How I stopped worrying and learned to love SELinux

Introduction

I have spent at least 10 years with several misconceptions about SELinux. With most of the systems that I manage, I have purposely gone through and disabled SELinux on them. I know, shame on me. This is the kind of thing that would be up there with Game of Thrones and the scene with "Shame .. Shame .. Shame". A good friend of mine that is also security savvy at one point taught me how he was trained on creating security policies with SELinux somewhere around 2010.

The challenge I ran into in the real world, especially on production systems, is that I needed the application(s) working, not acting weirdly and later finding that SELinux was blocking some object that ties back to the application. My rationale was [past tense pointed out here], when time was of critical essence for keeping everything working, and with limited resources, I chose the easier path to disable SELinux on all systems because of my lack of knowledge with the subject matter.

Working towards my RHCE, I recently discovered updates to SELinux that made me step back and re-evaluate the situation. What I didn't realize in the last decade is that the engineers that maintain SELinux have made huge strides in making it easier to discover what is blocking an object, but also how to fix them quickly.

In fact, SELinux and its policy set have now become so transparently powerful that SELinux functions as a behavior-based antivirus system (AV). The default policies automatically protect all actions by installable programs (at least on Red Hat Enterprise Linux). After a developer has validated functionality of an application, abnormal accesses by malware are blocked and flagged. This 'live action' protection stands in contrast to the stale and bypassable signature-based AV from days of past. In many cases, this protection has made SELinux become declared as the Operating System baseline function for AV in many organizations.

This paper is my give back to the community to illuminate others on the advancements within SELinux built-in policies and how rock-solid it is today. Disabling SELinux whether in dev or production at this point in time is not only a bad choice on a system administrator's part, but more so a companies choice in regards to policies and guidelines because the decisions bring real world consequences with it in regards to security compliance and regulations [what I am alluding to is that companies need to mandate in their policies and guidelines that SELinux will not be disabled on any Linux system with consequences up to termination – caveat; setting Permissive mode is fine for short bursts of time in order to troubleshoot underlying problems]. Imagine sitting in front of Congress and trying to explain why it was easier to disable SELinux and then the subsequent data loss your company faced. If you are not worried about Congress, how about the trust of your customers in your ability to protect their data. My answer to both is, "No thanks! I will happily adjust SELinux and keep it working correctly."

Getting started

If you are not comfortable with SELinux or have no idea what is going on with it, I would recommend that you start here:
https://www.redhat.com/en/topics/linux/what-is-selinux

And then this is my go to source for all things SELinux:
https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/selinux_users_and_administrators_guide/chap-security-enhanced_linux-introduction

If you need a fun break, this small coloring book that you can print is a lot of fun to go through and conceptualize what is happening with SELinux. <jokingly> It is Fun for the whole family.
https://people.redhat.com/duffy/selinux/selinux-coloring-book_A4-Stapled.pdf

Terminology

Type Enforcement (TE): Type Enforcement is the primary mechanism of access control used in the **targeted** policy.

Context: Label on processes, files, and ports that determines access. Access this with use of the -Z option on the end of certain SELinux-enabled commands like, ls and ps.

Tags: Unique names.

Booleans: Booleans allow parts of SELinux policy to be changed at runtime, without any knowledge of SELinux policy writing.

Role-Based Access Control (RBAC): Based around SELinux users (not necessarily the same as the Linux user), but not used in the default configuration of the **targeted** policy.

Multi-Level Security (MLS): Not commonly used and often hidden in the default **targeted** policy.

States and Modes

You can use on a running system, as the root user (to show the active state of SELinux):

```
# getenforce
Enforcing
```

To change the current state to Permissive, run:

```
# setenforce 0
# getenforce
Permissive
```

To change back to enforcing mode, run:

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

My preferred way of controlling SELinux is the /etc/selinux/config file. Use the editor of your choice, vi, vim, emacs, or nano to edit this file as the root user.

```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#     enforcing - SELinux security policy is enforced.
#     permissive - SELinux prints warnings instead of enforcing.
#     disabled - No SELinux policy is loaded.
#SELINUX=permissive
SELINUX=enforcing
# SELINUXTYPE= can take one of three values:
#     targeted - Targeted processes are protected,
#     minimum - Modification of targeted policy. Only selected processes are
protected.
#     mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

I duplicated the line with `SELINUX=enforcing` and then set to `SELINUX=permissive`.

This allows me to semi-quickly change between states by commenting out one line or the other.
Using this method would require a reboot to change the state.
However, this is my preferred method because then I know what state that SELinux is in and my older brain likes to keep things reasonably simple. Use the method that works for you.

That is the basics for enabling and disabling SELinux.

A colleague pointed out to me that SELinux does not need a specific daemon to function since the enforcement happens within the kernel.

To see the status of the daemon, run:

```
# sestatus
```

Output:

```
SELinux status:                 enabled
SELinuxfs mount:                /sys/fs/selinux
SELinux root directory:         /etc/selinux
```

```
Loaded policy name:             targeted
Current mode:                   enforcing
Mode from config file:          enforcing
Policy MLS status:              enabled
Policy deny_unknown status:     allowed
Max kernel policy version:      31
```

In the main config file (`/etc/selinux/config`), our preferrred method to use is `targeted` mode. My reasoning behind this is Red Hat has put enormous resources into updating the database for files and booleans that support this mode. That translates into saved time on your part so that you can use predefined controls for most popular software without putting any (much) effort into maintaining said. Multi-Level Security (MLS) is really for hardcore systems that can either bridge the gap between classified networks or have users with many different levels of clearance work on the same system. This is very similar to Trusted Solaris. Most companies don't need this level of security because of the engineering + security cost that are associated with putting this into operational mode with certified accreditation. I didn't mention `minimum` mode, because I can't envision a valid use case for this mode [if this prompts the reader to Google `SELinux minimum mode`, go for it]. That stated, stick with the `targeted` mode. What we are looking to achieve with this is to minimize the amount of time we have to mess with adding to the pre-existing security database that comes as the default with `targetted` mode.

The old way

I feel this is worth mentioning here because I believe some people still use this method. One of the methods that you can use is to search for the abbreviation `avc` in the file `/var/log/audit/audit.log`. AVC stands for Access Vector Cache.

In the following command with grep, we are performing a text search of the audit log file for the string "avc" in order to capture any alerts that have occurred.

As root, run the following command:

```
# grep -i "avc" /var/log/audit/audit.log
```

On my Fedora 30 box that I am writing this on, I have a great example that I found.

```
type=AVC msg=audit(1562933511.443:90): avc:  denied  { create } for  pid=976
comm="vboxdrv.sh" name="vbox-setup.log" scontext=system_u:system_r:init_t:s0
tcontext=system_u:object_r:var_log_t:s0 tclass=file permissive=0
```

So the old way to solve this would be either to grep for avc in the above file and then pipe out to the command `audit2allow`, OR an admin could go through and echo each line (with `avc`) individually and run that through the command `audit2allow`. I personally don't like the idea of using a blanket policy to allow any denied operation that gets logged in the `audit.log` file. This action to me is a really bad method in the sense that I believe would lead to blindly allowing potentially bad applications to behave in a way that you don't want running on your system. Again for clarification, what I do per denied object that fails and get's logged is to go back and verify, one "YES" that application is supposed to be running on my system, and two "YES" the access to the new resource is supposed to happen.

As an example, let's run it through this way, wrapped in single quotes.

```
# grep -E 'type=AVC' | grep -E 'comm="vboxdrv.sh"' audit.log | audit2allow
```

This gives us this output:

```
#============= init_t ==============

#!!!! This avc can be allowed using the boolean 'use_virtualbox'
allow init_t var_log_t:file create;
```

I could easily rerun the previous command and redirect to a file like, `/root/SELinux/vboxdrv.pp`, and then run the command:

```
# semodule -i /root/SELinux/vboxdrv.pp
```

and add to the current policy. However, if you carefully read what the output of audit2allow says, we already have a **boolean** setup for us and we just need to activate it. Let's discover the current state right now with the following command:

```
# semanage boolean -l | grep 'use_virtualbox'
use_virtualbox                 (off  ,  off)  Allow use to virtualbox
```

We can see within the parentheses that both the current and on-start values are set to off. Let's activate said with the next command:

```
# semanage boolean -m --on use_virtualbox
```

Re-run the `–l` version to see the status:

```
# semanage boolean -l | grep 'use_virtualbox'
use_virtualbox                 (on   ,   on)  Allow use to virtualbox
```

To summarize this section, the Old Way is to create a policy file per single or set of AVCs that correlates to an application, and then run `semodule -i ./file.pp` to import that into the SELinux database.

The New Way

First, make sure the following packages are installed to make your life easier with managing SELinux. These packages will give us automated tools we can use to discover SELinux errors in the audit.log as well as in /var/log/messages file.

- `policycoreutils`
- `policycoreutils-devel`
- `policycoreutils-python`
- `rsyslog`

- `selinux-policy`
- `selinux-policy-devel`
- `selinux-policy-targeted`
- `setroubleshoot`
- `setroubleshoot-server`

The following command executes as a one-liner to save you time. You could run as root:

```
# yum install -y policycoreutils policycoreutils-devel policycoreutils-python rsyslog
selinux-policy selinux-policy-devel selinux-policy-targeted setroubleshoot
setroubleshoot-server
```

SELinux has a tool called sealert that analyzes the audit log. Sealert will scan the log file and will then generate a report containing all discovered SELinux issues.

Next enable the following services to make discovery of sealerts easier:

```
# systemctl enable auditd
# systemctl enable rsyslog
```

The new way to me is very simple, when compared to the Old Way. Run the following command:

```
# grep -i 'selinux is preventing' /var/log/messages
```

Output:

```
Sep 22 15:58:38 rhel7 python: SELinux is preventing httpd from getattr access on the
file /custom/index.html.#012#012*****  Plugin catchall_labels (83.8 confidence)
suggests   *******************#012#012If you want to allow httpd to have getattr
access on the index.html file#012Then you need to change the label on
/custom/index.html#012Do#012# semanage fcontext -a -t FILE_TYPE
'/custom/index.html'#012where FILE_TYPE is one of the following:
NetworkManager_exec_t, NetworkManager_log_t, NetworkManager_tmp_t,
abrt_dump_oops_exec_t, abrt_etc_t, abrt_exec_t, abrt_handle_event_exec_t,
.. abbreviated for brevity ..
load_policy_exec_t, loadkeys_exec_t, locale_t, locate_exec_t, lockdev_exec_t,
login_exec_
```

The first part that matters is this piece of information:

```
semanage fcontext -a -t FILE_TYPE '/custom/index.html'
```

That is critical because it tells us what is going on with the block. In this case, we see that `/custom/index.html` is being blocked by SELinux.

The next part of this, is what Context should you apply to make this work.

Run this command:

```
# grep -i 'sealert' /var/log/messages
```

Output:

```
Sep 22 15:58:25 rhel7 setroubleshoot: SELinux is preventing httpd from getattr access
on the file /custom/index.html. For complete SELinux messages run: sealert -l
b788e70e-4e02-4a14-8a8c-895a7156209e
```

You will see in the output that you need to run:

# sealert -l b788e70e-4e02-4a14-8a8c-895a7156209e

Doing so produces:

```
SELinux is preventing httpd from getattr access on the file /custom/index.html.

*****  Plugin catchall_labels (83.8 confidence) suggests   *******************

If you want to allow httpd to have getattr access on the index.html file
Then you need to change the label on /custom/index.html
Do
# semanage fcontext -a -t FILE_TYPE '/custom/index.html'
where FILE_TYPE is one of the following: NetworkManager_exec_t, NetworkManager_log_t,
NetworkManager_tmp_t, abrt_dump_oops_exec_t, abrt_etc_t, abrt_exec_t,
abrt_handle_event_exec_t, abrt_helper_exec_t, abrt_retrace_coredump_exec_t,
… httpd_cache_t, httpd_config_t, httpd_exec_t,
 httpd_helper_exec_t, httpd_keytab_t, httpd_lock_t, httpd_log_t, httpd_modules_t,
httpd_passwd_exec_t, httpd_php_exec_t, httpd_php_tmp_t, httpd_rotatelogs_exec_t,
httpd_squirrelmail_t, httpd_suexec_exec_t, htt
pd_suexec_tmp_t, httpd_sys_content_t, httpd_sys_htaccess_t,
httpd_sys_ra_content_t, httpd_sys_rw_content_t, httpd_sys_script_exec_t, httpd_tmp_t,
httpd_tmpfs_t, httpd_unconfined_script_exec_t, httpd_user_htacc
ess_t, httpd_user_ra_content_t, httpd_user_rw_content_t, httpd_user_script_exec_t,
httpd_var_lib_t, httpd_var_run_t …
zoneminder_content_t, zoneminder_exec_t, zoneminder_htaccess_t, zoneminder_log_t,
zoneminder_ra_content_t, zoneminder_rw_content_t, zoneminder_script_exec_t,
zoneminder_var_lib_t, zos_remote_exec_t.
Then execute:
restorecon -v '/custom/index.html'


