SELinux

1. Introduction

Security-Enhanced Linux (SELinux) is a mandatory access control (MAC) security mechanism implemented in the kernel. SELinux was first introduced in CentOS 4 and significantly enhanced in CentOS 5 and 6. These enhancements mean that content varies as to how to approach SELinux over time to solve problems.

1.1. Some of the Problems

In order to better understand why SELinux is important and what it can do for you, it is easiest to look at some examples. Without SELinux enabled, only traditional discretionary access control (DAC) methods such as file permissions or access control lists (ACLs) are used to control the file access of users. Users and programs alike are allowed to grant insecure file permissions to others or, conversely, to gain access to parts of the system that should not otherwise be necessary for normal operation. For example:

- Administrators have no way to control users: A user could set world readable permissions on sensitive files such as ssh keys and the directory containing such keys, customarily: ~/.ssh/
- Processes can change security properties: A user's mail files should be readable only by that user, but the mail client software has the ability to change them to be world readable
- Processes inherit user's rights: Firefox, if compromised by a **trojaned** version, could read a user's private ssh keys even though it has no reason to do so.

Essentially under the traditional DAC model, there are two privilege levels, root and user, and no easy way to enforce a model of least-privilege. Many processes that are launched by root later drop their rights to run as a restricted user and some processes may be run in a chroot jail but all of these security methods are discretionary.

1.2. The Solution

SELinux follows the model of least-privilege more closely. By default under a strict **enforcing** setting, everything is denied and then a series of exceptions policies are written that give each element of the system (a service, program or user) only the access required to function. If a service, program or user subsequently tries to access or modify a file or resource not necessary for it to function, then access is denied and the action is can be logged.

Because SELinux is implemented within the kernel, individual applications do not need to be especially written or modified to work under SELinux although, of course, if written to watch for the error codes which SELinux returns, vide infra, might work better afterwards. If SELinux blocks an action, this is reported to the underlying application as a normal (or, at least, conventional) "access denied" type error to the application. Many applications, however, do not test all return codes on system calls and may return no message explaining the issue or may return in a misleading fashion.

Please note, however, that the hypothetical examples posed to provide possible greater safety of:

- constraining those programs authorized to a limited set of programs, permitted to read a user's ~/.ssh/ directory
- preventing a Mail Delivery Agent program from tampering with group ownership or setting on group or other file read permissions
- a web browser being constrained from reading the user's home directory

HAS NOT BEEN IMPLEMENTED in the SELinux rules accompanying any version of CentOS up to version 6. This is a developing area and, in all honesty, it is quite unlikely that it will be implemented soon. This is because the sysadmin customer community of the Upstream Vendor would not well tolerate such measures to do any of the foregoing, without much "wailing" and subsequent increased support load upstream.

2. SELinux Modes

SELinux has three basic modes of operation, of which **Enforcing** is set as the installation default mode. There is, however, an additional qualifier of **targeted** or **mls** which control how pervasive SELinux rules are applied, with **targeted** being the less stringent level.

- **Enforcing:** The default mode which will enable and enforce the SELinux security policy on the system, denying access and logging actions
- **Permissive:** In Permissive mode, SELinux is enabled but will not enforce the security policy, only warn and log actions. Permissive mode is useful for troubleshooting SELinux issues
- **Disabled:** SELinux is turned off

The SELinux mode can be viewed and changed by using the SELinux Management GUI tool available on the Administration menu or from the command line by running 'system-config-selinux' (the SELinux Management GUI tool is part of the **policycoreutils-gui** package and is not installed by default).

Users who prefer the command line may use the 'sestatus' command to view the current SELinux status:

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

The 'setenforce' command may be used to switch between **Enforcing** and **Permissive** modes on the fly but note that these changes do not persist through a system reboot.

To make changes persistent through a system reboot, edit the 'SELINUX=' line in `/etc/selinux/config` for either 'enforcing', 'permissive', or 'disabled'. For example: 'SELINUX=permissive'

Note: When switching from **Disabled** to either **Permissive** or **Enforcing** mode, it is highly recommended that the system be rebooted and the filesystem relabeled.

