Lesson #7
Increasing system security with SELinux

Advanced Linux Administration

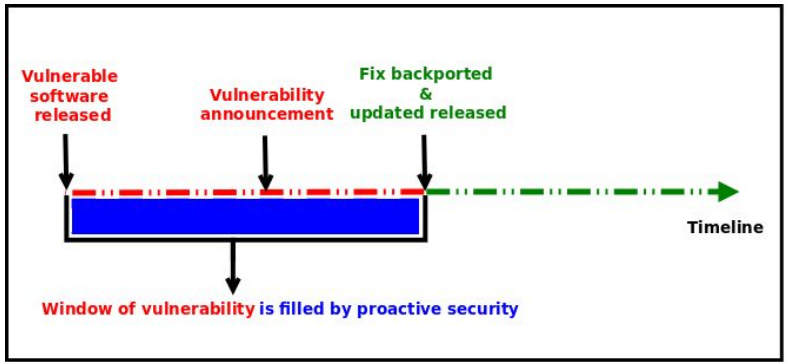
Karel Šrot, Eduard Beneš

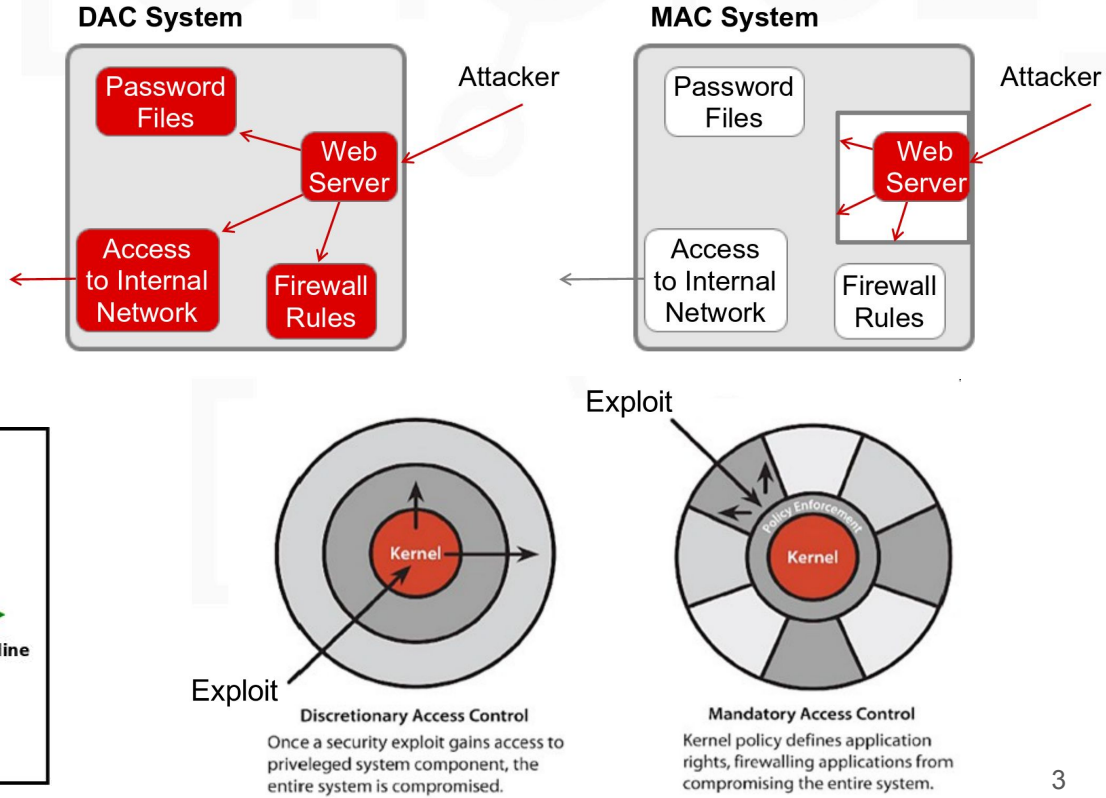
Introduction

30 min

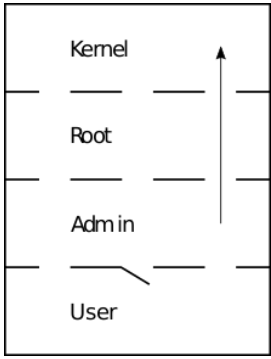
Motivation

- Compromised Service or Kernel
- Window of vulnerability
  - Reactive VS proactive approach
- Few exploits SELinux can prevent
  - Venom
  - Docker CVE-2016-9962
  - Shellshock





