Lesson 1: Mandatory Access Control

1.1 Running SELinux

legal

Pearson

Understanding SELinux Protection

- SELinux prevents any access that hasn't been specifically allowed
- This protects applications from unauthorized access by other applications
- SELinux is not a firewall. Firewalling is for network traffic only
- SELinux adds protection for bugs and zero-day exploits
- SELinux implements mandatory access control to go beyond discretionary access control

SELinux Labels

- SELinux secures parts of the operating system by using labels
- A label is applied to an initiator (users, processes), as well as a target object (files, ports)
  - The initiator is also referred as the source or domain
  - The target is defined as a type and class
- The labels are defined in contexts
- A context has three common parts: user, role, and type. Out of these three, type is the most important part
- In the target, security levels can optionally be used
- To show context, the -Z option can be used with many commands
  - **ls -Z**
  - **ps -Zaux**

SELinux Working

- To perform an action, the initiator context must have access privileges to the target context
- This is defined through rules in the SELinux policy
- For example, an Apache web server may run with the httpd_t context type, and the Apache DocumentRoot is set to httpd_sys_content_t
- If a rule exists in the policy to allow this type of access, access will be allowed
- Otherwise, access is denied with an AVC denied log message

Terminology

- The context of the initiating process is called a *domain*
- The context of the target resource is called the *type*
- The object class to which access is provided in the type is called the *class*
  - File and Socket are examples of classes
- The domain has permissions to access the type and class
- This is summarized in rules and looks like following:
  allow <domain> <type>:<class> { <permissions> };

Type Enforcement

- SELinux context labels are used for type enforcement
- In type enforcement, a specific domain gets permissions to a specific type and class
  allow httpd_t httpd_sys_content_t: file { read };
- Rules defining these permissions are stored in the policy
- If no rules exist for incoming requests, access will be denied

Storing SELinux Attributes

- SELinux rules and contexts are stored in the policy
- File-based attributes are also stored in the filesystem extended attributes
- Use **getfattr -m security.selinux -d /etc/hosts** to print this type of attribute
- If the right procedure is used, file contexts are written to the policy and from there applied to the filesystem
- Writing directly to the filesystem is possible but not recommended

Lesson 1: Mandatory Access Control

1.2 Requiring Mandatory Access Control

Pearson

Why Mandatory Access Control is Needed

- By default, Linux uses Discretionary Access Control (DAC)
- In DAC, files have owners and the owner of the file can grant permissions to other users on a system
- This approach makes it difficult to maintain complete control over security settings
- As an alternative, Mandatory Access Control can be used
- In Mandatory Access Control, security is centrally managed and cannot be changed according to the discretion of individual users
- Mandatory Access Control is used in addition to Discretionary Access Control to make a system more secure

Mandatory Access Control and Linux

- In Linux, different systems for Mandatory Access Control exist
- All systems are backed by the Linux Security Modules (LSM) which are a part of the Linux kernel
- AppArmor originated in 1998 to secure specific processes by creating a profile for them
- SELinux was initially developed in 2000 by NSA and RedHat with the goal to harden Linux completely
- Of the two solutions, SELinux is more inclusive but also harder to implement
- The main Linux distributions offering SELinux are Red Hat family distributions and Gentoo

Lesson 1: Mandatory Access Control

1.3 Understanding SELinux and Discretionary Access Control

SELinux and Discretionary Access Control

- Regular permissions are always evaluated first
- SELinux Mandatory Access Control is used to further fine-tune access permissions
- Thus, a user that doesn't have filesystem permissions, won't get access regardless the SELinux policy settings

Lesson 1: Mandatory Access Control

Lab: Exploring SELinux Settings

Lab: Exploring SELinux Settings

- Check the current SELinux labels for the crond service
- Check SELinux labels for the /etc/crontab file
- Does SELinux allows the crond service to modify the contents of the /etc/crontab file?

Lesson 2: Enabling SELinux

2.1 Managing States and Modes on Red Hat

Understanding SELinux States

- SELinux is either enabled or disabled in the Linux kernel
- Changing between enabled and disabled state requires a system reboot
- On a system where SELinux is enabled, you can toggle between enforcing and permissive mode
- In enforcing mode, SELinux is fully operational and blocks unauthorized requests
- In permissive mode, SELinux does not blocking anything but writes audit events to the audit log
- While analyzing and troubleshooting SELinux, ensure that the auditd process is operational

Managing States

- From the Grub boot menu, following options are available:
  - selinux=[0|1] to enter disabled or enabled state
  - enforcing=[0|1] to set permissive or enforcing mode
- On a running system, use **seinfo** or **getenforce** to get information about the current state and mode
- Use **setenforce** to toggle between enforcing and permissibe mode

Demo: Managing SELinux States

- **reboot**
- enter GRUB menu -> **selinux=0**
- Boot, using Ctrl-X
- **seinfo**
- **getenforce**
- **setenforce**

Lesson 2: Enabling SELinux

2.2 Installing SELinux on Ubuntu

Understanding Ubuntu and SELinux

- Ubuntu doesn't use or support SELinux; Ubuntu uses AppArmor
- You can start Ubuntu with SElinux enabled but you'll need to make multiple modifications to get it working
- The procedure on the next slide shows how to get SELinux running on Ubuntu; consult lesson 6 for tips on troubleshooting
- Until all problems are fixed expect significant parts of Ubuntu to fail
- If you would still like to try it, start enabling SELinux on Ubuntu Server LTS

Demo: Enabling SELinux on Ubuntu

- **sudo systemctl disable --now apparmor**
- **sudo apt update && sudo apt upgrade**
- **sudo apt install policycoreutils selinux-utils selinux-basics auditd -y**
- **sudo selinux-activate**
- **reboot**
- Access GRUB and add **enforcing=0**
- **cat /var/log/audit/audit.log | audit2allow -m initial > initial.te**
- **checkmodule -M -m -o initial.mod initial.te**
- **semodule_package -o initial.pp -m initial.mod**
- **semodule -i initial.pp**

Pearson

Lesson 2: Enabling SELinux

2.3 Understanding Policies

The Role of the Policy

- The SELinux policy contains rules that allow domains to access specific types
- If an activity is not allowed in the policy, access will be denied
- As a result, to enable a distribution to work with SELinux, rules need to be added
- As a quick fix, **audit2allow** can be used to convert all deny messages into policy rules which next are loaded with **semodule**

