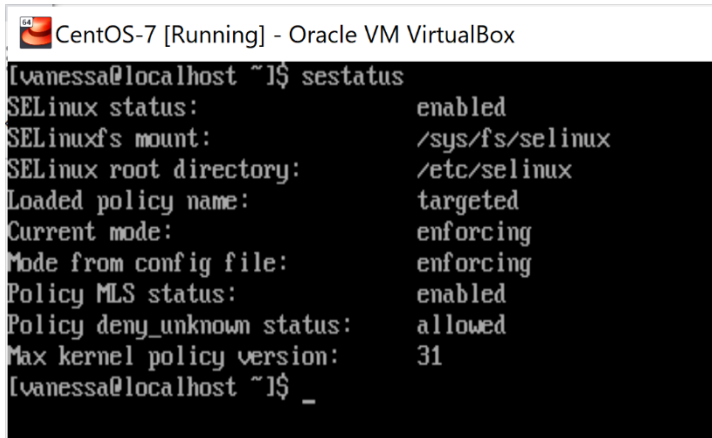


## Lab 3: SELinux Lab

### Initial Setup:

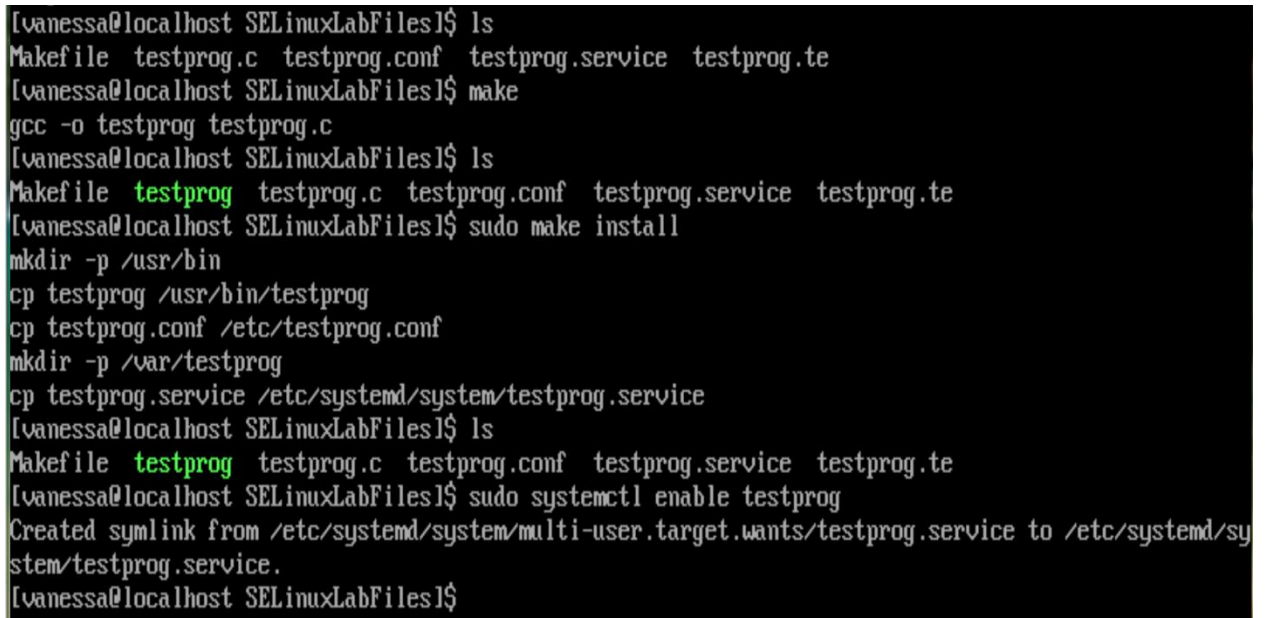
1. Downloaded VM with root and vanessa accounts.
2. Ran sestatus to see if my system is capable of running in SELinux enforcing mode.