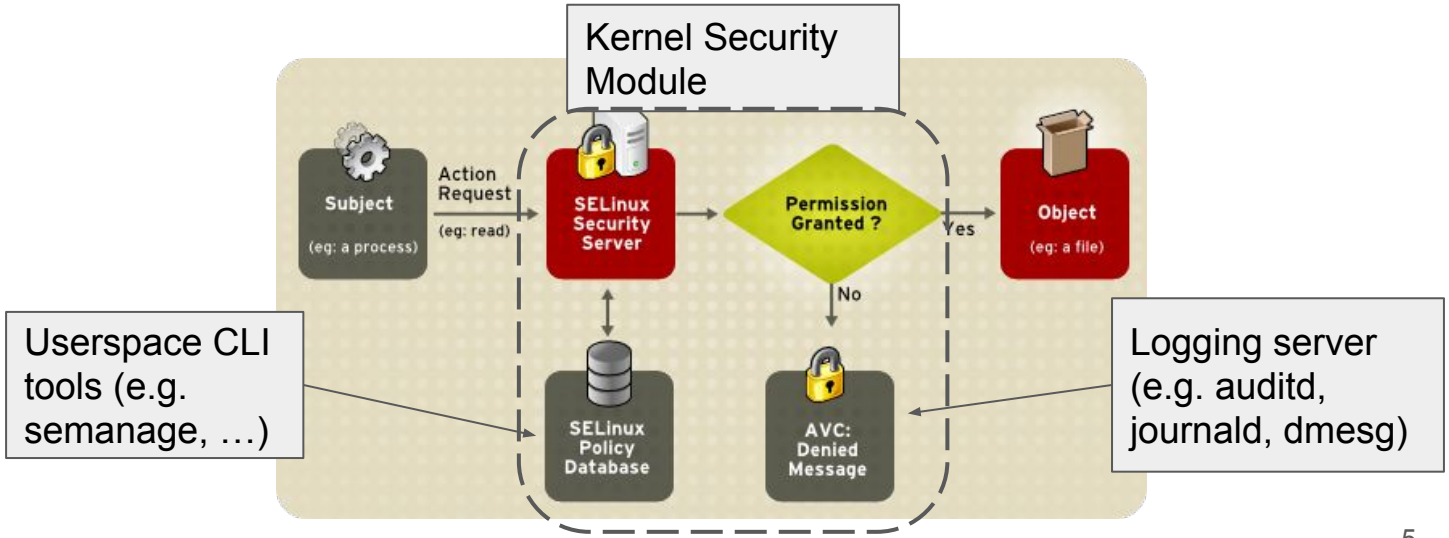
What is SELinux and how it can help?



- SELinux brings proactive security to your system with Mandatory Access Control (MAC).
- Provides isolation by separation of processes and resources on your system.
  - With process isolation it can mitigate attacks via privilege escalation.
  - All you need to know about SELinux is in this coloring book :)
- It is enforced by a Kernel!
- All processes and files have a SELinux context (labels)
- SELinux policy is enforced system-wide
- Other known implementations of MAC are:
  - AppArmor
  - Smack

SELinux Architecture Overview

- Kernel security module and user-space tools.
- Security policy rules are loaded into policy database in kernel from user-space.
- Kernel enforces the security policy on the system between Subjects and Objects
- When access is denied it is called **AVC denial** and message is generated.



SELinux Decision Making in Detail

SELinux Modes
- Enforcing (deny by default)
- Permissive (evaluate and allow)