3. SELinux Policy

As noted, SELinux follows the model of least-privilege; by default everything is denied and then a policy is written that gives each element of the system only the access required to function. This description best describes the **strict** policy. However, such a policy is difficult to write that would be suitable in the wide range of circumstances that a product such as Enterprise Linux is likely to be used. The end result is that SELinux is likely to cause problems for system administrators and end users and rather than resolve these issues, system administrators may just disable SELinux thereby defeating the built-in protections.

By design, SELinux allows different policies to be written that are interchangeable. The default policy in CentOS 4, 5 and 6 is the **targeted** policy which "targets" and confines selected system processes. In CentOS 4 only 15 defined targets existed

(including httpd, named, dhcpd, mysqld). Later, in CentOS 5 this number had risen to over 200 targets.

All other system processes and all remaining userspace programs, as well as any in-house applications, that is everything else on the system, runs in an **unconfined** domain and is not covered by the SELinux protection model.

One goal might be for every process that is installed and, by default, running at boot should be run in a confined domain. The **targeted** policy is designed to protect as many key processes as possible without adversely affecting the end user experience and most users should be totally unaware that SELinux is even running.

4. SELinux Access Control

SELinux has 3 forms of access control:

- **Type Enforcement (TE):** Type Enforcement is the primary mechanism of access control used in the **targeted** policy
- **Role-Based Access Control (RBAC):** Based around SELinux users (not necessarily the same as the Linux user), but not used in the default **targeted** policy
- **Multi-Level Security (MLS):** Not commonly used and often hidden in the default **targeted** policy.

All processes and files have an SELinux security context. Lets see these in action by looking at the SELinux security context of the Apache homepage: '/var/www/html/index.html'

```
$ ls -Z /var/www/html/index.html
-rw-r--r--  username username system_u:object_r:httpd_sys_content_t /var/www/html/index.html
```

Note: The -Z switch will work with most utilities to show SELinux security contexts (e.g, 'ls -Z', 'ps axZ' etc).

In addition to the standard file permissions and ownership, we can see the SELinux security context fields: system_u:object_r:httpd_sys_content_t.

This is based upon user:role:type:mls. In our example above, user:role:type fields are displayed and mls is hidden. Within the default **targeted** policy, **type** is the important field used to implement Type Enforcement, in this case httpd_sys_content_t.

Now consider the SELinux security context of the Apache web server process: 'httpd'

```
$ ps axZ | grep httpd
system_u:system_r:httpd_t        3234 ?        Ss     0:00 /usr/sbin/httpd
```

Here we see the from the type field that Apache is running under the httpd_t type domain.

Finally, lets look at the SELinux security context of a file in our home directory:

```
$ ls -Z /home/username/myfile.txt
-rw-r--r--  username username user_u:object_r:user_home_t      /home/username/myfile.txt
```

where we see the type is user_home_t, the default type for files in a user's home directory.

Access is only allowed between similar types, so Apache running as httpd_t can read /var/www/html/index.html of type httpd_sys_content_t. Because Apache runs in the httpd_t domain and does not have the userid:username, it can not access **/home/username/myfile.txt** even though this file is world readable because **/home/username/myfile.txt** SELinux security context is not of type httpd_t. If Apache were to be exploited, assuming for the sake of this example that the **root** account right needed to effect a SELinux re-labeling into another context were not obtained, it would not be able to start any process not in the httpd_t domain (which prevents escalation of privileges) or access any file not in an httpd_t related domain.

5. Troubleshooting SELinux

Sooner or later you may run into situations where SELinux denies access to something and you need to troubleshoot the issue. There are a number of fundamental reasons why SELinux may deny access to a file, process or resource:

- A mislabeled file.
- A process running under the wrong SELinux security context.
- A bug in policy. An application requires access to a file that wasn't anticipated when the policy was written and generates an error.
- An intrusion attempt.

The first 3 we can deal with, whereas giving alarm and notice in the 4th case is exactly the intended behaviour.

To troubleshoot any issue, the log files are key and SELinux is no different. By default SELinux log messages are written to **/var/log/audit/audit.log** via the Linux Auditing System auditd, which is started by default. If the auditd daemon is not running, then messages are written to /var/log/messages . SELinux log messages are labeled with the "AVC" keyword so that they might be easily filtered from other messages, as with **grep**.