A terminal window titled 'CentOS-7 [Running] - Oracle VM VirtualBox' showing the output of the 'sestatus' command. The output lists various SELinux configuration details.

```
CentOS-7 [Running] - Oracle VM VirtualBox
[vanessa@localhost ~]$ sestatus
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
[vanessa@localhost ~]$ _
```

Figure 1

Build the program.

A terminal window showing the steps to build and install a program named 'testprog'. The user runs 'ls', 'make', and 'sudo make install'. The output shows the creation of files and directories, and the installation of the binary and service files.

```
[vanessa@localhost SELinuxLabFiles]$ ls
Makefile testprog.c testprog.conf testprog.service testprog.te
[vanessa@localhost SELinuxLabFiles]$ make
gcc -o testprog testprog.c
[vanessa@localhost SELinuxLabFiles]$ ls
Makefile testprog testprog.c testprog.conf testprog.service testprog.te
[vanessa@localhost SELinuxLabFiles]$ sudo make install
mkdir -p /usr/bin
cp testprog /usr/bin/testprog
cp testprog.conf /etc/testprog.conf
mkdir -p /var/testprog
cp testprog.service /etc/systemd/system/testprog.service
[vanessa@localhost SELinuxLabFiles]$ ls
Makefile testprog testprog.c testprog.conf testprog.service testprog.te
[vanessa@localhost SELinuxLabFiles]$ sudo systemctl enable testprog
Created symlink from /etc/systemd/system/multi-user.target.wants/testprog.service to /etc/systemd/system/testprog.service.
[vanessa@localhost SELinuxLabFiles]$
```

Figure 2

### Task 1: Running testprog

Here, I ran the binary from the shell. Since the current mode in SELinux was set to enforcing, we would expect the application to fail however, it was working as seen on figure3. This is because

the default targeted policy is setup to run unknown applications in an *unconfined* context (seen on figure 4). In summary, these unknown applications run as if SELinux was completely disabled.

```
[vanessa@localhost SELinuxLabFiles]$ sudo /usr/bin/testprog /etc/testprog.conf /var/run/testprog.pid
&
[1] 8473
[vanessa@localhost SELinuxLabFiles]$ Using configuration file: /etc/testprog.conf
Wrote PID to /var/run/testprog.pid
Writing output to: /var/testprog/testprg.txt
Iteration count: -1
```

Figure 3

```
Ps -efZ | grep $(cat /var/run/testprog.pid)
```

unconfined\_u:unconfined\_r:unconfined\_t:s0 meaning user, role, type, and level fields respectively.

```
[vanessa@localhost SELinuxLabFiles]$ ps -efZ | grep $(cat /var/run/testprog.pid)
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 root 8478 8476 0 18:15 tty1 00:00:00 /usr/bin/testprog /etc/testprog.conf /var/run/testprog.pid
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 vanessa 8484 8169 0 18:17 tty1 00:00:00 grep --color=auto 8478
```

Figure 4

## Task 2: Targeted Policies

Ran the application using system as seen on figure5.

```
[vanessa@localhost SELinuxLabFiles]$
[vanessa@localhost SELinuxLabFiles]$ sudo systemctl start testprog
[vanessa@localhost SELinuxLabFiles]$ sudo systemctl status testprog
■ testprog.service - SELinux Test Program
   Loaded: loaded (/etc/systemd/system/testprog.service; enabled; vendor preset: disabled)
   Active: active (running) since Fri 2021-03-05 18:18:25 EST; 7s ago
 Main PID: 8495 (testprog)
   CGroup: /system.slice/testprog.service
           └─8495 /usr/bin/testprog /etc/testprog.conf /var/run/testprog.pid

Mar 05 18:18:25 localhost.localdomain systemd[1]: Started SELinux Test Program.
Mar 05 18:18:25 localhost.localdomain testprog[8495]: Using configuration file: /etc/testprog.conf
Mar 05 18:18:25 localhost.localdomain testprog[8495]: Wrote PID to /var/run/testprog.pid
Mar 05 18:18:25 localhost.localdomain testprog[8495]: Writing output to: /var/testprog/testprg.txt
Mar 05 18:18:25 localhost.localdomain testprog[8495]: Iteration count: -1
```