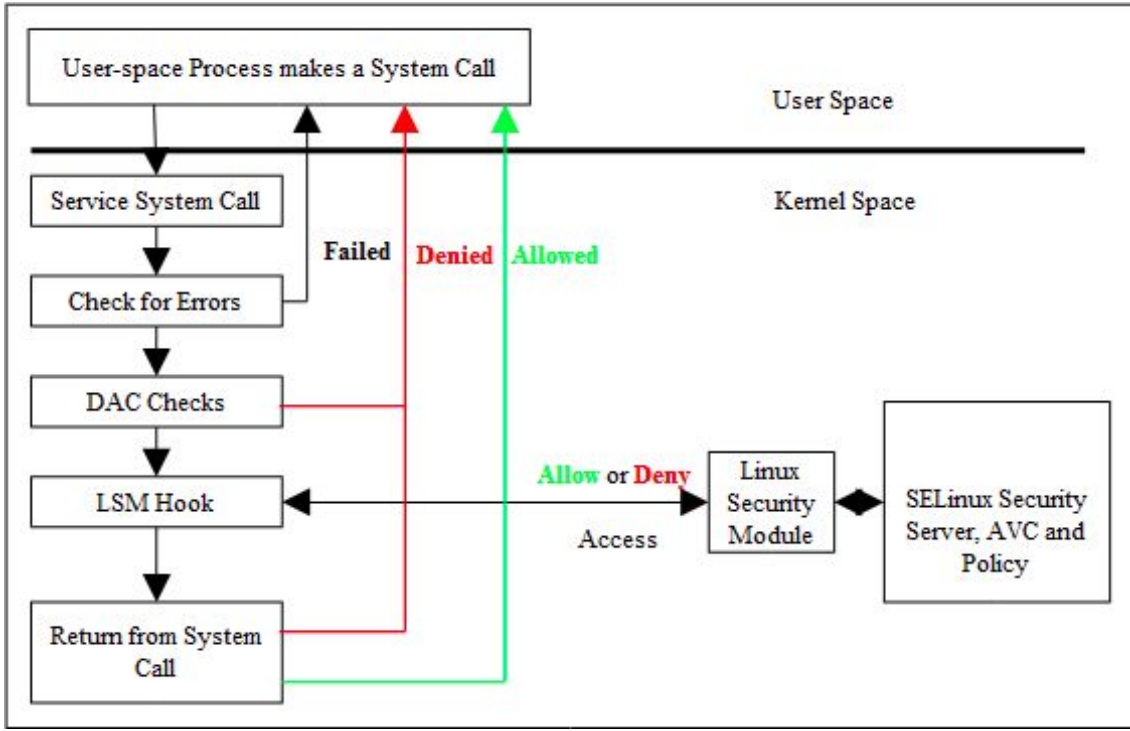
Default mode is set in /etc/selinux/config

DAC checks
- User-Group-Other
- Read-Write-Execute

```
$ ls -la
$ man chown
```

ACL
- Posix access control lists
- Allow fine-grained DAC rights on files and directories

```
$ getfacl /etc/passwd
$ man 5 acl
```



SELinux Labels (a.k.a. SELinux Context)

- SELinux context for all files (everything is a file) and processes is represented by a SELinux label
- Label consists of four fields:

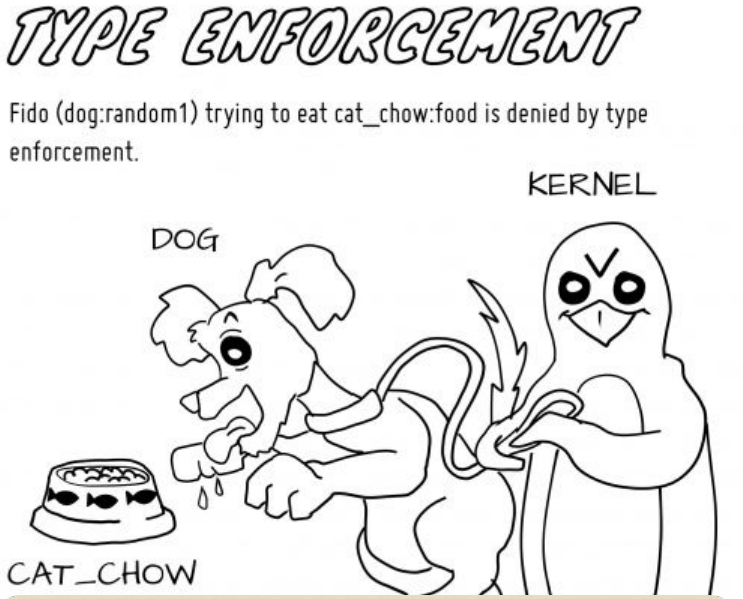
  <user>:<role>:<type>:<MLS/MCS>

- All of these information is used to make access control decisions by the Linux kernel
- Let’s look at the labels on a system for different resources (file, process, port, …)

```
$ ls -Z /etc/passwd
system_u:object_r:passwd_file_t:s0 /etc/passwd

# pstree -Z | head
# ls -Z /
# id -Z
# netstat -Z ; ss -Z
```