Understanding Policies

- Red Hat comes with a very inclusive target policy
- In this policy, a wide range of modules are loaded even for services that are not currently installed
- As a result, you won't need to configure a lot if default configuration is used
- On unsupported Linux distributions, a perfectly matching policy is often not available

Understanding RefPolicy

- Refpolicy is a generic policy provided by the SELinux community
- Sources can be found at https://github.com/SELinuxProject/refpolicy
- Using refpolicy allows users to compile their own policy from scratch
- Instructions for Red hat, Debian and gentoo based distributions can be found in the README.md in the Git repository

Lesson 2: Enabling SELinux

Lab: Managing SELinux States

Lab: Managing SELinux States

- Start your Linux distribution with SELinux in disabled state, using a GRUB2 kernel parameter
- Next, add a user
- Restart in normal mode and see what happens
- Once logged in, switch to permissive mode. Next, switch back to enforcing mode

Lesson 3: Understanding Context Labels

3.1 Showing Context Labels

Understanding Labels

- SELinux uses labels to manage security settings
- Labels can be set on initiators like processes and users and targets like files and network ports
- In a label, following elements are used:
  - user: the SELinux user involved (see lesson 10)
  - role: the SELinux role that is used (see lesson 10)
  - type: the type, which defines a set of permissions that belongs to the label
  - an optional security clearance (see lesson 11)
  - an optional category (see lesson 12)

Showing Labels

- To display labels that are currently set, many commands use the **-Z** option
- **ps Zaux** shows all processes with their SELinux context label
- **ls -Z** shows labels set on files and directories

Lesson 3: Understanding Context Labels

3.2 Understanding when to Set Context Labels

When to Set Context Labels

- A good SELinux policy provides standard labels for standard situations
- It implies that if your distribution uses a good policy, you don't have to set anything in standard situations
- Not every distribution comes with a good policy
- If non-standard elements are set, you'll have to set the appropriate label to allow access
- In many cases, you won't have to do anything though

Lesson 3: Understanding Context Labels

3.3 Using the audit.log to Examine Issues

Using audit.log

- SELinux messages are sent to the **auditd** process
- This process writes messages to /var/log/audit/audit.log
- SELinux related messages are marked with AVC (Access Vector Cache)
- Use the audit log to identify issues with labels that are currently set
- Notice that **auditd** is not always installed and enabled by default

Lesson 3: Understanding Context Labels

3.4 Understanding Context Inheritance

Understanding Context Inheritance

- Context labels are set in the SELinux policy
- Files created in a directory inherit the directory context label
- When a file is moved, its context label moves along
- When a file is copied, it gets a context label according to its new location

Understanding Context Inheritance

- Context inheritance is the default
- Even if a file has a specific context, after it is created in a directory it will first inherit the context from the parent directory
- The specific context will only be created later when **restorecon** is used to re-apply context to the entire filesystem

Pearson

Demo: Understanding Context Inheritance

- semanage fcontext -a -t public_content_t /etc/bogus
- touch /etc/bogus
- ls -Z /etc/bogus
- restorecon -v /etc/bogus

Lesson 3: Understanding Context Labels

Lab: Examining SELinux Events

Lab: Examining SELinux Events

- Install the Apache webservice and configure it to use the directory /web as its DocumentRoot
- Start the webserver and access a document using curl localhost. This will fail
- Investigate SELinux related messages that have been logged and check how these relate to the context labels that are currently set

Lesson 4: Managing Context Labels

4.1 Finding the Right Context

Finding the Right Context

- Most services work with a default environment. Check the context set on that environment and use it on the non-default environment
- Use the man pages provided through the selinux-policy-doc package
- Apply instructions generated by **sealert** (see lesson 6)

Lesson 4: Managing Context Labels

4.2 Setting Context on Files

Setting Context on Files

- To manage file context, the context should be written to the policy, and from there applied to the filesystem
- This approach makes it possible to fix mislabeled filesystems using one simple command
- Use **semanage fcontext** to change the context in the policy
- Next, use **restorecon** to apply the context from the policy to the filesystem

Demo: Using **semanage fcontext**

- **mkdir /files**
- **touch /files/file{1..10}**
- **semanage fcontext -a -t public_content_t "/files(/.*)?"**
- **ls -Zd /files**
- **restorecon -Rv /files**

Setting Context to the Filesystem Only

- Use **chcon** to set the context to the filesystem only
- This can be convenient in exceptional cases like HA cluster resources or temporary settings
- If **chcon** is used to change the context on a file that has it's own context set in the policy, **restorecon** will relabel the file

Demo: Using **chcon**

- **touch /tmp/chconfile**
- **chcon -t httpd_sys_content_t /tmp/chconfile**
- **ls -Z /tmp/chconfile**
- **restorecon -Rv /tmp**
- **ls -Z /tmp/chconfile /etc/hosts**
- **chcon -t httpd_sys_content_t /etc/hosts**
- **restorecon -v /etc/hosts**

Lesson 4: Managing Context Labels

4.3 Using **semanage port** to Set Context on Ports

Pearson

Labeling Ports

- Port context is only managed in the policy
- There is no need to use **restorecon** to apply port context from the policy to anywhere
- Use **semanage port -a -t http_port_t -p tcp 82** to set port context

Lesson 4: Managing Context Labels

4.4 Using Customizable Types

Understanding Customizable Types

- A customizable type is a type that will persist through a standard **restorecon**
- They are commonly used on files that don't have a fixed location
- Since the file location is not fixed, it's hard for the policy writer to write a rule for them
- A list of customizable types is kept in /etc/selinux/*/contexts/customizable_types
- Thus, customizable types will not be relabeled which you will see in the following demo

Demo: Understanding Customizable Types

- touch /tmp/customizable1
- ls -Z /tmp/customizable1
- chcon -t container_file_t /tmp/customizable1
- ls -Z /tmp/customizable1
- restorecon -Rv /tmp
- ls -Z /tmp/customizable1

Lesson 4: Managing Context Labels

Lab: Running SSH on Port 443

Lab: Running SSH on Port 443

- Configure SSH to run on port 443 permanently

Lesson 5: Using Booleans

5.1 Understanding Booleans

Understanding Booleans

- Booleans are provided to make SELinux behavior optional
- It is used to change the way how SELinux reacts in specific cases
- By applying a boolean, a complete set of rules is applied to allow or deny specific behavior
- Use **getsebool** to print a list of all booleans currently available
- Use **semanage boolean -l** for a list of all booleans including a description
- Use **sesearch -b boolean_name -A** to see which rules exactly a boolean changes