Figure 5

Here the application is still working however, the contexts are different this time. Instead of seeing “unconfined” keyword in user, role and type fields, we see it in type field only unlike when using the shell. Please refer to figure6.

```
Ps -efZ | grep $(cat /var/run/testprog.pid)
```

system\_u:system\_r:unconfined\_service\_t:s0

```
[vanessa@localhost SELinuxLabFiles]$ ps -efZ | grep $(cat /var/run/testprog.pid)
system_u:system_r:unconfined_service_t:s0 root 8495 1 0 18:18 ? 00:00:00 /usr/bin/testprog /
tc/testprog.conf /var/run/testprog.pid
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 vanessa 8500 8169 0 18:18 tty1 00:00:00 grep
--color=auto 8495
[vanessa@localhost SELinuxLabFiles]$ _
```

Figure 6

Notice the different context this time? At this stage we have not defined any SELinux policy for this application, so why is it behaving differently?

- ❖ The simple answer to this question is inheritance. The type context of the file inherited the context of its parent. In our testprog binary, although we had no SELinux configuration setup, the program still worked and we got some context back because /usr/bin/testprog inherited from /usr/bin/ its parent.

### Task 3: Targeting testprog

1. Now we know that we can run our binary in both shell and system although shell is running in unconfined space that has no restrictions. To make it more secure, I wrote SELinux policy using the files provided in SELinuxLabFiles. Lastly, I verified that the policy was loaded. The module now has a name and a version number to be referred to as seen on figure7.

```
[vanessa@localhost SELinuxLabFiles]$ make -f /usr/share/selinux/devel/Makefile testprog.pp
Compiling targeted testprog module
/usr/bin/checkmodule: loading policy configuration from tmp/testprog.tmp
/usr/bin/checkmodule: policy configuration loaded
/usr/bin/checkmodule: writing binary representation (version 19) to tmp/testprog.mod
Creating targeted testprog.pp policy package
rm tmp/testprog.mod tmp/testprog.mod.fc
[vanessa@localhost SELinuxLabFiles]$
[vanessa@localhost SELinuxLabFiles]$
[vanessa@localhost SELinuxLabFiles]$ sudo semodule -i testprog.pp
[vanessa@localhost SELinuxLabFiles]$
[vanessa@localhost SELinuxLabFiles]$ sudo semodule -l | grep testprog
testprog 0.1
[vanessa@localhost SELinuxLabFiles]$ _
```

Figure 7

2. Now that you see the policy module is loaded, once again try running testprog from the shell and see what happens. Is it working properly? Is the binary running in the correct context now?
  - ❖ Even after creating the SELinux policy, the application was still working when I ran it via shell. Simply because it was still running in unconfined environment as seen in figure8.
  - ❖ The binary is not working in the correct context. To fix it, I needed to change the context of the file itself. This will be done in part4.

```

[vanessa@localhost SELinuxLabFiles]$ sudo /usr/bin/testprog /etc/testprog.conf /var/run/testprog.pid
&
[1] 16900
[vanessa@localhost SELinuxLabFiles]$ Using configuration file: /etc/testprog.conf
Wrote PID to /var/run/testprog.pid
Writing output to: /var/testprog/testprg.txt
Iteration count: -1
^C
[vanessa@localhost SELinuxLabFiles]$ ps -efZ | grep $(cat /var/run/testprog.pid)
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 root 16902 16900 0 18:37 tty1 00:00:00 /usr/b
in/testprog /etc/testprog.conf /var/run/testprog.pid
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 vanessa 16905 8169 0 18:40 tty1 00:00:00 grep
--color=auto 16902
[vanessa@localhost SELinuxLabFiles]$

```

Figure 8

#### Task 4: Filesystem Contexts

1. Based on what we saw in Task 3, we may need to change the context on files themselves. One way to do this would be to use `chcon` on our binary. Will this work for our purposes? What might be a side effect of using this command?