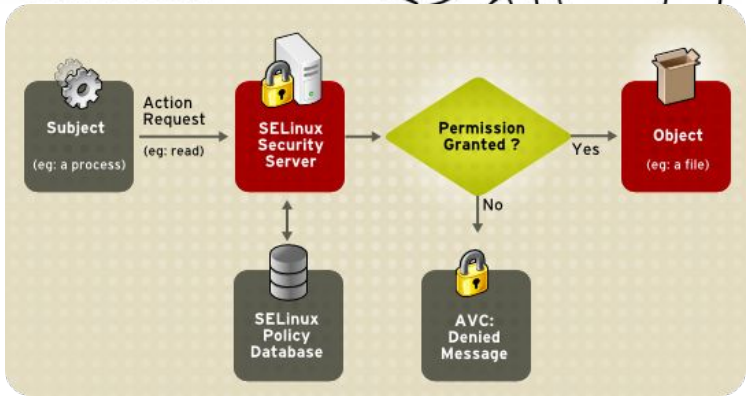
Type Enforcement (TE)



- Type Enforcement policy defines type of an subject and which action it can perform against object
- It uses just types (3rd field of a SELinux label) to define policy rules
- So called “unconfined” processes run with a types (e.g. `unconfined_t`) that are allowed almost all access, therefore they are not protected by SELinux when they get compromised.

`<user>:<role>:`**`<type>`**`:<MLS/MCS>`

```
# sesearch -s passwd_t -t etc_t -p write -A -ds -dt
```



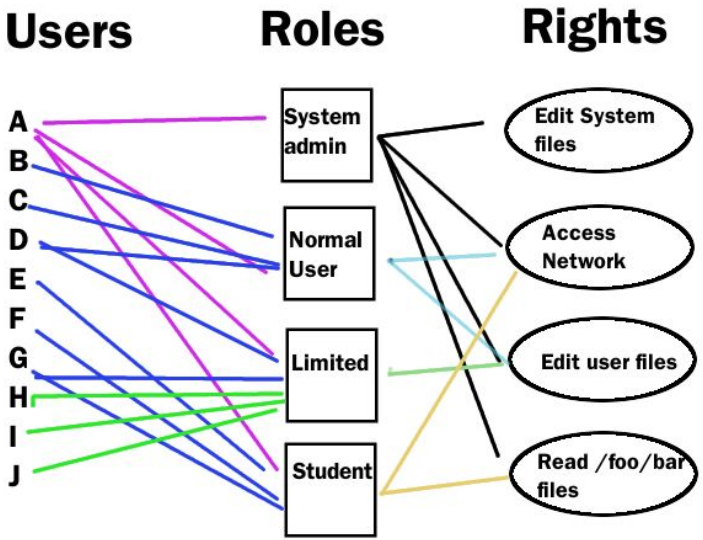
Role Based Access Control (RBAC)

- SELinux supports RBAC to implement refinement of security policy based on users and roles.
- The SELinux user definitions declare what the supported roles are that a user can "go" to
- Roles dictate which domains that the user (or session) is allowed to go in

<user>:<role>:<type>:<MLS/MCS>

```
# semanage user -l

# semanage login -l
# seinfo --role --user    (... -x)
# id -Z
# useradd -Z ...
# newrole -r sysadm_r
```



SELinux Policy Rules

- Rules are describing interactions between Subjects (process) and Objects (file, process, socket, …)
- Rules can be also other than `allow (dontaudit, neverallow)`
- For example, we might have a rule like this to allow `httpd_process` read its log files
  - `allow httpd_process httpd_log: FILE READ;`
- You can easily generate custom policy rules using `audit2allow` from logged AVC denials
- General pattern of rules used to define SELinux policy loaded into kernel:

```
rule subject-type object-type : object-class { permissions }
```

- Real example following the rule pattern above:

```
# sesearch -s passwd_t -t etc_t -p write --allow -ds -dt
allow passwd_t etc_t:file { append getattr ioctl read write };
```