*****  Plugin catchall (17.1 confidence) suggests   **************************

If you believe that httpd should be allowed getattr access on the index.html file by
default.
Then you should report this as a bug.
You can generate a local policy module to allow this access.
Do
allow this access for now by executing:
# ausearch -c 'httpd' --raw | audit2allow -M my-httpd
# semodule -i my-httpd.pp


Additional Information:
Source Context                system_u:system_r:httpd_t:s0
Target Context                unconfined_u:object_r:default_t:s0
Target Objects                /custom/index.html [ file ]
Source                        httpd
Source Path                   httpd
…
Local ID                      b788e70e-4e02-4a14-8a8c-895a7156209e

Raw Audit Messages
type=AVC msg=audit(1569167899.855:231): avc:  denied  { getattr } for  pid=4828
comm="httpd" path="/custom/index.html" dev="dm-0" ino=35590924
scontext=system_u:system_r:httpd_t:s0 tcontext=unconfined_u:object_r:default_t:s0
tclass=file permissive=0


Hash: httpd,httpd_t,default_t,file,getattr
```

Bear with me, this looks like a complicated mess. It is not.

I know from years of experience that the default base directory for Apache web server is, /var/www/html, so let's check the default context for that dir.

I know, from this command:

```
# ls -ld -Z /var/www/html
drwxr-xr-x. root root system_u:object_r:httpd_sys_content_t:s0 /var/www/html
```

That the default context for my base directory that comes with the application is `httpd_sys_content_t`. That context also aligns with the context provided in the sealert -l output above, with the bold and increased font size.

Now that we know two pieces of information, we are going to fix the file system/directory so that we don't have to deal with this again.

Update the SELinux database with the correct context:
```
# semanage fcontext -a -t httpd_sys_conn_t '/custom(/.*)?'
```

Then restore the new context onto the existing directory structure:
```
# restorecon -Rfv /custom
```

Output:
```
restorecon reset /custom context unconfined_u:object_r:default_t:s0-
>system_u:object_r:httpd_sys_content_t:s0
restorecon reset /custom/index.html context unconfined_u:object_r:default_t:s0-
>system_u:object_r:httpd_sys_content_t:s0
```

What you need to memorize are:
```
  httpd_sys_content_t
```
AND
```
  (/.*)?
```

Over time, you will build your memory of the handful of SELinux contexts you need to fix the file system and keep everything in functional order. The second part is the REGEX that you need to recursively include all files and directories in a directory structure.

More Tools of the Trade:
Another gem is `matchpathcon`, and can be used as:

```
# /usr/sbin/matchpathcon /var/www/html
/var/www/html system_u:object_r:httpd_sys_content_t:s0
```

Matchpathcon comes with the base OS in the package, libselinux-utils. This tool will save you having to remember all of the contexts. However, you need to know how to map it to the base directory that the developer mapped with the package. In the case of using Apache web server, I know the package is httpd, so I could use rpm to find the configuration file. As in:

```
# rpm -ql httpd | grep '\.conf'
...
/etc/httpd/conf/httpd.conf
...
```

Looking through the httpd.conf file, I see:

```
# DocumentRoot: The directory out of which you will serve your
# documents. By default, all requests are taken from this directory, but
# symbolic links and aliases may be used to point to other locations.
#
DocumentRoot "/var/www/html"
```

Therefore giving me the base directory that the developer(s) intended for documents to go into. With that knowledge, I can now use that file context on my custom directory under /custom.

Finally, I would be remiss if I did not talk about semanage and listing all of the built-in file context(s). Use the command, `semanage fcontext -l` to show all patterns in the local database.