Starting with CentOS 5 the SELinux Troubleshooting tool can be used to help analyze log files converting them into a more human-readable format. The tool consists of a GUI tool for displaying messages in human-readable format and possible solutions, a desktop notification icon alerting of new issues and a daemon process, setroubleshootd, that checks for new SELinux AVC alerts and feeds the notification icon. Email notifications may also be configured, as for those not running an X server. The SELinux Troubleshooting tool is provided by the **setroubleshoot** package. The tool may be launched from the X Window GUI manager System menu or from the command line:

```
sealert -b
```

Those not running an X server may generate human-readable reports from the command line:

```
sealert -a /var/log/audit/audit.log > /path/to/mylogfile.txt
```

5.1. Relabeling Files

The 'chcon' command may be used to change SELinux security context of a file or files/directories in a similar way to how 'chown' or 'chmod' may be used to change the ownership or standard file permissions of a file.

Let's look at some examples.

Using Apache as an example, suppose you want to change the DocumentRoot to serve web pages from a location other than the default **/var/www/html/** directory. Assume we create a directory (or maybe a mount point) at **/html/** and create an **index.html** file there:

```
# mkdir /html
# touch /html/index.html
# ls -Z /html/index.html
-rw-r--r--  root root user_u:object_r:default_t        /html/index.html
# ls -Z | grep html
drwxr-xr-x  root root user_u:object_r:default_t        html
```

We see that both the directory **/html/** and file **/html/index.html** have the security context type: default_t. If we start our web browser and try to view the page, SELinux will properly deny access and log the error because the directory and file(s) have the wrong security context. We need to set the correct security context type for Apache of: httpd_sys_content_t.

```
# chcon -v --type=httpd_sys_content_t /html
```

```
context of /html changed to user_u:object_r:httpd_sys_content_t
# chcon -v --type=httpd_sys_content_t /html/index.html
context of /html/index.html changed to user_u:object_r:httpd_sys_content_t
# ls -Z /html/index.html
-rw-r--r--  root root user_u:object_r:httpd_sys_content_t    /html/index.html
# ls -Z | grep html
drwxr-xr-x  root root user_u:object_r:httpd_sys_content_t    html
```

Equally we could have set both in one go using the -R recursive switch:

```
# chcon -Rv --type=httpd_sys_content_t /html
```

Modifying security contexts in this manner will persist between system reboots but only until the modified portion of the filesystem is relabeled. This is a not uncommon operation and the proper solution, after testing, is to write a local custom rule (a so-called Policy Module) and merge it into the base local rules. This will be an additional rule on top of the 200+ rules mentioned above. To make the security context changes permanent, even through a complete filesystem relabel, we can use the SELinux Management Tool or the 'semanage' command from the command line:

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

to add a file context of type httpd_sys_content_t for everything under /html.

5.2. Restore Default Security Contexts

The 'restorecon' command may be used to restore file(s) default SELinux security contexts.

Again, lets use Apache as an example. Suppose a user edits a copy of index.html in his/her home directory and moves (mv) the file to the DocumentRoot /var/www/html. Whilst the copy (cp) command will typically adopt the destination directory's or file's security context, move (mv) will maintain the source's security context. We could use the 'chcon' command to change the security context of the file(s) in question but as the file(s) are now in the default Apache DocumentRoot (/var/www/html) we can just restore the default security contexts for that directory or file(s). To restore just the index.html file, we would use:

```
# restorecon -v /var/www/html/index.html
```

or to recursively restore the default security contexts for the whole directory:

```
# restorecon -Rv /var/www/html
```

Additionally, if we simply wanted to examine the security contexts of the /var/www/html directory to see if any files needed their security contexts restored, we can use restorecon with the -n switch to prevent any relabelling occurring:

```
# restorecon -Rv -n /var/www/html
```

5.3. Relabel Complete Filesystem

Sometimes it is necessary to relabel the complete filesystem although this should only be necessary when enabling SELinux after it has been disabled or when changing the SELinux policy from the default **targeted** policy to **strict**. To automatically relabel the complete filesystem upon reboot, do:

```
# touch /.autorelabel
# reboot
```