Lesson 5: Using Booleans

5.2 Using Booleans

Using Booleans

- Use **getsebool -a** for a list of available booleans
- Boolean names in general are rather intuitive
- Use **setsebool -P boolean_name on** to switch the boolean on

Lesson 5: Using Booleans

5.3 Finding Booleans

Finding Booleans

- Based on an AVC deny action that you have found in /var/log/audit/audit.log, you can use **sesearch** to check if there is a boolean that would allow this behavior
- Look for the source context type and the target context type
- Find the appropriate Boolean using **sesearch -s source_t -t target_t -A**

Demo: Finding Booleans

- Enable Apache userdir access by adding the following lines to /etc/httpd/conf.d/userdir.conf
  - UserDir enabled
  - UserDir public_html
- **useradd anna; chmod -R 755 /home/anna**
- **su - anna**
- **mkdir public_html**
- **echo "hello anna" > public_html/index.html; exit**
- **systemctl restart httpd**

Demo: Finding Booleans

- From a browser (not curl) **http://localhost/~anna**
- **grep AVC /var/log/audit/audit.log**
- **sesearch -s httpd_t -t httpd_user_content_t -A**
- **setsebool -P httpd_enable_home_dirs on**

Lesson 5: Using Booleans

Lab: Configuring vsftpd for Anonymous Uploads

Lab: Configuring vsftpd for anonymous uploads

- Install the vsftpd service and configure it such that anonymous user uploads are allowed. Use **sesearch** to find the appropriate booleans

Lesson 6: Troubleshooting SELinux

6.1 Troubleshooting SELinux Issues

Generic Troubleshooting

- Confirm that the issue is really caused by SELinux: **setenforce 0** and try again
- Once it is confirmed that the issue is caused by SELinux, analyze logs in /var/log/audit/audit.log and define a hypothesis of what might be wrong
- Use information logged by **sealert** to confirm

Lesson 6: Troubleshooting SELinux

6.2 Understanding the Audit Logs

Understanding the Audit Logs

- SELinux denial messages are written to the Access Vector Cache (AVC)
- This cache is used to remember SELinux decisions and used to make SELinux faster
- To analyze SELinux denials, the Access Vector Cache can be checked through the Linux audit daemon
- Logs are typically found in /var/log/audit/audit.log
- Use **grep AVC /var/log/audit/audit.log** for a complete overview of messages that have been logged
- To see all AVC denied messages related to a specific action, SELinux should be in permissive mode

Analyzing a Log Message

- AVC denied messages logged by auditd always have a specific structure

```
type=AVC msg=audit(1687246293.012:187): avc:  denied  { getattr } for  pid=3063 comm="httpd"
path="/home/anna/public_html/index.html" dev="dm-0" ino=52352420
scontext=system_u:system_r:httpd_t:s0 tcontext=unconfined_u:object_r:httpd_user_content_t:s0
tclass=file permissive=0
```

- type=AVC: applies to all SELinux messages
- msg=audit(1687246293) audit time stamp in epoch: convert to human readable format using **date -d @**
- 012:187: audit event number and order number
- denied: the type of action SELinux has taken
- getattr: the specific permission that was denied

Analyzing a Log Message

- for pid=3063: the process of the initiator of the action
- comm="httpd": the command used by the initiator
- path=: the path that was accessed by the initiator
- dev=: the storage device on which that path resides
- ino=: the inode number of the file that was accessed
- scontext=: the context of the initiator
- tcontext=: the context of the target
- tclass=: the kind of target object
- permissive=: a boolean value that indicates if the system was in permissive mode or not

Lesson 6: Troubleshooting SELinux

6.3 Understanding Dontaudit Rules

Pearson

Understanding Dontaudit

- While working with SELinux, many cosmetic audit messages are logged
- These are flagged as Dontaudit and won't show in the audit logs
- Use **seinfo | grep audit** for an impression of the amount of Dontaudit rules which you haven't seen in the audit logs
- In some cases, it can make sense to temporarily also show dontaudit rules
- To do so, use **semanage dontaudit off**
- This should always be a temporary solution. Once you're done analyzing, use **semanage dontaudit on** to no longer show dontaudit rules
- To show a list of all rules flagged as dontaudit, use **sesearch --dontaudit**

Lesson 6: Troubleshooting SELinux

6.4 Using **audit2allow**

Using **audit2allow**

- **audit2allow** is a generic command that allows you to convert AVC denied messages into a policy module that will allow the action
- **audit2allow** generates a policy module which can then be enabled in an easy way
- Never use **audit2allow** without analyzing what exactly you are going to allow
- Best practice: before using **audit2allow** to allow specific actions, use **audit2why** to find out what was going on

Demo: Using **audit2allow**

- **grep AVC /var/log/audit.log | grep http > http_denied.log**
- edit http_denied.log
- **cat http_denied | audit2why**
- **cat httpd_denied | audit2allow -M myhttp**
- **semodule -i myhttp**

Lesson 6: Troubleshooting SELinux

6.5 Using **sealert**

Understanding **sealert**

- **sealert** is a part of the setroubleshoot-server package
- It automatically interprets AVC messages written to the audit.log and converts them into a more readable message in the default syslog system in use
- In the **sealert** output, you'll find suggestions on how to fix issues
- Using **sealert** can be a nice addition to troubleshoot SELinux. However, it should never be used without additional interpretation
- Without considering additional interpretation, some suggestions can miss the point while some could even be dangerous

Lesson 6: Troubleshooting SELinux

6.6 Loading SELinux Manually

Loading SELinux Manually

- While starting in troubleshooting mode using the GRUB arguments init=/bin/bash or rd.break, SELinux is not loaded
- If in this mode SELinux access is required, use **load_policy -i**

Demo: Loading SELinux Manually

- From the GRUB boot menu prompt, start your system with **init=/bin/bash**
- **export PATH=$PATH:/usr/sbin**
- **getenforce**
- **load_policy -i**
- **getenforce**
- **exec /usr/lib/systemd/systemd**

