by default, since it's one of the ways to prevent SELinux from interfering with the functioning of the various daemons on the system.

Things get more interesting when you switch from `permissive` to `enforcing` mode. Once that happens, SELinux actively begins preventing services from doing things they shouldn't, based on the policy configuration provided with the operating system. You can switch to `enforcing` mode with

the `setenforce` command:

```
# setenforce 1
# getenforce
Enforcing
```

Notice that there's also a `getenforce` command that will let you know what state the system is in without having to look at all of the other output from `sestatus`.

However, if you look at the output from our first `sestatus` example you'll see that the "`Mode from config file`" is `permissive`. That means, the next time we reboot the system we will be back in `permissive` mode. To change the default mode to `enforcing`, you will need to modify the appropriate SELinux configuration file on your system (it's `/etc/sysconfig/selinux` on Red Hat systems).

Context Is Everything

Before we go any farther, it's important to understand the concept of SELinux *contexts*. A *context* in the SELinux universe is nothing more than a special label given to the various objects in the operating system. Everything in the OS has an SELinux context: files and directories, sockets, processes, and even users. SELinux policy statements describe what an object in one context-- typically a process like a web server daemon-- can do to other objects on the system-- usually files, directories, and network sockets.

In order to view the SELinux contexts associated with different objects, you use standard OS commands like `ls` and `ps`, but with the "`-Z`" option that reveals the SELinux information:

```
# id -Z
user_u:system_r:unconfined_t
# ls -dZ /var/www/html
drwxr-xr-x  root root system_u:object_r:httpd_sys_content_t
/var/www/html
# ps -efZ | grep httpd
user_u:system_r:httpd_t   root    3728     1  2 10:49 ?
00:00:00 /usr/sbin/httpd
user_u:system_r:httpd_t   apache  3730  3728  0 10:49 ?
00:00:00 /usr/sbin/httpd
...
```

The weird looking strings like "`user_u:system_r:unconfined_t `" are the SELinux contexts. There's nothing particularly magical about these labels: they're simply strings that were chosen by humans to differentiate objects in the operating system. Various conventions have developed for the context names, but understanding all of the different conventions is not critical for getting started with SELinux.

Note that contexts are generally inherited. If we were to create a file under `/var/www/html`, then by default that file would inherit the `system_u:object_r:httpd_sys_content_t` context of the parent directory. Similarly, if the web server were to spawn another process, that new process would have the `user_u:system_r:httpd_t` context of the parent process. There are some special exceptions-- *transitions* in the SELinux lingo-- that allow an object to shift contexts. For example, if you were to restart the web server process from your user shell in the `user_u:system_r:unconfined_t` context, we would want the web server to end up in the

proper `user_u:system_r:httpd_t` context. These transitions are taken care of for you by the standard SELinux policy provided with the OS.

Causing Trouble

The standard SELinux policy and context configuration that comes with your Linux distro should have been thoroughly tested by your vendor, and in general things should work fine as long as you don't depart from the vendor's standard way of doing things. But nobody ever does that. You always have site-specific customizations, and this is where people start running into trouble with SELinux. And since the documentation on SELinux is unhelpful, the easiest thing for administrators to do is disable SELinux at the first sign of trouble. However, there are ways that you can fix your SELinux issues without disabling SELinux or putting it into permissive mode. We're going to work through a couple of examples that will show you some common troubleshooting strategies and introduce you to several useful SELinux-related commands at the same time.

Our first example is quite common: using a different directory from the OS "standard". For example, instead of using the Red Hat default `/var/www/html` directory for our web server document root, suppose we've decided to use `/docroot` instead. We create our `/docroot` directory and put some content underneath it, and then update our `httpd.conf` file. However, when we go to restart our web server we get a very strange error message:

```
# /etc/init.d/httpd start
Starting httpd: Syntax error on line 281 of httpd.conf:
DocumentRoot must be a directory
                                                   [FAILED]
# ls -ld /docroot
drwxr-xr-x 2 root root 1024 May 16 11:34 /docroot
```

The error message seems to be saying that our new `DocumentRoot` isn't a directory or doesn't exist. But you can see clearly from the `ls` output that `/docroot` does exist and is a directory. At this point you might suspect that perhaps SELinux has something to do with the problem, but how can you be sure?

Information about SELinux violations end up in the `/var/log/audit/audit.log` file. However, other types of log messages end up here as well. Here's a useful idiom for pulling out just the SELinux-related log messages we're interested in:

```
# egrep '^type=(AVC|SELINUX)' /var/log/audit/audit.log | tail -1
type=AVC msg=audit(1274034906.250:20): avc:  denied  { getattr }
for  pid=2603 comm="httpd" path="/docroot" dev=sda6 ino=24481
scontext=user_u:system_r:httpd_t:s0
tcontext=user_u:object_r:default_t:s0 tclass=dir
```

The messages we want to look at always begin with either "`type=AVC`" or "`type=SELINUX`", so we're using `egrep` to pull just those matching lines out of the log file. I'm also using `tail` here to look at only the last message, hoping that it will relate to the problem we just had starting the web server.

