(/) **Wiki**

SELinux/Tutorials/Where to find SELinux permission denial details

From Gentoo Wiki
< SELinux (/wiki/Special:MyLanguage/SELinux) | Tutorials (/wiki/Special:MyLanguage/SELinux/Tutorials)
Jump to:navigation Jump to:search

Contents

- 1 Where to find SELinux permission denial details
  - 1.1 SELinux logging
  - 1.2 Disecting the AVC denial
  - 1.3 Hidden denials
  - 1.4 Other ways to read denial information
    - 1.4.1 ausearch
    - 1.4.2 sealert
  - 1.5 What you need to remember

Where to find SELinux permission denial details

Now that you are aware that SELinux governs file access by verifying the security context of the process (the *domain*) and the context of the file, it is time to find out how, if SELinux denies a certain access, you can troubleshoot this in more detail.

SELinux logging

The **most important** feature of SELinux, and one you should start learning by heart, is that it is able to log *everything*. And with *everything*, I mean everything. If you want, we can have SELinux log all granted accesses (although I can imagine that becomes dull to look at, and might slow down the performance of the system), but more importantly it logs access denials.

The default location where you can find this logging depends a bit on the distribution, but generally it is either in `/var/log/avc.log` if you are not running the Linux audit daemon, and in `/var/log/audit/audit.log` or `/var/log/audit.log` if you are. This logging is very verbose, mainly because you need many details in order to troubleshoot problems.

But before we take a look at the denials, we also want to give you a heads up.

1. Not every denial you find in the logs is a problem by itself. Some denials are *cosmetic*, meaning that they do occur but do not influence the behavior of an application. This is often because of an application development malpractice (like not properly closing file descriptors) or because of high-level library functions where only a small fraction of the features are used by an application.
2. Denials are logged as they come along. That means that you will see a lot of denials, and although many will be related to each other (one denial leads to the other) many will also have nothing to do

with the problem you are investigating.

3. If there are too many denials succeeding each other, they might be suppressed by the Linux kernel; if that happens, you will get a message like the following, so if you find this message in your logs you have to understand that you are not seeing everything that SELinux might be reporting:

```
Mar 12 17:46:42 hpl kernel: [   14.453644] audit_printk_skb: 84 callbacks suppressed
```

So let's take a look at one denial. This one comes from the audit log (which you can tell from the start of the log, *type=AVC*, which you will not see in the `avc.log` as it is implied there).

```
type=AVC msg=audit(1363289005.532:184): avc:  denied  { read } for  pid=29199 comm="T
race"
name="online" dev="sysfs" ino=30 scontext=staff_u:staff_r:googletalk_plugin_t
tcontext=system_u:object_r:sysfs_t tclass=file
```

We warned you that it was verbose ;-)

Disecting the AVC denial

AVC stands for *Access Vector Cache*. Not that that is worth much right now, but the word *cache* does already give you feedback as to why the logs might not show everything if SELinux had too many things to report. Just thought it might interest you (since the term *avc denial* might show up in documentation). But let's get back to the denial we had.

```
type=AVC msg=audit(1363289005.532:184): avc:  denied  { read } for  pid=29199 comm="T
race"
name="online" dev="sysfs" ino=30 scontext=staff_u:staff_r:googletalk_plugin_t
tcontext=system_u:object_r:sysfs_t tclass=file
```

Once you get to know this denial structure, you can translate this into the following:

```
The Trace process with PID 29199 tried to read a file called online on
a file system hosted on the sysfs device. This file has inode number 30, and has the
security context system_u:object_r:sysfs_t assigned to it. The Trace process
itself is running with the staff_u:staff_r:googletalk_plugin_t context (domain).
```

You probably find the details from the translated sentence back in the denial easily, but I'm going to disect the denial part by part anyway. The following table gives a part by part explanation. We do want to tell you though that the logs can be a bit different based on what is denied - for instance, a file read access shows a few different parts than a socket connect denial. The majority of fields however will be present in all cases.