Sometimes a complete filesystem relabel will fail if the system has been upgraded to CentOS-5.2 with SELinux disabled, and SELinux is then enabled. If the above procedure doesn't correctly perform a complete filesystem relabel, try issuing the 'genhomedircon' command first:

```
# genhomedircon
```

```
# touch /.autorelabel
# reboot
```

5.4. Allowing Access to a Port

We may want a service such as Apache to be allowed to bind and listen for incoming connections on a non-standard port. By default, the SELinux policy will only allow services access to recognized ports associated with those services. If we wanted to allow Apache to listen on tcp port 81, we can add a rule to allow that using the 'semanage' command:

```
# semanage port -a -t http_port_t -p tcp 81
```

A full list of ports that services are permitted access by SELinux can be obtained with:

```
# semanage port -l
```

6. Customizing SELinux Policies

Minor modifications to SELinux policies can be made without modifying and recompiling the policy source by setting boolean values for optional features. Such features include allowing users to share their home directories under Samba or allowing Apache to serve files from users home directories which would otherwise be denied by the SELinux policy.

There is a separate Wiki page[1] dealing with booleans.

7. Creating Custom SELinux Policy Modules with audit2allow

Sometimes there are occasions when none of the above methods deal with a given situation and we need to extend the SELinux policy by creating a custom policy module to allow for a certain set of conditions. For example, consider the postgrey[2] service add-on for an smtp mail server. Our smtp server needs to communicate with postgrey over a Unix socket and that is something the default SELinux policy for our smtp server does not allow. Consequently the service is blocked by SELinux. This is an issue that can not be fixed by changing or restoring file type security contexts and isn't something that has a boolean value we can toggle to allow. We could disable SELinux protection of the smtp server through a boolean, which would be better than disabling SELinux completely, but that is still far from ideal.

If we switch SELinux into Permissive mode and run our mail server for a set period of time, we can log SELinux issues whilst still permitting access. Checking our logs, we see the following SELinux AVC messages:

```
type=AVC msg=audit(1218128130.653:334): avc:  denied  { connectto } for  pid=9111 comm="smtpd" path="/var/spool/postfix/postgrey/socket"
scontext=system_u:system_r:postfix_smtpd_t:s0 tcontext=system_u:system_r:initrc_t:s0 tclass=unix_stream_socket
type=AVC msg=audit(1218128130.653:334): avc:  denied  { write } for  pid=9111 comm="smtpd" name="socket" dev=sda6 ino=39977017
scontext=system_u:system_r:postfix_smtpd_t:s0 tcontext=system_u:object_r:postfix_spool_t:s0 tclass=sock_file
```

Then we can use 'audit2allow' to generate a set of policy rules that would allow the required actions. We can generate a local postgrey Type Enforcement policy file (postgreylocal.te):

```
# grep smtpd_t /var/log/audit/audit.log | audit2allow -m postgreylocal > postgreylocal.te
# cat postgreylocal.te
module postgreylocal 1.0;
require {
        type postfix_smtpd_t;
        type postfix_spool_t;
        type initrc_t;
        class sock_file write;
```

```
        class unix_stream_socket connectto;
}
#============= postfix_smtpd_t ==============
allow postfix_smtpd_t initrc_t:unix_stream_socket connectto;
allow postfix_smtpd_t postfix_spool_t:sock_file write;
```

Above we see that we can grep the audit.log file for issues relating to our smtp server and pipe those issues to audit2allow which generates a set of rules that it thinks would permit the actions currently denied by the SELinux policy. Reviewing these rules we see our smtp server wants to connect and write to a Unix socket which we see from out logs is the Unix socket that the postgrey service is listening on. As this seems perfectly reasonable, we can go ahead and use audit2allow to make a custom policy module to allow these actions:

```
# grep smtpd_t /var/log/audit/audit.log | audit2allow -M postgreylocal
```

We then load our postgrey policy module using the 'semodule' command into the current SELinux policy:

```
semodule -i postgreylocal.pp
```

which will add our postgrey policy module to /etc/selinux/targeted/modules/active/modules/postgreylocal.pp. We can check the policy module loaded correctly by listing loaded modules with 'semodule -l'.