SELinux Booleans

- Booleans enable runtime customization of SELinux policy to adjust it for a desired system configuration. Changes can be temporary or persistent.
- Conditional rules can be easily enabled or disabled by admin. For example to allow `httpd` daemon to serve web pages from user home directories.
- Checking status of available booleans?

```
$ getsebool -a

$ semanage boolean -l

$ man booleans
# dnf install selinux-policy-doc
$ apropos _selinux
# setsebool -P httpd_enable_homedir
```

Audit Daemon

- Auditd is an userspace daemon part of Linux Auditing System
  - Make sure it is running `# systemctl status auditd.service`
- By default all AVC messages end up in `/var/log/audit/audit.log` managed by Auditd
- In case `auditd` is not running, AVC will go to systemd journal (or `/var/log/messages`), or kernel ring buffer (`dmesg`)
  - with `systemd-journald-audit.socket` enabled AVCs always end up in journal
  - auditd logs should be queried using `ausearch` tool
- Tools like `sealert, audit2allow, audit2why` can help you diagnose SELinux denials
- Audit record types: AVC, USER_AVC, SELINUX_ERR, USER_AUTH, ...

```
# aureport
# man ausearch

# passwd --help >& /root/output.txt
# ausearch -m avc -ts recent
# systemctl kill auditd.service
# journalctl -S-300
```

AVC Messages and Searching SELinux Rules

- Access Vector Cache (AVC) messages are generated whenever access to an operation is granted or denied by kernel
  - By default only AVC denials are logged
  - To log allowed or hide denied operation `auditallow` and `dontaudit` rules can be used
- You can use `sesearch` from `setools-console` package to query loaded policy

```
# passwd --help >& /root/output.txt
# ausearch -c "passwd" ; ausearch -m avc -ts recent -sv no
----
time->Wed Oct 16 12:40:35 2019
type=AVC msg=audit(1571222435.365:8278): avc:  denied  { write } for  pid=8924
comm="passwd" path="/root/output.txt" dev="dm-1" ino=528771
scontext=unconfined_u:unconfined_r:passwd_t:s0-s0:c0.c1023
tcontext=unconfined_u:object_r:admin_home_t:s0 tclass=file permissive=0

# sesearch -p write -s passwd_t -t admin_home_t --allow
```

Checking SELinux Status on a System

- RTFM .. man pages
- Is SELinux in Enforcing or Permissive mode? Does it use targeted policy or some other?

  ```
  # getenforce
  # sestatus
  ```
- Switching to enforcing mode

  ```
  # setenforce 1
  ```
- In case of a mislabeled filesystem you can force to relabel during a next boot.

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

  .. NOTE: this might take a looong time
- What is a default SELinux policy state?

  ```
  # cat /etc/selinux/config
  ```
- I need more information

  ```
  # seinfo
  # avcstat
  ```

Disabling SELinux

- No longer possible through `/etc/selinux/config`
- Update kernel parameters with `selinux=0`

```
# grubby --update-kernel
ALL --args selinux=0
```

- To enabled SELinux again

```
# grubby --update-kernel
ALL --remove-args selinux
```

Managing File SELinux Context

- SELinux policy defines context for newly created objects depending on its type (`-m`) and path
- Context is by default 1) inherited or 2) policy default is used when creating new objects

```
$ matchpathcon -m type /some/path/to/dir/or/file
```

- **Temporary** changes with `chcon` will not survive reboot or restoring to default context

```
# mkdir /web/
# chcon -R -t httpd_sys_content_t /web/
# restorecon -R -v /web/
```

- **Persistent** changes to the policy will survive also reboot. Using `` `semanage fcontext' `` changes will be saved to `/etc/selinux/targeted/contexts/files/` directory, (see `man pages`)