❖ Using `chcon` works until when the system is relaunched. Thus, I had to make the changes as part of the policies I created in part3.

```

[vanessa@localhost SELinuxLabFiles]$ sudo chcon -t testprog_exec_t /usr/bin/testprog
[vanessa@localhost SELinuxLabFiles]$ ls -lZ /usr/bin/testprog
-rwxr-xr-x. root root unconfined_u:object_r:testprog_exec_t:s0 /usr/bin/testprog
[vanessa@localhost SELinuxLabFiles]$ ls -ldZ /usr/bin
dr-xr-xr-x. root root system_u:object_r:bin_t:s0 /usr/bin
[vanessa@localhost SELinuxLabFiles]$ ls -lZ /usr/bin/testprog
-rwxr-xr-x. root root unconfined_u:object_r:testprog_exec_t:s0 /usr/bin/testprog
[vanessa@localhost SELinuxLabFiles]$ sudo restorecon -v /usr/bin/*
restorecon reset /usr/bin/testprog context unconfined_u:object_r:testprog_exec_t:s0->unconfined_u:ob
ject_r:bin_t:s0
[vanessa@localhost SELinuxLabFiles]$

```

Figure 9

2. Changing the context in the policy itself. I reused `testprog.te` and introduced a new file context (`.fc`) file.

```

[vanessa@localhost SELinuxLabFiles]$ ls
Makefile testprog.c testprog.fc testprog.pp testprog.te
testprog testprog.conf testprog.if testprog.service tmp
[vanessa@localhost SELinuxLabFiles]$ cat testprog.fc
/usr/bin/testprog -- system_u:object_r:testprog_exec_t:s0
[vanessa@localhost SELinuxLabFiles]$ _

```

Figure 10

Since I made the changes, I rebuild the policies to have a full effect as seen on figure11.

❖ On figure12, I got a segmentation fault when I tried to run the application in shell. This indicates that the contexts were switched and SELinux in shell has full restrictions.

```

[vanessa@localhost SELinuxLabFiles]$ make -f /usr/share/selinux/devel/Makefile testprog.pp
Compiling targeted testprog module
/usr/bin/checkmodule: loading policy configuration from tmp/testprog.tmp
/usr/bin/checkmodule: policy configuration loaded
/usr/bin/checkmodule: writing binary representation (version 19) to tmp/testprog.mod
Creating targeted testprog.pp policy package
rm tmp/testprog.mod tmp/testprog.mod.fc
[vanessa@localhost SELinuxLabFiles]$ sudo semodule -r testprog
libsemanage.semanage_direct_remove_key: Removing last testprog module (no other testprog module exists at another priority).
[vanessa@localhost SELinuxLabFiles]$ sudo semodule -i testprog.pp
[vanessa@localhost SELinuxLabFiles]$
[vanessa@localhost SELinuxLabFiles]$
[vanessa@localhost SELinuxLabFiles]$ sudo restorecon -v /usr/bin/*
restorecon reset /usr/bin/testprog context unconfined_u:object_r:bin_t:s0->unconfined_u:object_r:testprog_exec_t:s0
[vanessa@localhost SELinuxLabFiles]$ _

```

Figure 11

Testing by running binary from the shell

```

[vanessa@localhost SELinuxLabFiles]$ sudo /usr/bin/testprog /etc/testprog.conf /var/run/testprog.pid
Segmentation fault
[vanessa@localhost SELinuxLabFiles]$