```
# semanage fcontext -l  | head -22
SELinux fcontext                                   type               Context

/.*                                                all files          system_u:object_r:default_t:s0
/[^/]+                                             regular file       system_u:object_r:etc_runtime_t:s0
/a?quota\.(user|group)                             regular file       system_u:object_r:quota_db_t:s0
/nsr(/.*)?                                         all files          system_u:object_r:var_t:s0
/sys(/.*)?                                         all files          system_u:object_r:sysfs_t:s0
/xen(/.*)?                                         all files          system_u:object_r:xen_image_t:s0
/mnt(/[^/]*)?                                      directory          system_u:object_r:mnt_t:s0
/mnt(/[^/]*)?                                      symbolic link      system_u:object_r:mnt_t:s0
/bin/.*                                            all files          system_u:object_r:bin_t:s0
/dev/.*                                            all files          system_u:object_r:device_t:s0
/run/.*                                            all files          system_u:object_r:var_run_t:s0
/var/.*                                            all files          system_u:object_r:var_t:s0
/tmp/.*                                            all files          <<None>>
/usr/.*                                            all files          system_u:object_r:usr_t:s0
/srv/.*                                            all files          system_u:object_r:var_t:s0
/opt/.*                                            all files          system_u:object_r:usr_t:s0
/etc/.*                                            all files          system_u:object_r:etc_t:s0
/lib/.*                                            all files          system_u:object_r:lib_t:s0
/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
```

I normally like to run, `semanage fcontext -l | less` in order to search for my pattern. Don't forget, if you have forward slashes, you will need to escape each of those in your search with a backslash. e.g. `/\/var\/www`. Another option is to grep for the directory structure you want, e.g.

```
# semanage fcontext -l  | grep '/var/www' | head  -10
/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
/var/www/[^/]*/cgi-bin(/.*)?                       all files          system_u:object_r:httpd_sys_script_exec_t:s0
/var/www/svn(/.*)?                                 all files          system_u:object_r:httpd_sys_rw_content_t:s0
/var/www/git(/.*)?                                 all files          system_u:object_r:git_content_t:s0
/var/www/perl(/.*)?                                all files          system_u:object_r:httpd_sys_script_exec_t:s0
/var/www/wiki[0-9]?\.php                           regular file       system_u:object_r:mediawiki_content_t:s0
/var/www/wiki[0-9]?(/.*)?                          all files          system_u:object_r:mediawiki_rw_content_t:s0
/var/www/html(/.*)?/uploads(/.*)?                  all files          system_u:object_r:httpd_sys_rw_content_t:s0
/var/www/html(/.*)?/wp-content(/.*)?               all files          system_u:object_r:httpd_sys_rw_content_t:s0
/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
```

Recap

You have learned to set booleans to leverage the power of of the built-in library of predefined objects that you can quickly set and forget. The next part, which is my favorite is how to apply the correct context to a folder so that you can recursively apply the right permissions to a folder. When I saw this, it instantly clicked in my mind how powerful this is because with two pieces of information, I can resolve a directory's SELinux permissions and get the application working very quickly with SELinux. The hardest part to this is discovering the right SELinux context to apply (`matchpathcon` if your friend here).

From this point, I would recommend exploring Ansible and how to use it to apply SELinux in regards to fcontext and booleans via a playbook.

Here: https://docs.ansible.com/ansible/latest/modules/sefcontext_module.html
And Here: https://docs.ansible.com/ansible/latest/modules/seboolean_module.html

Refs:
https://wiki.centos.org/es/HowTos/SELinux
https://fedoraproject.org/wiki/SELinux
https://fedoraproject.org/wiki/SELinux/Troubleshooting
https://github.com/SELinuxProject/refpolicy/wiki

Deep dive on context names and mapping for Apache web server:
https://github.com/fedora-selinux/selinux-policy-contrib/blob/rawhide/apache.fc