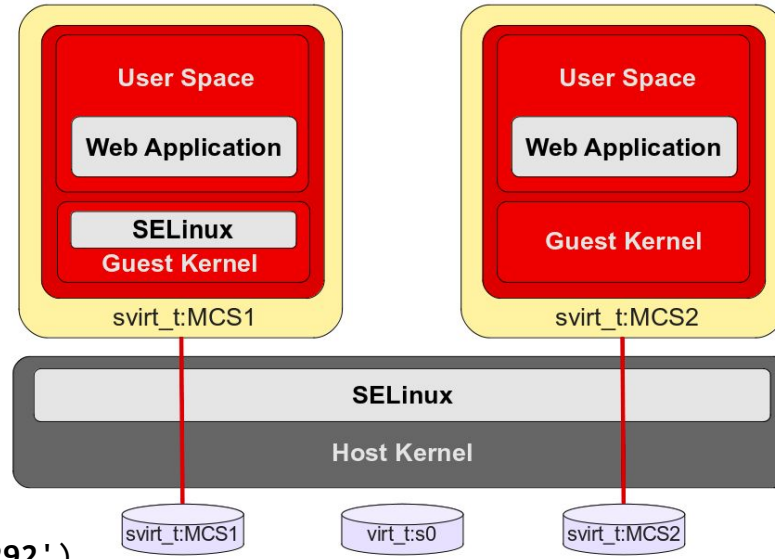
```
# semanage fcontext -a -t httpd_sys_content_t "/web(/.*)?"
# restorecon -R -v /web
```

- **Restore** file context to its default according to current SELinux policy:

```
# restorecon -v some-file        … (use -n for passive check without applying any changes)
# restorecon -R -v directory-name
# man fixfiles
```

Usage of MCS for Virtual Guests and Containers

- Type enforcement on host system using just dedicated types `container_t, container_runtime_t, container_file_t` is not sufficient to isolate running container processes
- Unique MCS category is used to confine containers and protect them from each other. Kernel enforces that MCS must match after Type is enforced
- Each container process is assigned an unique MCS




```
# pstree -Zp | grep container_t
  |   |-bash(24031,`system_u:system_r:container_t:s0:c137,c292')
  |   |-bash(2597,`system_u:system_r:container_t:s0:c563,c1010')
  |   |-bash(2681,`system_u:system_r:container_t:s0:c119,c927')

# ls -lZ `podman inspect -l --format "{{.HostnamePath}}"`
-rw-r--r--. 1 ... system_u:object_r:container_file_t:s0:c119,c927 12 Oct 15 09:47
/run/user/1000/overlay-containers/64c02f69cac16bc360b06a4ff8d2181a57b5a273a89412bae0533ac1e
dbf3c6a/userdata/hostname
```

Overview of SELinux policy management tools

- Use man pages for detailed description, options, and usage examples
- File contexts
  - `# semanage fcontext`
- Booleans
  - `# semanage boolean`
- Ports
  - `# semanage port`
- Permissive domains
  - `# semanage permissive`
- Searching policy and logs
  - `# sesearch`
  - `# ausearch`
- Creating local policy modules by parsing policy logs
  - `# audit2allow`
- Building and managing policy modules
  - `# semodule`

Workshop

60 min

Workshop labs

- In the following next slides there are 7 labs total
  - 6 regular + 1 bonus lab
- Each lab has a time estimate how much time should you spend on it if everything goes well
- We encourage you to help each other or raise your hand to get help from lecturers
- If you couldn’t finish all labs during this class, you should complete them on your own later because learned skills will be used in following lectures or during a final practical exam.
- *HINT: Focus on the lab content and leave any exploration or deep dive desires as a self-study for later*
- *HINT2: Use or create a fresh snapshot of your virtual guest machine to have and preserve a clean work environment without breaking your system during the labs.*

Overview of SELinux Labs

1. Lab1 - SELinux status and AVC logs (5min)
2. Lab2 - Basic policy management (5 mins)
3. Lab3 - Searching audit logs (5min)
4. Lab4 - Changing file context temporarily (5min)
5. Lab5 - Permanent policy changes (10m)
6. Lab6 - Simple SELinux booleans (10min)
7. Bonus Lab - Allow httpd access to homedirs (15min)

Lab1 - SELinux status and AVC logs ½ (5min)

- Check status of SELinux on your system

`# getenforce ; sestatus` → check it is in Enforcing mode using Targeted policy

`# cat /etc/selinux/config`

- Install packages required for all labs

```
# dnf install selinux-policy-doc selinux-policy-devel policycoreutils-python
policycoreutils-python-utils setools-console
```

Still you may have to install some additional packages later.