```

Figure 12

## Task 5: sealert and audit.log

1. Audit.log that saves logs of SELinux when it is enabled.

Ran `sudo grep testprog /var/log/audit/audit.log` to view my failed runs of testprog on the system.

```

path="/dev/tty1" dev="devtmpfs" ino=5385 scontext=unconfined_u:unconfined_r:testprog_t:s0-s0:c0.c1023 tcontext=unconfined_u:object_r:user_tty_device_t:s0 tclass=chr_file permissive=0
type=SYSCALL msg=audit(1614988737.652:361): arch=c000003e syscall=59 success=yes exit=0 a0=5568cfbd2c68 a1=5568cfbccc88 a2=5568cfbe09f0 a3=0 items=0 ppid=16996 pid=16998 auid=1000 uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=(none) ses=1 comm="testprog" exe="/usr/bin/testprog" subj=unconfined_u:unconfined_r:testprog_t:s0-s0:c0.c1023 key=(null)
type=AVC msg=audit(1614988737.653:362): avc: denied { create } for pid=16998 comm="testprog" scontext=unconfined_u:unconfined_r:testprog_t:s0-s0:c0.c1023 tcontext=unconfined_u:unconfined_r:testprog_t:s0-s0:c0.c1023 tclass=unix_dgram_socket permissive=0
type=SYSCALL msg=audit(1614988737.653:362): arch=c000003e syscall=41 success=no exit=-13 a0=1 a1=800002 a2=0 a3=b items=0 ppid=16996 pid=16998 auid=1000 uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=(none) ses=1 comm="testprog" exe="/usr/bin/testprog" subj=unconfined_u:unconfined_r:testprog_t:s0-s0:c0.c1023 key=(null)

```

Figure 13

The above commnd had a lot of information thus, I used grep to filter lines with testprog as seen below.

Ran `sudo grep AVC /var/log/audit/audit.log | grep testprog`

```

type=AVC msg=audit(1614988737.653:364): avc: denied { write } for pid=16998 comm="testprog" name="console" dev="devtmpfs" ino=5379 scontext=unconfined_u:unconfined_r:testprog_t:s0-s0:c0.c1023 tcontext=system_u:object_r:console_device_t:s0 tclass=chr_file permissive=0
type=AVC msg=audit(1614988737.653:365): avc: denied { write } for pid=16998 comm="testprog" name="testprog.pid" dev="tmpfs" ino=36314 scontext=unconfined_u:unconfined_r:testprog_t:s0-s0:c0.c1023 tcontext=unconfined_u:object_r:var_run_t:s0 tclass=file permissive=0
[vanessa@localhost SELinuxLabFiles]$