We can then continue to monitor our SELinux log files to check that our custom policy module works and once we are satisfied we can re-enable SELinux Enforcing mode and again benefit from SELinux protection of our now fully functional smtp server.

7.1. Manually Customizing Policy Modules

Often audit2allow will automatically create a custom policy module that will resolve a particular issue, but there are times when it doesn't get it quite right and we may want to manually edit and compile the policy module. For example, consider the following AVC audit log:

```
Summary:
SELinux is preventing postdrop (postfix_postdrop_t) "getattr" to
/var/log/httpd/error_log (httpd_log_t).
Detailed Description:
SELinux denied access requested by postdrop. It is not expected that this access
is required by postdrop and this access may signal an intrusion attempt. It is
also possible that the specific version or configuration of the application is
causing it to require additional access.
Allowing Access:
Sometimes labeling problems can cause SELinux denials. You could try to restore
the default system file context for /var/log/httpd/error_log,
restorecon -v '/var/log/httpd/error_log'
If this does not work, there is currently no automatic way to allow this access.
Instead, you can generate a local policy module to allow this access - see FAQ
(http://fedora.redhat.com/docs/selinux-faq-fc5/#id2961385) Or you can disable
SELinux protection altogether. Disabling SELinux protection is not recommended.
Please file a bug report (http://bugzilla.redhat.com/bugzilla/enter_bug.cgi)
against this package.
Additional Information:
Source Context                system_u:system_r:postfix_postdrop_t
Target Context                root:object_r:httpd_log_t
Target Objects                /var/log/httpd/error_log [ file ]
Source                        postdrop
Source Path                   /usr/sbin/postdrop
```

```
Port                        <Unknown>
Host                        sanitized
Source RPM Packages         postfix-2.3.3-2
Target RPM Packages
Policy RPM                  selinux-policy-2.4.6-137.1.el5
Selinux Enabled             True
Policy Type                 targeted
MLS Enabled                 True
Enforcing Mode              Enforcing
Plugin Name                 catchall_file
Host Name                   sanitized
Platform                    Linux sanitized 2.6.18-53.1.21.el5 #1 SMP Tue
                            May 20 09:35:07 EDT 2008 x86_64 x86_64
Alert Count                 599
First Seen                  Wed Jul  2 08:27:15 2008
Last Seen                   Sun Aug 10 22:47:52 2008
Local ID                    c303a4ea-8e7a-4acc-9118-9cc61c6a2ec8
Line Numbers
Raw Audit Messages
host=sanitized type=AVC msg=audit(1218397672.372:352): avc:  denied  { getattr } for  pid=4262 comm="postdrop"
path="/var/log/httpd/error_log" dev=md2 ino=117005 scontext=system_u:system_r:postfix_postdrop_t:s0
tcontext=root:object_r:httpd_log_t:s0 tclass=file
host=sanitized type=SYSCALL msg=audit(1218397672.372:352): arch=c000003e syscall=5 success=no exit=-13 a0=2
a1=7fffd6febca0 a2=7fffd6febca0 a3=0 items=0 ppid=4261 pid=4262 auid=4294967295 uid=48 gid=48 euid=48 suid=48
fsuid=48 egid=90 sgid=90 fsgid=90 tty=(none) comm="postdrop" exe="/usr/sbin/postdrop"
subj=system_u:system_r:postfix_postdrop_t:s0 key=(null)
```

Running audit2allow on the above error, and reviewing the resultant postfixlocal.te policy file we see:

```
# grep postdrop /var/log/audit/audit.log | audit2allow -M postfixlocal
# cat postfixlocal.te
    module postfixlocal 1.0;
    require {
            type httpd_log_t;
            type postfix_postdrop_t;
            class dir getattr;
            class file { read getattr };
    }
    #============= postfix_postdrop_t ==============
    allow postfix_postdrop_t httpd_log_t:file getattr;
```