While the log message is a little difficult to understand at first, it does appear to relate to our problem: it's a "`denied`" message relating to "`httpd`" and "`/docroot`". "`scontext`" here means *source context*, and if you'll refer back to our earlier example with the output of "`ps -efZ`", you'll see that `user_u:system_r:httpd_t` is the standard context for the `httpd` process. Similarly,

"tcontext" means *target context* and we can use "ls -Z" to confirm that user_u:object_r:default_t is indeed the context on our /docroot directory:

```
# ls -dZ /docroot
drwxr-xr-x  root root user_u:object_r:default_t        /docroot
```

So it would seem that this message is indeed about an httpd process trying to access the /docroot directory. SELinux is completely denying the web server any access to /docroot, which the httpd process is interpreting as the directory not existing. Hence the strange error message we saw when trying to start the web server above.

The one thing that seems to be missing from our log message is some kind of time stamp so that we can figure out if this log message is related to our most recent attempt to start the web server, or is just a relic of some previous action. It turns out that there really is a time stamp in the message-- it's just in a format that you don't recognize. The value "1274034906" near the front of the log message is actually a time stamp in Unix "*epoch time*" (seconds since Jan 1, 1970). While it's unfortunate that this date format was chosen for the log messages, the good news is that GNU date command provides a mechanism for converting these dates to human-readable strings:

```
# date -d @1274034906
Sun May 16 11:35:06 PDT 2010
```

You should be able to confirm that the date you get correlates with your attempt to start the web server.

So now we're able to track down the logs related to our SELinux failures, and even understand them a little bit. And our problem starting up the web server does seem to be related to SELinux. But how can we fix the problem and get our web server to start up with the DocumentRoot set to /docroot?

Changing File Contexts

As noted earlier, the Red Hat default DocumentRoot is /var/www/html. We can verify that the web server works when DocumentRoot is set to this directory. That would suggest that there's some difference in the SELinux context set on /var/www/html and our new /docroot directory. If we could somehow set the same SELinux context on /docroot as currently exists on /var/www/html, then perhaps SELinux would allow us to start up our web server.

First let's compare the SELinux contexts on the two directories:

```
# ls -dZ /docroot /var/www/html
drwxr-xr-x  root root user_u:object_r:default_t        /docroot
drwxr-xr-x  root root system_u:object_r:httpd_sys_content_t
/var/www/html
```

It's clear that the two directories are in different contexts. But how can we change the context on /docroot?

Just like chown and chmod for changing permissions, SELinux comes with a chcon command for changing contexts. So we could try that:

```
# chcon -R system_u:object_r:httpd_sys_content_t /docroot
# ls -dZ /docroot /var/www/html
drwxr-xr-x  root root system_u:object_r:httpd_sys_content_t
/docroot
drwxr-xr-x  root root system_u:object_r:httpd_sys_content_t
```

```
/var/www/html
# /etc/init.d/httpd start
Starting httpd:                                            [  OK  ]
```

Awesome! We were able to change the file context on `/docroot` and that in turn allowed us to start up our web server without interference from SELinux.

It turns out, however, that `chcon` is not the recommended method for making these sorts of changes because there are times when the settings you make with `chcon` can be reversed by other actions. The most common example is file and directory contexts being lost when you restore data from backup. Many backup methods will not preserve SELinux context information-- when you restore the data, your files and directories may end up with generic contexts like the `user_u:object_r:default_t` context we saw on `/docroot` originally. We can simulate this by simply removing and recreating our `/docroot` directory:

```
# rm -rf /docroot
# mkdir /docroot
# ls -dZ /docroot
drwxr-xr-x  root root user_u:object_r:default_t        /docroot
```

As you can see, the context on the directory is once again `user_u:object_r:default_t`.

The "right way" to set file contexts on new directories is with the `semanage` command. `semanage` allows us to create an entry in the SELinux policy database that creates a more "permanent" context setting for our directory:

```
# semanage fcontext -a -t httpd_sys_content_t '/docroot(/.*)?'
# semanage fcontext -l | grep /docroot
/docroot(/.*)?   all files
system_u:object_r:httpd_sys_content_t:s0
```

"`semanage fcontext`" is used for setting file contexts. The "`-a`" option allows us to *add* a file context entry. Here we're adding and entry that associates type ("`-t`") `httpd_sys_content_t` with "`/docroot(/.*)?`" The funny looking regular expression here matches not only "`/docroot`" but also all path names under `/docroot`, and is the usual way you would apply a setting recursively to an entire directory structure. You can use "`semanage fcontext -l`" to *list* all policy database entries set up this way. There are a lot of them, so you'll need to use `grep` to pick out just the entries you're interested in.