```

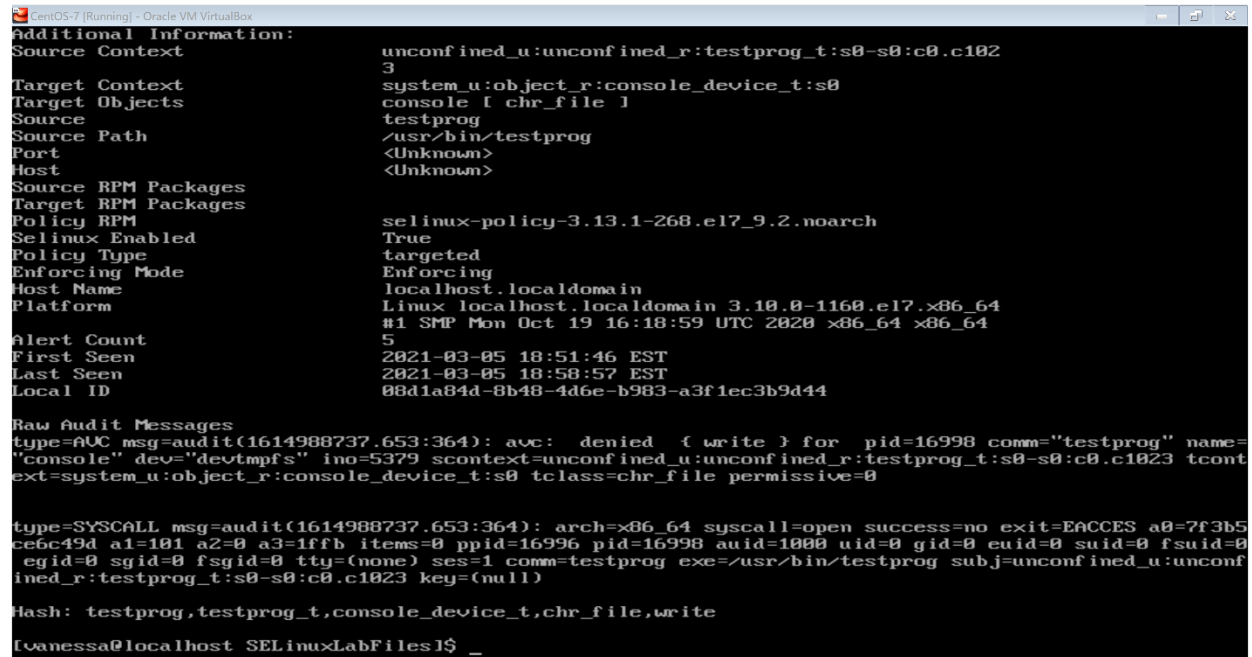
Figure 14

Figure14 shows a process with PID as 16998 named testprog.pid that tried to write a file object and its context was unconfined\_u:object\_r:var\_run\_t:s0-s0:c0.c1823 and its source context was

unconfined\_u:unconfined\_r:testprog\_t:s0. All in all, this line of code verifies that the testprog\_t created context.

## 2. Sealert

cat sealert.txt



```
Additional Information:
Source Context          unconfined_u:unconfined_r:testprog_t:s0-s0:c0.c102
                        3
Target Context          system_u:object_r:console_device_t:s0
Target Objects          console [ chr_file ]
Source                  testprog
Source Path              /usr/bin/testprog
Port                     <Unknown>
Host                     <Unknown>
Source RPM Packages
Target RPM Packages
Policy RPM               selinux-policy-3.13.1-268.el7_9.2.noarch
Selinux Enabled          True
Policy Type              targeted
Enforcing Mode           Enforcing
Host Name                localhost.localdomain
Platform                 Linux localhost.localdomain 3.10.0-1160.el7.x86_64
                        #1 SMP Mon Oct 19 16:18:59 UTC 2020 x86_64 x86_64
Alert Count              5
First Seen               2021-03-05 18:51:46 EST
Last Seen                2021-03-05 18:58:57 EST
Local ID                 08d1a84d-8b48-4d6e-b983-a3f1ec3b9d44

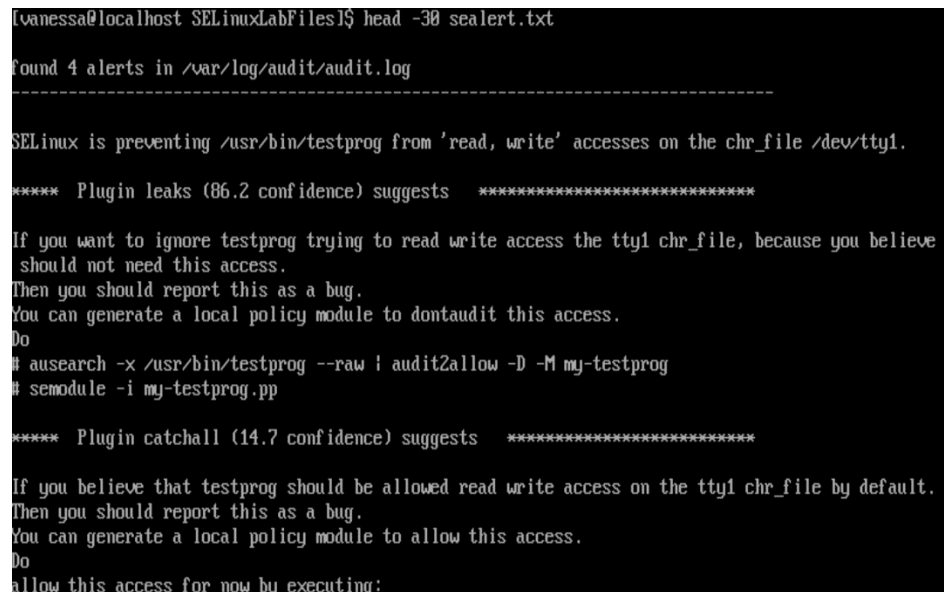
Raw Audit Messages
type=AVC msg=audit(1614988737.653:364): avc: denied { write } for pid=16998 comm="testprog" name=
"console" dev="devtmpfs" ino=5379 scontext=unconfined_u:unconfined_r:testprog_t:s0-s0:c0.c1023 tcont
ext=system_u:object_r:console_device_t:s0 tclass=chr_file permissive=0

type=SYSCALL msg=audit(1614988737.653:364): arch=x86_64 syscall=open success=no exit=EACCES a0=7f3b5
ce6c49d a1=101 a2=0 a3=1fffb items=0 ppid=16996 pid=16998 auid=1000 uid=0 gid=0 euid=0 suid=0 fsuid=0
egid=0 sgid=0 fsgid=0 tty=(none) ses=1 comm=testprog exe=/usr/bin/testprog subj=unconfined_u:unconf
ined_r:testprog_t:s0-s0:c0.c1023 key=(null)

Hash: testprog,testprog_t,console_device_t,chr_file,write
[vanessa@localhost SELinuxLabFiles]$ _
```