Log part	Name	Description
type=AVC	Log type	Only in the `audit.log` file; it informs the user what kind of audit log type this is. So in our case, it is an AVC log entry
msg=audit(1363289005.532:184)	Timestamp	Timestamp in seconds since *epoch*, meaning the number of seconds since January 1st, 1970. You can convert this to a more human readable format using **date -d @** followed by the number, like so: `user $ date -d @1363292159.532` `Thu Mar 14 21:15:59 CET 2013`
avc:	Log type (again)	Informs the user what kind of log entry this is. In this case, an AVC log entry.
denied	State (if enforced)	What SELinux did, which can be either *denied* or *granted*. Note that, if SELinux is in permissive mode (we'll talk about this later), then it will still log as *denied* even though it was allowed.
{ read }	Permission	The permission that was requested / executed. In this case, it is a read operation. Sometimes the permission contains a set (like *{ read write }* but in most cases, it is a single permission request).
for pid=29199	Process PID	The process identifier of the process that took the action (in this case, tried to read)
comm="Trace"	Process CMD	The process command (without arguments, and limited to 15 characters), which helps users identify what the process was in case the process is already gone (a PID is only useful if the process is still running)
name="online"	Target name	The name of the target (in this case, the file name). This field depends heavily on the target itself; it can also be *path*=, *capability*=, *src*= and more. But in those cases, its purposes should be clear from the rest of the log.
dev="sysfs"	Device	Device on which the target resides (in case of a file or file system). In this case, the device is *sysfs* so we have the hint immediately that this is for something inside `/sys`. Other valid examples are *dev=md-0*, *dev=sda1* or *dev=tmpfs*.

ino=30	inode number	The inode number of the target file. In this case, since we know it is on the *sysfs* file system (and thus in `/sys`), we can look for this file using *find*: **user $** `find /sys -xdev -inum 30` `/sys/devices/system/cpu/online`
scontext=staff_u:staff_r:googletalk_plugin_t	Source context	The security context of the process (the domain)
tcontext=system_u:object_r:sysfs_t	Target context	The security context of the target resource (in this case the file)
tclass=file	Target class	The class of the target. We have seen *file* already, and *dir* shouldn't surprise you either. SELinux supports a whole lot of classes, which we will describe later.

This is a lot to digest, but it is very important that you understand this log. When you have a *permission denied* error, you should look into the SELinux logs for avc denials to see if SELinux is the culprit or not.

Hidden denials

We've told you already that denial logs can be *cosmetic*. Since they do not reflect real problems, SELinux policy writers can *hide* those denials from regular logging so that users are not baffled by the various denials they get. This is done through *dontaudit* statements. A default SELinux policy in most distributions will already have a lot of such statements active, which you can verify through **seinfo**.

```
root # seinfo | grep audit
```

```
   Auditallow:          1    Dontaudit:        5341
```

In some cases, the SELinux policy writers can be wrong (of course, they are still human) so it might make sense to disable these *dontaudit* statements for a short while (while you are reproducing the permission problem you are facing). This can be done using the **semodule** command:

```
root # semodule --disable_dontaudit --build
```

The shorter version of this command is **semodule -DB**, btw. What happens here is that the SELinux management utility **semodule** rebuilds the SELinux policy, but ignores the *dontaudit* statements. The policy is then loaded in memory. Once you disable the *dontaudit* statements, effectively all denials are logged.

When you are tired of seeing all those denials, you can re-enable the *dontaudit* statements, by rebuilding the policy:

```
root # semodule --build
```

If you want to see all the *dontaudit* statements, run **sesearch --dontaudit**. You will notice that they follow the same structure as the *allow* statements we have seen earlier on.

```
root # sesearch --dontaudit
```

```
...
   dontaudit httpd_t user_tty_device_t : chr_file { ioctl read write getattr append o
pen } ;
   dontaudit mta_user_agent httpd_sys_script_t : fd use ;
```

Later, when we learn how to create our own policy modules, we will show you how you can add your own *dontaudit* statements.