Hopefully the first thing to strike us here is why does postdrop needs access to /var/log/httpd/error_log? Presumably this isn't something we would expect so we have to assess if this is an action we wish to allow or not. We have a number of options. We could just ignore the error and allow SELinux to continue blocking and logging access attempts or we could allow the action by creating the custom policy module as suggested by audit2allow. Alternatively we can edit the custom policy module .te file to prevent auditing of this particular error whilst still allowing SELinux to continue preventing access. We do this by editing the **allow** line, changing it to **dontaudit**:

```
    #============= postfix_postdrop_t ==============
    dontaudit postfix_postdrop_t httpd_log_t:file getattr;
```

Now we can manually compile and load the edited custom policy module:

```
# checkmodule -M -m -o postfixlocal.mod postfixlocal.te
# semodule_package -o postfixlocal.pp -m postfixlocal.mod
# semodule -i postfixlocal.pp
```

Access to /var/log/httpd/error_log by postdrop will still be prevented by SELinux but we won't receive constant alerts and error messages filling up our log files each time access is blocked.

8. Summary

This article is intended to give an overview of working with SELinux for users new to SELinux. SELinux is installed and enabled by default, and for most users it will function without issue affording an enhanced level of security. SELinux is suitable for all classes of installation including servers, workstations, desktops and laptops.

Although SELinux can appear quite daunting and complex to users not familiar with it, that is no reason to disable it at installation. If SELinux does present issues then it is easy to switch into Permissive mode at which point issues are only logged and not blocked. When issues do arise the techniques presented in this article can be used to troubleshoot and resolve them.

9. Additional Resources

http://fedoraproject.org/wiki/SELinux[3]

http://docs.fedoraproject.org/en-US/Fedora/13/html-single/Security-Enhanced_Linux/[4]

http://danwalsh.livejournal.com/[5]

10. User Notes and Gotchas

This section is provided by a user who learned most of what he knows of SELinux from this document. This document is a wonderful and detailed resource. However, it is somewhat dry. It misses a couple of practical points I found rather frustrating as I tried to actually get stuff done. Note this is all for CentOS 6.

1. *semanage* is found in the package *policycoreutils-python* which is not installed by default. Note there is a separate *policycoreutils* package.
2. Finding the right context to use as you manage a system is difficult. One place to start is *ls -Z*. Look at the directories and data pre-installed by a package and copy the contexts already used. The next tool is *seinfo -t* which lists all contexts currently in use on your system. grep for the name of your application.
3. Don't forget about the -t parameter to *chcon*. It sets just the type context which is generally all you want to do and is easier than specifying the entire string as reported by *ls -Z*.
4. *audit2allow* is actually easier to use than presented here. When you have a conflict between two contexts, find the error messages in *audit.log* and extract them to a separate text file. Feed the errors back to *audit2allow* like this:

```
audit2allow -M mynewpolicyname <errors.txt
```

- What will be generated is *mynewpolicyname.te* and *mynewpolicyname.pp* along with helpful instructions on how to import the new policy. This new policy will enable whatever was previously conflicting.

  I discovered this process attempting to get a script running under postfix that was previously installed on a non-SELinux system. Under SELinux, the script needs to run under the *postfix_pipe_exec_t* context and it's spooling directory needs the *postfix_pipe_tmp_t* context. However the script also calls the *spamc* binary of *spamassassin* for processing. Alas, this binary runs under *spamc_t* and thus couldn't read or write into the spool directory.

  I found two error messages in *audit.log*: one for spamc_t trying to read in the spool directory and one for trying to write. After executing this process on the extracted error messages we get a *.te* file that looks like this:

```
module mynewpolicy 1.0;

require {
        type spamc_t;
        type postfix_pipe_tmp_t;
        class file { read write };
}

#============= spamc_t ==============
allow spamc_t postfix_pipe_tmp_t:file { read write };
```

- If you look at the last line, this policy allows the *spamc_t* context to read and write files in the *postfix_pipe_tmp_t* context. *spamassassin* now works as before.

1. http://wiki.centos.org/TipsAndTricks/SelinuxBooleans
2. http://wiki.centos.org/HowTos/postgrey
3. http://fedoraproject.org/wiki/SELinux
4. http://docs.fedoraproject.org/en-US/Fedora/13/html-single/Security-Enhanced_Linux/
5. http://danwalsh.livejournal.com/