Lesson 6: Troubleshooting SELinux

Lab: Troubleshooting SELinux

Lab: Troubleshooting SELinux

- Configure the Apache webserver to offer its services on port 82
- Apply the appropriate troubleshooting procedure to fix all issues

Lesson 7: Analyzing Booleans and Rules

7.1 Analyzing the Policy

Analyzing the Policy

- To analyze rules in the policy, SELinux provides two useful tools – **seinfo** and **sesearch**
- **seinfo** shows information about SELinux objects
  - Use it to find what exactly an SELinux component is allowed to
- **sesearch** allows users to search for rules in the policy and query relations between source and target
  - Use it to find booleans or rules that do what you need in a specific situation

Lesson 7: Analyzing Booleans and Rules

7.2 Terminology

Understanding Terminology

- *Context*: The SELinux security settings that are applied to every process and system resource
- *Label*: The specific settings in the context, consisting of a user, role and type part
- *Type*: The part of the label that identifies the SELinux security properties for processes and other system resources
- *Type enforcement*: The process where SELinux determines which source application type has access to which target application type
- *Domain*: The privileges that are defined in a security context and used by an application

Understanding Terminology

- *Target*: The target context that the source application (domain) tries to access. Within one target different *classes* can be addressed
- *Class*: The system resource that is accessed (often files and ports)
- *Permissions:* The specific SELinux allowances that are granted when a domain accesses a specific target:class. Permissions are granted to a specific class within a target. Different classes can have different permissions
- *Attributes*: Attributes are generic entities that can be used in an SELinux policy. The daemon attribute allows to give specific permissions to all daemons so that you don't have to set it for each individual daemon again and again

Terminology Applied

- In SELinux, rules are defined: allow user_t var_log_t: dir { getattr search open read };
- In this rule, the following elements are used:
  - user_t: the domain
  - var_log_t: the target
  - dir: the class within that target
  - { getattr search open read }: the permissions that apply to that target

Lesson 7: Analyzing Booleans and Rules

7.3 Using **sesearch**

Using sesearch

- **sesearch** is used to perform advanced queries on the policy
- It uses different parameters to specify what you are searching for
  - **sesearch -s sshd_t -A** shows rules that sshd_t is allowed to access
  - **sesearch -t public_content_t -A** shows source domains that are allowed to access this target
  - This can be combined for more specific results: **sesearch -s vmware_t -t public_content_t -A**
  - You can add the **-p** option to search for source domains that have a specific permission to the target domain: **sesearch -t shadow_t -c file -p write -A**
  - You can use it to analyze rules that are enabled by booleans: **sesearch -b ftpd_anon_write -A**

Pearson

Lesson 7: Analyzing Booleans and Rules

7.4 Using **seinfo**

Using **seinfo**

- Use **seinfo** to get detailed information about any attribute in SELinux and add **-x** for detailed information
- **seinfo -a domain -x** will print all types with the domain attribute
- **seinfo -a domain -u** prints users
- **seinfo -a domain -r** prints roles
- **seinfo -a domain -b** shows booleans
- **seinfo -a unconfined_domain_type -x** shows all unconfined domains

Using **seinfo**

- **seinfo -c** shows all classes (the item to which access is given)
- **seinfo -c file -x** shows all permissions that can be used for the file class
- **seinfo -a** shows all attributes
- **seinfo -a exec_type -x** shows all domains that use this type
- **seinfo -a httpd_content_type -x** shows all context types that do something with the httpd_content_type attribute

Lesson 7: Analyzing Booleans and Rules

7.5 Finding What a Domain can Do

Demo: Finding What a Domain can Do

- **sesearch -s auditd_t -t var_t -A** will show what auditd_t can do on var_t
- **seinfo -t auditd_t -x** shows extended information about auditd_t, including the attributes assigned to it
- **seinfo -a** shows all attributes
- **seinfo -a domain -x** shows all types associated to the domain attribute

Demo: Finding Specific Information

- **sesearch -A -s httpd_t -t httpd_sys_script_exec_t -c file -p execute** will show that the httpd_enable_cgi boolean implements the rule that allows httpd_t to run a file.
- **sesearch -A -s domain -t lib_t -c dir** lists the rules that allow domains to access directories with the **lib_t** type

Pearson

Lesson 7: Analyzing Booleans and Rules

7.6 Analyzing Booleans

Analyzing Booleans

- Start by getting an overview of what a boolean can do: **sesearch -b ftpd_full_access -A**
- The types include the non_security_file_type attribute. To explore this attribute, use **seinfo -a non_security_file_type -x**
- You now know what non_security_file_type applies to and how it makes it easy to provide access to thousands of types

Lesson 7: Analyzing Booleans and Rules

7.7 Analyzing Transition Rules

Understanding Context Transitioning

- To create exceptions for the default behavior of inheritance, SELinux supports transitioning
- Without transitioning, all processes would run with kernel_t and all files would have the root_t context type
- A transition rule defines that when a process is forked, a new context can be set
- A transition rule looks like type_transition init_t initrc_exec_t: process initrc_t
- This specifies that when a process that runs with init_t executes a file that has initrc_exec_t, the resulting process will get the initrc_t context type

Understanding the Need for Transition

- By default, a child process inherits the context of the parent process
- This is not always useful, because if would have all systemd services run with init_t
- To set the proper context for a domain, the child process (sshd in this case) gets its context from the file that it executes
- /usr/sbin/sshd is set to sshd_exec_t
- A transition rule identifies sshd_exec_t as entrypoint for sshd_t, with the result that the process started from this file gets sshd_t and not init_t

Default Transition and Custom Transitions

- The SELinux policy comes with transition rules
- Custom transition rules can be set using commands like **runcon** (covered in lesson 9)
- To get an overview of all transition rules, use **sesearch -c process -p transition -A**
- To verify if a specific domain transition is allowed, include a source and a target in your command: **sesearch -s sshd_t -t sshd_exec_t -c file -p entrypoint -A**

Understanding Transition Requirements

- To allow a domain transition to happen, there are three requirements:
  - The source domain has the SELinux execute permission on the file used by the child process
  - The file context of the child process has the SELinux entrypoint permission set for the source domain. The entrypoint permission is used to identify two SELinux contexts as related
  - The source domain is allowed to transition to the target domain through the SELinux transition permission