Other ways to read denial information

The friendly developers that work with SELinux on a daily basis have made a few tools that help you identify SELinux-related issues.

ausearch

The **ausearch** utility is not an SELinux-specific utility. It is a Linux audit related utility, which parses the audit logs and allows you to query the entries in the logs. One of the advantages that it shows is that it already converts the time stamp into a human readable one.

```
root # ausearch -m avc --start recent
```

```
time->Thu Mar 14 21:15:57 2013
type=AVC msg=audit(1363292157.560:188): avc:  denied  { read } for  pid=29495 comm="T
race"
name="online" dev="sysfs" ino=30 scontext=staff_u:staff_r:googletalk_plugin_t
tcontext=system_u:object_r:sysfs_t tclass=file
```

The *recent* start gives the denials from the last 10 minutes. You can also use *today* for, well, today's denials.

sealert

The **sealert** command is not provided on an SELinux-enabled Gentoo system by default, but it is available on RedHat Enterprise Linux and related distributions. It integrates together with a specific daemon called **setroubleshootd**, which gives a translation of an AVC denial similar to the human translation given earlier in this tutorial. For instance, the following message can be displayed in the system logs:

```
setroubleshoot: SELinux is preventing httpd (httpd_t) "getattr" to /var/www/html/file
1
(samba_share_t). For complete SELinux messages.
run sealert -l 84e0b04d-d0ad-4347-8317-22e74f6cd020
```

The **sealert** tool then gives a more detailed explanation of the denial:

```
root # sealert -l 84e0b04d-d0ad-4347-8317-22e74f6cd020
```

```
Summary:

SELinux is preventing httpd (httpd_t) "getattr" to /var/www/html/file1
(samba_share_t).

Detailed Description:

SELinux denied access to /var/www/html/file1 requested by httpd.
/var/www/html/file1 has a context used for sharing by different program. If you
would like to share /var/www/html/file1 from httpd also, you need to change its
file context to public_content_t. If you did not intend to this access, this
could signal a intrusion attempt.

Allowing Access:

You can alter the file context by executing chcon -t public_content_t
'/var/www/html/file1'

Fix Command:

chcon -t public_content_t '/var/www/html/file1'

Additional Information:

Source Context                unconfined_u:system_r:httpd_t:s0
Target Context                unconfined_u:object_r:samba_share_t:s0
Target Objects                /var/www/html/file1 [ file ]
Source                        httpd
Source Path                   /usr/sbin/httpd
Port                          <Unknown>
Host                          hostname
Source RPM Packages           httpd-2.2.10-2
Target RPM Packages
Policy RPM                    selinux-policy-3.5.13-11.fc11
Selinux Enabled               True
Policy Type                   targeted
MLS Enabled                   True
Enforcing Mode                Enforcing
Plugin Name                   public_content
Host Name                     hostname
Platform                      Linux hostname 2.6.27.4-68.fc11.i686 #1 SMP Thu Oct
30 00:49:42 EDT 2008 i686 i686
Alert Count                   4
First Seen                    Wed Nov  5 18:53:05 2008
Last Seen                     Wed Nov  5 01:22:58 2008
Local ID                      84e0b04d-d0ad-4347-8317-22e74f6cd020
Line Numbers

Raw Audit Messages

node=hostname type=AVC msg=audit(1225812178.788:101): avc:  denied  { getattr } for
```

```
pid=2441 comm="httpd" path="/var/www/html/file1" dev=dm-0 ino=284916
scontext=unconfined_u:system_r:httpd_t:s0 tcontext=unconfined_u:object_r:samba_share_
t:s0 tclass=file

node=hostname type=SYSCALL msg=audit(1225812178.788:101): arch=40000003 syscall=196
success=no exit=-13 a0=b8e97188 a1=bf87aaac a2=54dff4 a3=2008171 items=0 ppid=2439
pid=2441 auid=502 uid=48 gid=48 euid=48 suid=48 fsuid=48 egid=48 sgid=48 fsgid=48
tty=(none) ses=3 comm="httpd" exe="/usr/sbin/httpd" subj=unconfined_u:system_r:httpd_
t:s0
key=(null)
```

What you need to remember

From this tutorial, you should remember that

- denials are logged in the `avc.log` (no audit daemon running) or `audit.log` (audit daemon running) log files
- denials might be obscured through *dontaudit* statements, which you can disable using **semodule -DB** and re-enable through **semodule -B**
- the denial logging gives you great detail about who (process information, including security context) is trying to do what (permission) against something (target information, including security context)

Retrieved from "https://wiki.gentoo.org/index.php?title=SELinux/Tutorials/Where_to_find_SELinux_permission_denial_details&oldid=1073942 (https://wiki.gentoo.org/index.php?title=SELinux/Tutorials/Where_to_find_SELinux_permission_denial_details&oldid=1073942)"