Lab1 - SELinux status and AVC logs 2/2 (5min)

- Finding logged SELinux AVC denials with(&out) auditd service running

`# systemctl status auditd.service` → make sure it is active and enabled

`# passwd --help >& /root/output.txt` → this will generate AVC denial

`# cat /root/output.txt ;` → this should be empty

`# ausearch -m avc -sv no -ts recent` → there should be AVC denial message for the passwd

- Now kill the auditd service and do the same action with the passwd

`# systemctl kill auditd.service` → make sure it is not running with systemctl status

```
# passwd --help >& /root/output.txt
# cat /root/output.txt ; ausearch -m avc -sv no -ts recent
```

- All SELinux logs/denials should now end up in systemd journal

`# journalctl -S-60` → … please raise your hand if not! :)

Lab2 - Basic policy management ½ (5 mins)

- Set the SELinux to `Permissive` mode using the Targeted policy

`# setenforce permissive` → check it is Permissive mode with commands from previous lab

- We want our logs to go to audit.log so better reboot

`# systemctl status auditd.service` → make sure it is active and enabled

- Save system time for checking logs and execute previously denied `passwd` action again.

```
# d1=`date +%T`
# passwd --help >& /root/output.txt
# cat /root/output.txt
# ausearch -m avc -sv no -ts $d1
```
→ make sure you've defined the `d1` variable

- Now, compare logged audit records to the output from Lab1 when in Enforcing mode
- There should be a new record for { ioctl } together with the previous { write }, and you should see `permissive=1` at the end of the audit record compared to `permissive=0` before (look at the outputs from lab1)

Lab2 - Basic policy management 2/2 (5 mins)

- The `passwd` action in Permissive mode should be allowed, all generated AVC denials recorded in logs and it should print the help message. Please let us know if something went wrong! :)
- Now set the system back to Enforcing mode and make sure auditd.service is running

`# systemctl status auditd.service` → start the service if it is not running properly

```
# setenforce 1
# getenforce
```

Lab3 - Searching audit logs ½ (5min)

Using `ausearch` tool we'll be working with previously generated events in `/var/log/audit/audit.log` and practicing various ways to search the logs.

```
# man ausearch
```

- Search the audit logs by name of executable using `-c` option and try `--pid` to specify process

`# ausearch -c passwd` → there should be all the denials you've generated in lab1,2

… now try to create a search using the --pid option

- Next you can specify a time period for the search using `-ts time-start -te time-end`, for example:

`# ausearch -ts 13:30 -te 13:55` → … please adjust the time values to for your class hours ;-)

- Or search by a file name:

```
# ausearch --file /root/output.txt
```

- Search based on success value. SELinux AVC denials have success value **no**

```
# ausearch -m avc -sv no
```

Lab3 - Searching audit logs 2/2 (5min)

- Audit records can have various types, .e.g. AVC, USER_AUTH, USER_LOGIN, … let's try them! :)

```
# ausearch -m user_auth -sv no
```

- Sometimes you might need to search audit logs related to a specific SELinux context

```
# ausearch -se container_t
```

- Now, authenticate successfully using `$ sudo -i' as a normal user and then as a user root list the corresponding success audit record:

```
user$ sudo -K ; sudo -i
root# ausearch -m user_auth -sv yes -ts recent -x /usr/bin/sudo
```

```
----
time->Fri Oct 18 09:31:05 2019
type=USER_AUTH msg=audit(1571383865.850:8689): pid=4882 uid=1000 auid=1000 ses=3
subj=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 msg='op=PAM:authentication grantors=pam_unix acct="ebenes"
exe="/usr/bin/sudo" hostname=? addr=? terminal=/dev/pts/9 res=success'
```

Lab4 - Changing file context temporarily 1/2 (5min)

In this lab we'll adjust context of `/root/output.txt` to allow access to passwd.

`# systemctl status auditd.service ; getenforce` → make sure we are good to go

- Compare context of these two files and set context of the `output.txt` to match `/etc/passwd`

```
# ls -lZ /etc/passwd /root/output.txt
# chcon -v --reference=/etc/passwd /root/output.txt
# restorecon -v /root/output.txt → to set it back to default context
# chcon -v -t passwd_file_t /root/output.txt → try to change it again
# ls -lZ /etc/passwd /root/output.txt → they should have the same context
```

- Now lets try to run passwd example again and observe it to succeed! :)

```
# d1=`date +%T` ; passwd --help >& /root/output.txt
```

- There should be no AVC denials related to the action before.

```
# cat /root/output.txt
# ausearch -m avc -sv no -ts $d1
```

Lab4 - Changing file context temporarily 2/2 (5min)