Figure 15

The first 30 lines of the sealert.txt file



```
[vanessa@localhost SELinuxLabFiles]$ head -30 sealert.txt

found 4 alerts in /var/log/audit/audit.log
-----

SELinux is preventing /usr/bin/testprog from 'read, write' accesses on the chr_file /dev/tty1.

***** Plugin leaks (86.2 confidence) suggests *****

If you want to ignore testprog trying to read write access the tty1 chr_file, because you believe it
should not need this access.
Then you should report this as a bug.
You can generate a local policy module to dontaudit this access.
Do
# ausearch -x /usr/bin/testprog --raw | audit2allow -D -M my-testprog
# semodule -i my-testprog.pp

***** Plugin catchall (14.7 confidence) suggests *****

If you believe that testprog should be allowed read write access on the tty1 chr_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:
```

Figure 16



Try running the first command it suggests and inspect the results. What is the automatic policy saying? What are the pros and cons of using this generated policy versus writing our own like in the previous task?

```
[vanessa@localhost SELinuxLabFiles]$ sudo ausearch -x /usr/bin/testprog --raw | audit2allow -D -M my-testprog
[sudo] password for vanessa:
***** IMPORTANT *****
To make this policy package active, execute:

semodule -i my-testprog.pp

[vanessa@localhost SELinuxLabFiles]$ semodule -i my-testprog.pp
```

Cat my-testprog.te

```
-testprog
***** IMPORTANT *****
To make this policy package active, execute:

semodule -i my-testprog.pp

[vanessa@localhost SELinuxLabFiles]$ cat my-testprog.te

module my-testprog 1.0;

require {
    type var_run_t;
    type user_tty_device_t;
    type testprog_t;
    type console_device_t;
    class unix_dgram_socket create;
    class chr_file { append read write };
    class file write;
}

#===== testprog_t =====

#!!!! This avc has a dontaudit rule in the current policy
dontaudit testprog_t console_device_t:chr_file write;

#!!!! This avc has a dontaudit rule in the current policy
dontaudit testprog_t self:unix_dgram_socket create;

#!!!! The file '/dev/tty1' is mislabeled on your system.
#!!!! Fix with $ restorecon -R -v /dev/tty1
#!!!! This avc has a dontaudit rule in the current policy
dontaudit testprog_t user_tty_device_t:chr_file { append read write };

#!!!! WARNING: 'var_run_t' is a base type.
#!!!! This avc has a dontaudit rule in the current policy
dontaudit testprog_t var_run_t:file write;
[vanessa@localhost SELinuxLabFiles]$ _
```

The automatic policy is saying not to audit testprog\_t file.

Pros of using generated policy is that they are better than not using SELinux or using unconfined state. However, these generated policy are not the most secured way of ensuring the highest level of security. Thus, it is better to write our own policies.