Lesson 7: Analyzing Booleans and Rules

Lab: Investigating Booleans

Lab: Investigating Booleans

- While using **sealert**, you have found the suggestion to either use the ftpd_anon_write boolean or the ftpd_full_access boolean. Use the appropriate tools to find which of these booleans is more secure

Lesson 8: SELinux Modules

8.1 Managing Modules

Pearson

Understanding Modules

- SELinux comes with modules that contain rules to allow applications to do what is required
- By default, many modules are provided to allow any application to be used in an SELinux environment
- The easiest way to add non-standard rules to the policy is by defining custom modules
- Custom modules can be written in either CIL or in M4
- Alternatively, modules can be generated with tools like **audit2allow**

Demo: Managing Modules

- **semodule -l**
- **semodule -d zabbix**
- **semodule -e zabbix**

Creating Custom Modules

- Any functionality can be enabled by creating custom modules
- However, using custom modules is not the best way that can be imagined as it makes the policy more complex
- In many cases, using custom modules can be avoided by applying the right context labels

Lesson 8: SELinux Modules

8.2 Writing Custom Modules

Understanding Module Format

- Policy modules can be written in m4 or CIL format
- This is what it looks like in CIL

```
(allow cupsd_lpd_t cupsd_var_run_t (sock_file (read)))
```

- And this is the m4 equivalent

```
module local_cupslpd-read-cupssock 1.0;

require {
    type cupsd_var_run_t;
    type cupsd_lpd_t;
    class sock_file read;
}

#============= cupsd_lpd_t ==============
allow cupsd_lpd_t cupsd_var_run_t:sock_file read;
```

Understanding Custom Rules

- Rules written in m4 use the following syntax:
  - allow <source> <destination> : <class> <permissions> ;
  - See audit.log for examples
- <source> is always a domain
- <destination> can be anything
- <class> is the thing that is accessed in the target
  - file, directory, socket, capability, etc
  - use **seinfo -c** for a complete overview
- Each class has specific permissions associated to it
  - Use **seinfo -c<class> -x** to show
  - As in **seinfo -cfile -x**

Translating Audit Messages to Custom Rules

- Consider this message in audit.log

```
type=AVC msg=audit(1413357425.988:1060): avc: denied { name_bind } for pid=29198
comm="sshd" src=443 scontext=unconfined_u:system_r:sshd_t:s0-s0:c0.c1023
tcontext=system_u:object_r:http_port_t:s0 tclass=tcp_socket
```

- This translates into the following rule if you want to allow:

```
allow sshd_t http_port_t : tcp_socket { name_bind };
```

- Avoid writing rules manually, use **audit2allow** instead
  - **grep ssh /var/log/audit/audit.log | grep AVC | audit2allow –M mypolicy**

Pearson

Demo: Manually Adding Policy Files - 1

- Start by creating a .te file (~/sander.te)

```
module sander 1.0;
require {
        type sshd_t;
        type http_port_t;
        class tcp_socket { name_bind };
}
allow sshd_t http_port_t:tcp_socket { name_bind };
```

Demo: Manually Adding Policy Files - 2

- File system resources can be defined using a .fc file
- Add the following to sander.fc file:

```
/opt/sander(/.*)?  system_u:object_r:httpd_sys_content_t:s0
```

Pearson

Demo: Manually Adding Policy Files - 3

- Create the policy module:
  - **checkmodule –M –m –o sander.mod sander.te**
  - **semodule_package –o sander.pp –m sander.mod -f sander.fc**

- Run the policy module
  - **semodule –i sander.pp**

Testing:
mkdir /opt/sander
restorecon -Rv /opt/sander

Lesson 8: SELinux Modules

8.3 Generating Custom Modules

Generating Custom Modules

- The **audit2allow** command can be used to convert AVC denied messages into a policy module that allows access
- Instructions provided by **sealert** frequently suggest using **audit2allow** to generate policy modules
- Generating policy modules using this method is dangerous if you're not well acquainted with the process and steps

Best practice: Carefully filter AVC denied messages from the audit log. Write them to a file and use that file as an input for audit2allow

Managing Module Priority

- Modules may have conflicting rules
- To handle conflicting rules correctly while inserting the module, a priority can be set
- Priority is set between 1 and 999, and rules with a higher priority have precedence
- Use **semodule -X 500 -i mymodule.pp** to set priority while inserting mymodule.pp

Demo: Using **audit2allow** Carefully

- **grep AVC /var/log/audit/audit/log | grep http > http_logs.txt**
- **less http_logs.txt**
- **cat http_logs.txt | audit2allow -M myhttp**
- **semodule -i myhttp.pp**

Lesson 8: SELinux Modules

Lab: Enabling your Application with Modules

Lab: Enabling your Application with Modules

- Write a module that allows httpd to bind to port 88
- Consider if this is the best way to enable this type of functionality

Lesson 9: Making any Application work with SELinux

9.1 Understanding Options for Running Custom Applications

Pearson

Options for Running Custom Applications

- Run the application as unconfined or permissive
- Have the application run with the context type inherited from the parent and tweak that using messages in audit.log
- Use **runcon** to set a (dummy) context type for the application and tweak that based on AVC messages in audit.log
- Use **sepolicy generate** to create your own context types and apply these to the application: preferred option

Lesson 9: Making any Application work with SELinux

9.2 Using Unconfined Domains

Using Permissive Domains

- Using permissive domains allows you to define exceptions. The domain is the source context type (like httpd_t)
- Use **semanage permissive -a domain** to set a permissive domain
- Switch off using **semanage permissive -d somelabel_t**
- Use **semanage permissive -l** for an overview of domains that are currently set to permissive

Handling Unconfined Domains

- An unconfined domain is an application that can do whatever it wants in an SELinux environment - as long as this is not blocked by DAC
- Unconfined domains exist for environments where administrators only want to focus on restricting a few applications
- Use **seinfo -tunconfined_t** to find out if unconfined domains are supported. You'll get an error message if unconfined domains are not supported
- Use **seinfo -aunconfined_domain_type -x** for a list of all unconfined domains

Lesson 9: Making any Application work with SELinux

9.3 Using **runcon** to Run Applications with a Specific Context

Changing Application Labels with **runcon**

- Use **runcon -u user_u -r role_r -t type_t yourapp** to change application context
  - **runcon -u system_u -r system_r -t httpd_t vsftpd** will run vsftpd with the httpd context type
  - Modify the systemd unit file to start your application using **runcon**
- Check **sealert** when failing
- Notice that this procedure is not the most efficient. Hence, better use **sepolicy** to generate application specific context labels