The one thing that `semanage` doesn't do is actually change the current file context settings on our `/docroot` directory. You could use `chcon` for this, but the usual approach is to use `restorecon` instead:

```
# restorecon -Rvv /docroot
restorecon reset /docroot context user_u:object_r:default_t:s0-
>system_u:object_r:httpd_sys_content_t:s0
restorecon reset /docroot/index.html context
user_u:object_r:default_t:s0-
>system_u:object_r:httpd_sys_content_t:s0
```

`restorecon` looks at the policy settings in the policy database you manipulated with `semanage` and resets all file and directory contexts to their appropriate values. The "`-R`" option means "recursive", descending through the entire directory structure-- I've added an `index.html` file under

/docroot so you could see the recursive option in action. I've also upped the verbosity of the restorecon command ("-vv") so that you can see what it's doing.

So the advantage to using semanage is that if your file or directory contexts ever get reset, you can just run restorecon to fix everything without having to remember what the "correct" contexts should be. In fact, if you "touch /.autorelabel" and then reboot your system, Red Hat systems will automatically run restorecon from the root during the next reboot to fix *all* of the file contexts in the OS. This can take a while, however, so it's not something you want to do often. But it can be useful after you restore your system from backup.

Frankly, most problems that sites run into with SELinux are the result of mismatching or incorrectly set file contexts. So you now know enough to fix the most common sort of SELinux issues. But there are other problems that you might run into.

Dealing With Port Contexts

You can sometimes run into SELinux issues if you use non-standard port numbers for various services. For example, suppose we decided to run our web server on port 8001/tcp. After making the appropriate change to httpd.conf, we restart our web server and run into another unhelpful error message:

```
# /etc/init.d/httpd start
Starting httpd: (13)Permission denied: make_sock: could not bind
to address 0.0.0.0:8001
no listening sockets available, shutting down
Unable to open logs
                                                    [FAILED]
```

Again you might suspect that the problem is related to SELinux, but let's look at the audit.log file just to be sure:

```
# egrep '^type=(AVC|SELINUX)' /var/log/audit/audit.log | tail -1
type=AVC msg=audit(1274039656.994:35): avc:  denied  { name_bind
} for  pid=3254 comm="httpd" src=8001
scontext=user_u:system_r:httpd_t:s0
tcontext=system_u:object_r:port_t:s0 tclass=tcp_socket
```

It does appear that this is a "denied" message regarding "httpd" and port "8001". So SELinux very likely is the culprit. But how can we interact with SELinux contexts related to network ports?

The answer again is semanage:

```
# semanage port -l | grep ' 80,'
http_port_t      tcp      80, 443, 488, 8008, 8009, 8443
```

Just as "semanage fcontext -l" listed file context entries, we use "semanage port -l" to list policy entries related to network ports. And since there are a lot of these entries, we're once again using grep to pull out just the information we're interested in. Notice that my grep expression here is "<*space*>80<*comma*>" so that I only pull out information related to 80/tcp and not ports like 8080, which I would have matched had I just done "... | grep 80".

In a similar fashion to our earlier file context example, we can also use "semanage port -a ..." to set the appropriate context on the port we're trying to use. The syntax for the command is a little odd, however:

```
# semanage port -a -t http_port_t -p tcp 8001
# semanage port -l | grep ' 8001,'
http_port_t        tcp      8001, 80, 443, 488, 8008, 8009, 8443
# /etc/init.d/httpd start
Starting httpd:                                            [  OK  ]
# netstat -antp | grep 8001
tcp        0      0 0.0.0.0:8001                0.0.0.0:*
LISTEN      4434/httpd
```

The "`semanage port -a -t http_port_t`" portion of the command looks remarkably similar to our earlier file context example. The weird part is that you need to specify the protocol with "`-p`" followed by the port number. We can then use "`semanage port -l`" to confirm that the new port has been added to the list of ports associated with `http_port_t`. Unlike our earlier file context example there's no need to run an extra command like `restorecon`-- 8001/tcp is immediately associated with the appropriate port context and SELinux should now allow us to start up the web server on this port. We test this by starting the server and confirming that it's listening on the new port number.

So, slightly odd `semange` syntax aside, dealing with port contexts is actually somewhat easier than dealing with file contexts. And since inappropriate file and port contexts are easily the most common source of SELinux difficulties for most sites, you now know enough to navigate the typical problems you're likely to run into when getting started with SELinux at your site.

SELinux Can Help

SELinux can be a significant addition to the security configuration of your systems and make life much more difficult for attackers. But mysterious errors and poor or non-existent documentation has convinced many sites that it isn't worth the bother. Hopefully the examples we've worked through will help you troubleshoot SELinux issues at your site and give you the opportunity to use SELinux effectively in your environment.

Hal Pomeranz is the Founder and Technical Lead of Deer Run Associates, an IT and Information Security consulting firm. He is also a Faculty Fellow of the SANS Institute and the course developer and primary instructor for their Linux/Unix Security certification track (GCUX). He is currently developing the book and lyrics for "SELinux: The Musical!"