- Finally put the system into proper state. First our file and then recursively on a directory.

```
# restorecon -v /root/output.txt
Relabeled /root/output.txt from system_u:object_r:passwd_file_t:s0 to system_u:object_r:admin_home_t:s0
# restorecon -R -v /root
# rm /root/output.txt
```

Lab5 - Permanent policy changes 1/2 (10m)

In this lab we'll change context of `/root/output.txt` but permanently so it will survive reboot and relabelling of a system. Similarly you could change a label of port, boolean, …

```
# man semanage-fcontext
```

- Change system policy to label `/root/output.txt` as `passwd_file_t` and let it to fix the label.

```
# semanage fcontext -a -t passwd_file_t /root/output.txt
# restorecon -v /root/output.txt → you should be notified about the relabel
# ls -lZ /etc/passwd /root/output.txt → now, they should have the same context
```

- Now let's try to run `passwd` example again and observe it to succeed! :)

```
# date ; passwd --help >& /root/output.txt
```

- There should be no AVC denials logged as related to the `passwd` after time returned by `date`

```
# cat /root/output.txt
# ausearch -m avc -sv no -ts recent
```

- ~~At home you can try relabel the whole system during reboot … take a break :)~~

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

Lab5 - Permanent policy changes 2/2 (10m)

- Check that the SELinux type of `output.txt` still matches the context of `/etc/passwd`

```
# ls -lZ /etc/passwd /root/output.txt
```

- Finally remove the persistent rule and cleanup the system

```
# semanage fcontext -d -t passwd_file_t /root/output.txt
# restorecon -v /root/output.txt → you should be notified about the relabel
Relabeled /root/output.txt from system_u:object_r:passwd_file_t:s0 to system_u:object_r:admin_home_t:s0

# rm /root/output.txt
```

Lab6 - Simple SELinux Booleans 1/2 (5min)

In this lab we'll demonstrate usage and effect of SELinux booleans to a `ping` command for `user_u`.

`# systemctl status auditd.service ; getenforce` → make sure we are good to go

- List all available booleans on your system

```
# getsebool -a | less
# semanage boolean -l | less
```

- First we'll need to create a user with `user_u` role and login via ssh to use it.

```
# useradd -Z user_u useru
# passwd useru
# semanage login -l
```

- … you might need to install and enable `sshd.service` first

```
# ssh useru@localhost
useru$ id -Z    → are you really ssh-ed as a user with SELinux user_u ?
useru$ ping localhost → this should work without any issues
```

Lab6 - Simple SELinux Booleans 2/2 (5min)

```
useru$ ping localhost
```

…

- Now find the related selinux boolean and turn it off.

```
# semanage boolean -l | grep ping
# setsebool -P selinuxuser_ping off
```

- Login again as the confined user and try to use the `ping` or `traceroute` commands again

`useru$ ping localhost` → should give you permission denied … check for AVC denials

`# sesearch -b selinuxuser_ping --allow` → this is list of allow rules used by this boolean

- Finally list all local boolean customizations and return the ping boolean value to an default state

```
# semanage boolean -l -C
# setsebool -P selinuxuser_ping on
```

Extra Lab7 - Allow httpd access to homedirs (15-20 min)

In this lab we'll be using SELinux booleans to allow `httpd` server to access user home directories.

- First try to find httpd related SELinux booleans and and man pages

```
# getsebool -a | grep http
# semanage boolean -l | grep http
# apropos http  → . . you might need to run the `mandb -c' first
```

- Follow instructions in the following HOWTO ;-)

https://www.if-not-true-then-false.com/2010/enable-apache-userdir-with-selinux-on-fedora-centos-red-hat-rhel/

- WITH THE FOLLOWING CORRECTIONS:
  - Step 6) instead of chcon -R … do

```
# restorecon -Rv /home/testuser/public_html
```

  - After that when creating `test.html` and `test.php` files do

```
# chown -R testuser:testuser /home/testuser/public_html
```

Links and References

- short comparison and reference to other authorization models
- Shellshock exploit Demo - https://www.youtube.com/watch?v=Ysshrh4aGOs#action=share
- SELinux coloring book
- https://wiki.centos.org/TipsAndTricks/SelinuxBooleans
- https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/using_selinux/index
- https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/selinux_users_and_administrators_guide/
- https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/using_selinux/index#customizing-the-selinux-policy-for-the-apache-http-server-in-a-non-standard-configuration_configuring-selinux-for-applications-and-services-with-non-standard-configurations