Setting the Appropriate Context Label

- When using **runcon**, a context type must be set
- Ideal situation is to create your own context type for the application
- Alternatively, use an uncommon context type (xend, zarafa) and tune that for your application to do its work
- Next, start using the application and work with audit.log and **audit2allow** to create a policy module for your application

Lesson 9: Making any Application work with SELinux

9.4 Using **sepolgen** to Generate Application Policy Modules

Using **sepolgen**

- **sepolicy generate** (**sepolgen**) provides a generic interface to generate a policy for any application
- After generating a generic policy, define the exceptions by creating custom policy files, for instance using **audit2allow**

Demo: Generating a Custom Application Policy

- **git clone https://github.com/sandervanvugt/selinux; cd selinux**
- **sudo dnf install policycoreutils-devel setools-console gcc**
- **gcc -o mydaemon mydaemon.c**
- **sudo cp mydaemon /usr/local/bin**
- **sudo cp mydaemon.service /etc/systemd/system/**
- **sudo systemctl start mydaemon**
- **ps Zaux | grep mydaemon** # should show as unconfined
- **sepolicy generate --init /usr/local/bin/mydaemon** #check files
- **./mydaemon.sh** # ignore error messages

Demo: Generating a Custom Application Policy

- **sudo systemctl restart mydaemon**
- **ps Zaux | grep mydaemon**
- **grep AVC /var/log/audit/audit.log** # access to /var/log/messages is denied
- **sealert -l "*"**
- Use **audit2allow** to check suggestions: **ausearch -m AVC -ts recent | audit2allow -R**
- Check what the generic policy interface is doing: **cat /usr/share/selinux/devel/include/system/logging.if**
- Add the rule to the type enforcement file: **echo "logging_write_generic_logs(mydaemon_t)" >> mydaemon.te**

Pearson

Demo: Generating a Custom Application Policy

- **./mydaemon.sh**
- **ps Zaux | grep mydaemon**
- **sudo sesearch -m AVC -ts recent** should give no recent matches (use **date -d '@timestamp'** to verify exact times

Pearson

Lesson 9: Making any Application work with SELinux

Lab: Running any Application on a SELinux System

Lab: Running any Application on an SELinux system

- Install the application myapp from the course Git repository at https://github.com/sandervanvugt/selinux
  - Copy the myapp file to /usr/local/bin
  - Copy the myapp.service file to /etc/systemd/system/
- Make sure this application runs with its own context in an SELinux environment

Lesson 10: SELinux Users

10.1 Understanding Users and Roles

Understanding SELinux Users

- An SELinux user is an SELinux security profile that is managed independent of Linux users
- All Linux users are mapped to an SELinux user
- By default, this is the unconfined_u user
- Mapping to unconined_u doesn't impose any restrictions
- Check current user mappings using **id -Z**

Automatically Mapping Linux Users

- Each Linux user is mapped to the __default__ user
- The __default__ user is mapped to the SELinux unconfined_u user
- Unconfined users are subject to minimal SELinux restrictions
- Applications started by unconfined users, however, are subject to SELinux application restrictions
- As a result, applications that are started as a confined domain by an unconfined user, will run confined anyway
- Use **semanage login -l** for an overview of current mappings
- Use **seinfo -u** for a list of current SELinux users

Understanding SELinux Roles

- SELinux users are connected to SELinux roles
- Apart from different administrator roles, other roles are provided to restrict users
- Different administrator roles are available to allow for delegation of administator tasks
- Booleans are provided to fine-tune access permissions for specific roles
- Use **seinfo -r** for an overview of all roles
- Use **man** for more information about any of the roles

Roles Access Rights

Role	Type	XDM login	su/sudo	x in home	networking
guest_r	guest_t	no	no	yes	no
xguest_r	xguest_t	yes	no	yes	browser
user_r	user_t	yes	no	yes	yes
staff_r	staff_t	yes	sudo only	yes	yes
auditadm_r	auditadm_t		yes	yes	yes
secadm_r	secadm_t		yes	yes	yes
sysadm_r	sysadm_t	boolean	yes	yes	yes

Modifying Roles Access Rights

- Roles access rights can be tuned using Booleans
  - ssh_sysadm_login
  - xdm_sysadm_login
  - sysadm_exec_content
  - staff_exec_content
  - guest_exec_content
  - xguest_exec_content
  - xguest_connect_media

Lesson 10: SELinux Users

10.2 Mapping Linux Users to SELinux Users

Default Settings

- By default, Linux users are mapped to unconfined_u
- Use **useradd -Z** to map a new Linux user to an SELinux user
  - **useradd -Z staff_u isabelle**
- To map an existing user to an SELinux user, use **semanage login**
  - **semanage login -a -s user_u -r s0 daphne**
- And verify using **semanage login -l**
- Use __default__ for all new users:
  - **semanage login -m -s user_u -r s0 __default__**
- Notice that users created using __default__ don't show in **semanage login -l**
- Log in as that user, and use **id -Z** to verify the current mapping

Pearson

Understanding unconfined_u User Access

- Linux users, on login by default, are all mapped to the unconfined_u user
  - **semanage login -l** shows current mappings
- By default, unconfined_u users have access to items running in unconfined domains.
  - Use **seinfo -aselinux_unconfined_type -x** to find out what exactly those are
  - **seinfo -xt user_t** shows information about domains user_t has access to

Pearson

Lesson 10: SELinux Users

10.3 Using Booleans to Manage SELinux Users

Managing SELinux Users

- By default, users root and __default__ are mapped to unconfined_u
- For perfect security, these users should be mapped to SELinux users

Tip: Mapping the root user to an SELinux user can lock you out. While testing how SELinux users work, it is better to leave the root user as unconfined_u

Demo: Managing SELinux Users

- **semanage user -l**
- **useradd linda; echo password | passwd --stdin linda**
- **useradd -Z sysadm_u -G wheel lisa; echo password | passwd --stdin lisa**
- **semanage login -a -s user_u linda**
- **semanage login -l**
- login as linda (no su)
- **su -** # will be denied

Demo: Configuring Default Settings

- **semanage login -l**
- **semanage login -m -s sysadm_u root** #depends on distro version
- **semanage login -m -s user_u -r s0 __default__**
- **semanage login -l**
- **useradd anna; echo password | passwd --stdin anna**
- **getsebool -a | grep -E 'user|sysadm|staff'**

Demo: Restoring Access

- At this point, there is no root logging from XDM anymore. You should have created a user that is a member of staff_u as well
- Reboot with **systemd.unit=multi-user.target**
- **setsebool -P xdm_sysadm_login on**
- **setsebool -P ssh_sysadm_login on**
- Now, you'll be able to log in from XDM again

Lesson 10: SELinux Users

10.4 Restricting Root

Pearson

Restricting Root

- Using SELinux users is the first good step to restrict root access
- First, use **semanage login -m sysadm_u root** to change the root SELinux user from unconfined_u to sysadm_u
- Next, decide if you want to open the access a little bit
  - **setsebool -P xdm_sysadm_login on**
  - **setsebool -P ssh_sysadm_login on**
- However, using just SELinux users is not enough. To restrict the root user from accessing sensitive files, you should use MLS, MCS, or both

Lesson 10: SELinux Users

Lab: Creating a Kiosk User

Lab: Creating a Kiosk User

- Create the Linux user kiosk. Use SELinux to restrict this user such that the user can only work from a browser and nothing else

Lesson 12: Using Multi-Category Security (MCS)

12.1 Understanding MCS

Understanding MCS

- Multi Category Security (MCS) can be used in addition to SELinux context labels to organize users and data in categories
- When MCS is used, a user can only access data if the data has been classified in the same category as the user
- MCS is evaluated after discretionary acess control and context labels, and as such is used to add another layer to SELinux security
- MCS can be used in the targeted policy, as well as, in addition to MLS

MCS Labels

- To define MCS, categories are defined from c0 up to c1023
- Users and files can be assigned to multiple categories
- Alternatively, a descriptive text label can be used

Configuring MCS

- In the targeted policy, MCS security is only used for the following specific applications:
  - Openshift
  - virt
  - sandbox
  - network labeling
  - containers
- To configure MCS for users, you'll first need to create a local SELinux module

Demo: Using MCS in the Targeted Policy

- **dnf install setools-console policycoreutils-python-utils**
- **echo "(typeattributeset mcs_constrained_type (user_t))" >> local_mcs_user.cil**
- **semodule -i local_mcs_user.cil**
- **seinfo -xt user_t | grep mcs**

Defining Category Labels

- Optional human-readable category labels can be set in /etc/selinux/targeted/setrans.conf
- In this file, use lines like s0:c0=public, to define an MCS group with the name public
- In the targeted policy, only security clearance s0 is supported
- In the MLS policy, use any clearance level you want
- (Re)start the **mcstrans** systemd service to make the changes effective
- Use **chcat -L** to show current settings

Demo: Setting Category Labels

- Edit /etc/selinux/targeted/setrans.conf to include the following
  - s0:c0=public
  - s0:c1=sales
  - s0:c2=accounting
- **dnf install mcstrans**
- **systemctl enable --now mcstrans**
- **chcat -L**

Pearson

Lesson 12: Using Multi-Category Security (MCS)

12.2 Grouping Users and Applications with MCS

Assigning Categories for Users

- First, the SELinux user needs to be configured with the required categories
- Use **semanage user -m** to assign categories to the SELinux user
- Next, the Linux user must be mapped to the SELinux user. Only categories assigned to the SELinux user can be assigned to the Linux user
- Use **semanage login -m** to assign categories to the Linux user
- Use **chcat -L** to list category assignments for users
- The user needs to login to see effect
- Opening a shell in sudo or su doesn't work
- In the targeted policy, the clearance level cannot be set:
  - **semanage user -m rs0:c0.c5-s1:c0.c10 user_u** doesnt' work
  - **semanage user -m rs0-s0:c0.c9 user_u** will work

Demo: Assigning Categories for Users

- **useradd -Z user_u marcha**
- **useradd -Z user_u daphne**
- **useradd -Z staff_u -g wheel isabelle**
- **for i in isabelle daphne marcha; do echo password | passwd --stdin $i; done**
- **semanage user -l**
- **semanage user -m -rs0-s0:c0.c9 user_u**
- **semanage login -m -rs0:c10 marcha** # will fail
- **semanage login -m -rs0:c1 marcha**
- **semanage login -m -rs0:c1.c5 daphne**
- **semanage login -l**
- **chcat -L -l marcha**

Setting Categories for Files

- Users can only access files with a category matching their own category
- **semanage** doesn't support setting categories on files
- Use **chcat** to set categories on files
- Notice that while using **chcat**, the context settings are not written to the policy but directly to the inode
- As a result, category settings won't survive a file system relabel
- Consider using Ansible playbooks to manage file contexts consistently

Demo: Setting Categories for Files

- **semanage login -l** # should have daphne and marche with different settings
- **echo cat1 > /tmp/testfile1**
- **echo cat2 > /tmp/testfile2**
- **chcat -- +c1 /tmp/testfile1**
- **chcat -- +c5 /tmp/testfile2**
- login as marcha
  - **cat /tmp/testfile1** # succeeds
  - **cat /tmp/testfile2** # fails
- login as daphne
  - **cat /tmp/testfile1** # succeeds
  - **cat /tmp/testfile2** # succeeds

Lesson 12: Using Multi-Category Security (MCS)

12.3 Combining MLS and MCS

Combining MLS and MCS

- MLS and MCS can be combined to work with security clearances within specific groups
- In this case, the MLS security labels are used to define the clearance levels, and the groups are defined with MCS

Security level	Project A	Project B	Project C
Unclassified	s0:c0	s0:c1	s0:c2
Confidential	s1:c0	s1:c1	s1:c2
Secret	s2:c0	s2:c1	s1:c3

- The effective security context of the user is the combination of SELinux user, SELinux role, SELinux type, MLS classification level and MCS category

Lesson 12: Using Multi-Category Security (MCS)

Lab: Configuring MCS

Lab: Configuring MCS

- Configure MCS in the targetet policy. Ensure that you have users linda and anna. Anna should be assigned to categories c0-c9 and linda should be assigned to c0 only.
- Create the files /tmp/testfile1 with category c0 and /tmp/testfile2 with category c3. Verify that anna can read both files and linda can read only /tmp/testfile1
- Does the current configuration allow anna to write to /tmp/testfile2? Explain your answer

Lesson 13: SELinux and Containers

13.1 Understanding Container SELinux Needs

Understanding Container SELinux Needs

- Containers by default are stored with the container_t types
- If this would be the only configuration, there would be no difference between containers and containers would not be secure
- As a solution, when containers are started, a MCS category is assigned
- Each container gets assigned two random categories to ensure that from an SELinux perspective, containers have their own identity

Understanding Container Storage Access

- To access the shared storage in stand-alone containers, bind-mounts are commonly used
- In a bind mount, a directory on the host is mounted within the container
- To allow the container to get complete access to this directory, the SELinux context must be set to container_file_t, and MCS categories need to be assigned
- The best way to do this is by using the :Z option while performing the bind mount with **podman run -v**:
  - **mkdir container; podman run -d -v /home/student/container:/container:Z nginx**
- Using **semanage fcontext** on the bind mounted directory doesn't set the MCS category

Understanding **udica**

- To automatically generate a policy configuration for containers, the **udica** tool is provided
- To use this tool, a JSON file must be generated with all the container properties
- Based on this JSON file, **udica** generates a policy that is specific for this container

Lesson 13: SELinux and Containers

13.2 Configuring Container Storage Access

Demo: Using the :Z Option

- **mkdir ~/container{1..5}**
- **podman run -d -v /home/student/container1:/container1 --name container1 nginx**
- **podman exec -it container1 sh**
- **touch /container1/file1** # will fail
- **exit**
- **ls -ldZ ~/container1**

Demo: Using the :Z Option

- **podman run -d -v /home/student/container2:/container2:Z --name container2 nginx**
- **podman exec -it container2 sh**
- **touch /container2/file2**
- **exit**
- **ls -ldZ ~/container2**

Demo: Understanding why :Z is better

- **semanage fcontext -a -t container_file_t /home/student/container3**
- **restorecon -Rv /home/student/container3**
- **podman run -d -v /home/student/container3:/container3 --name container3 nginx**
- **podman exec -it container3 sh**
- **touch /container3/file3** # will work
- **exit**
- **ls -ldZ ~/container3** # has a context that is too generic

Pearson

Lesson 13: SELinux and Containers

13.3 Using **udica** to Configure Container Access

Understanding **udica** Benefits

- Containers may have access to several shared resources
- While the :Z option is useful for creating bind mounts, it doesn't help in protecting the port access
- Hence, using **udica** is better as it analyzes a container JSON file and provides access for this container only to the shared resources
- Even if **udica** is used, shared storage should still be bind-mounted with the :Z option
- In order to do this, context labels that include MCS categories are used

Demo: Using **udica**

- **podman run -d -v /home/student/container4:/container4 --rm --name container4 -p 8082:80 nginx**
- **podman inspect container4 > container4.json**
- **sudo udica -j container4.json container4**
- **cat container4.cil**
- **sudo semodule -i container4.cil /usr/share/udica/templates/{base_container.cil,net_container.cil}**
- **podman kill container4**
- **podman run -d -v /home/student/container4:/container4 --name container4 -p 8082:80 --security-opt label=type:container4.process nginx**
- **ls -ldZ container4**

Demo: Using **udica**

- **podman run -d -v /home/student/container5:/container5:Z --rm --name container5 -p 8083:80 nginx**
- **podman inspect container5 > container5.json**
- **sudo udica -j container5.json container5**
- **cat container5.cil**
- **sudo semodule -i container5.cil /usr/share/udica/templates/{base_container.cil,net_container.cil}**
- **podman kill container5**
- **podman run -d -v /home/student/container5:/container5 --name container5 -p 8083:80 --security-opt label=type:container5.process nginx**
- **ls -ldZ container5**

Lesson 13: SELinux and Containers

Lab: Configuring SELinux for Containers

Lab: Configuring SELinux for Containers

- Run a nginx container that bind mounts the directory ~/labcontainer to the directory /labcontainer and provides its services on the host port 8084.
- Ensure that this container has the best possible SELinux protection

Lesson 14: Managing SELinux with Ansible

14.1 Using SELinux Ansible Modules

Understanding SELinux and Ansible

- To configure different systems consistently, **semanage export** can be used as: **semanage export -f ./mypolicy.mod**
- Next, after copying over these settings to a remote host, they can be imported there: **semanage import -f ./mypolicy.mod**
- For a more systematic approach to manage SELinux settings, Ansible can be used

Understanding Ansible

- Ansible is a commonly used system for configuration management
- It requires a controle node that runs the Ansible software, as well as managed nodes that are accessed through secure shell
- For more information about setting up an Ansible managed environment, consult my "Ansible from Basics to Guru" video course

Using Modules to Manage SELinux

- The following modules can be used for managing SELinux
- `ansible.builtin.file`: sets attributes to files including SELinux context and can also create and remove files, symbolic links, and more
- `community.general.sefcontext`: manages SELinux file context in the SELinux Policy (but not on files)
- `ansible.builtin.command`: required to run **restorecon** after `sefcontext`
- Notice that `file` sets SELinux context directly on the file (like the **chcon** command), and not in the policy. DO NOT USE
- Also, consider using the RHEL system role for managing SELinux

Demo: Setting up Ansible

- On the CentOS 9.x control host
  - **sudo dnf install -y ansible-core**
  - **sudo sh –c 'echo <your.ip.addr.ess> control.example.com control >> /etc/hosts'**
  - **ssh-keygen**
  - **ssh-copy-id control**
  - **echo control >> inventory**
  - **ansible control -m ping -i inventory -u student**
- Copy the ansible.cfg file from the course Git repository
- Open sudo access: **sudo visudo**

Lesson 14: Managing SELinux with Ansible

14.2 Using the RHEL System Role to Manage SELinux

Pearson

Using RHEL System Roles

- RHEL System Roles provide an easy interface to manage complex tasks with Ansible
- After installing the rhel-system-roles.rpm package, use the example file in /usr/share/doc/rhel-system-roles/selinux as a template
- Next, run it to configure SELinux according to your needs

Lesson 14: Managing SELinux with Ansible

Lab: Using Ansible to Manage SELinux

Lab: Managing SELinux with Ansible

- Set up a minimal Ansible control node and configure an Ansible solution that allows the vsftpd service to allow anonymous user file uploads. Don't worry about anything that is not related to SELinux, it's OK to do that manually