Managing SELinux

Introduction

Course Overview and Audience

- This course is a complete Introduction to SELinux
- To attend it, no knowledge of SELinux is required
- At the end of this course, you will have seen all the important aspects of SELinux

Course Agenda (rough)

- Hour 1: introduction and working with contexts
- Hour 2: working with contexts on files and ports
- Hour 3: troubleshooting and booleans
- Hour 4: advanced stuff: creating your own policy modules
- Hour 5: very advanced stuff: making applications work with SELinux

Poll Question 1

On which distribution are you using SELinux?

- Red Hat and alike
- SUSE
- Ubuntu and alike
- Other: please specify in group chat

Poll Question 2

How would you rate your own Linux knowledge and experience?

- none
- beginner
- intermediate
- advanced
- expert
- guru

Poll Question 3

How would you rate your own SELinux knowledge and experience?

- none
- beginner
- intermediate
- advanced
- expert
- guru

Poll Question 4

Do you hold any Linux Certification?

- none
- RHCSA
- Other basic Linux (LFCS, Linux+, LPIC-1)
- RHCE
- Other Intermediate (LFCE, LPIC-2)
- Some RHCA modules
- Some LPIC-3 modules
- Completed RHCA
- Completed LPIC-3

Poll Question 5

Where are you from?

- North / Central America
- Europe
- India
- Asia (not India)
- Africa
- South America
- Australia / Pacific
- Netherlands

Managing SELinux

What is SELinux?

What is SELinux?

- SELinux is providing Mandatory Access Control to ensure that all actions are denied until specifically allowed
- Default on Red Hat Enterprise Linux and alike, Gentoo, Debian and more
- SEAndroid implements SELinux on Android
- SELinux uses the Linux kernel Linux Security Modules (LSM) framework
  - LSM is a framework that can work with SELinux, but also with other security systems
- SELinux decision is made after the DAC decision.
  - If DAC disables access to a file, SELinux is not going to enable it again
  - So if DAC doesn't allow something, you'll never see it in SELinux logs!

What is SELinux? - 2

- SELinux implements Mandatory Security. All syscalls are denied by default, unless specifically enabled
- All objects (files, ports, processes) are provided with a security label (the context)
  - User, role and type part in the context
  - Type part is the most important
- The SELinux policy contains rules where you can see which source context has access to which target context

SELinux Components

- The policy is the key component
- It defines security contexts for type enforcements
  - That means: if you use **ls –Z** on a directory, you'll find a context user, role and type and the policy knows what to do with it.
- It all comes down to a line like the following that is defined in the policy:
  - allow user_t bin_t file {read execute getattr};
    - Which states that user_t (the source) has allow to bin_t (the target) on the object class file with the permissions read execute and getattr.
- The SELinux Policy contains thousands of rules

SELinux versus AppArmor

- SELinux is becoming the de facto standard
  - Default in Red Hat and related, also in SUSE
  - wide support
  - relatively difficult configuration
  - starts from an all denied situation
- AppArmor offers an alternative
  - Used in SUSE and Ubuntu
  - default profiles available
  - relatively easy to create new configurations

Managing SELinux

Understanding SELinux Modes

SELinux States and Modes

- Enabled
  - Enforcing: fully functional
  - Permissive: not blocking anything, but logging
    - SELinux support is available in the kernel, so applications will load with all SELinux libraries and behave differently
    - Expect unexpected behavior on some occasions.
    - Log messages are in /var/log/audit/audit.log
      - Understand timestamps in audit.log using date –d @timestamp (e.g.: **date –d @1413359626**
- Disabled
  - SELinux support is not available in the kernel, so applications will load differently

Understanding Enforcing Mode

- In enforcing mode, the init system reads the configuration, locates the SELinux policy and loads the policy in memory
- If the init system is not SELinux aware, the **load_policy** command can be used to do this

Managing SELinux

Understanding the SELinux Policy

Understanding the Policy

- SELinux applies rules that are based on source and target context
- These rules are defined in the Policy
- There are multiple policies, depending on the distribution you're using

Understanding Refpolicy

- The refpolicy is a generic fully functional policy that is managed as a free software project at https://github.com/SELinuxProject/refpolicy
- Application developers provide code for the refpolicy, where after peer review it can be included
- Refpolicy is a common base for distributions that only need to make modifications to it

Policy Features

- Targeted is the normal policy, which works with context labels only
- Multi Level Security (MLS) is used to give every object a security clearance label
- Multi Category Security (MCS) is like MLS but less detailed
- Support of features is indicated by policy version – current version is 32
  - use **sestatus** to find out
- Several options need to be compiled in if desired
  - Handling of unknown permissions
  - Support for unconfined domains

Managing SELinux

Managing SELinux Access Control

Understanding SELinux Management Tasks

- While managing SELinux, you'll need to find out if an application is blocked by SELinux, and if that is the case, fix it
- To figure out if SELinux blocks an application, use the audit log (discussed later)
- If SELinux block access different options exist
  - Set the right label on the target resource
  - Set the right label on the source process
  - Use a boolean
  - Start the application as intended: daemons started from the command line will behave differently then if started through systemd
  - Modify the policy to allow the specific action

Understanding SELinux Access Control

- Type Enforcement is important in the targeted policy
  - Find out using –Z option on several commands
    - netstat –Ztulpen
    - ps –Zaux
    - ls –Z
- Access is allowed between similar source and target types
  - This prevents services from accessing user files
  - User processes are typically running as unconfined
- Note that a type that is applied to a process is called a domain

Understanding Type Enforcement

- The context type is set based on how a process is started
- Hence the same process may have different type enforcements
- See **ps -eZ | grep dbus-daemon** for instance

Managing SELinux Context

- Context on files are set in the policy and applied to the filesystem
  - **semanage fcontext –a –t httpd_sys_content_t "/web(/.*)?"**
  - **restorecon –Rv /web**

- **Do not use chcon**
  - chcon is evil
  - context applied to the inode, not to the policy
  - it won't survive an autorelabel

Finding the Right Context

- Easiest: check default objects
- Use **semanage fcontext –l** to get a list of all context settings in the policy
- Get the information from the policy, using **seinfo –t**
- Where available, use _selinux manpages (**man –k _selinux**)
  - Install from **selinux-policy-doc** package

Configuring **man -k**

- Additional man pages are available in **selinux-policy-doc**
- Alternatively, generate them manually
  - **yum install policycoreutils-devel**
  - **sepolicy manpage -a -p /usr/share/man/man8**

Applying Port Security

- Applications (**ps Zaux**) and ports have context labels also.
- Managing port context can be required

  semanage port –a –t ssh_port_t –p tcp 2022

  semanage port –m –t ssh_port_t –p tcp 443

  -m is necessary to relabel a port that has a current label applied already.

Using Booleans

- Booleans provide an easy interface to change settings in the policy
- Use semanage boolean –l for an overview of available booleans
- Use getsebool –a alternatively
- Set booleans using setsebool –P yourboolean [0|1]

Managing SELinux

Analyzing and Troubleshooting SELinux

Reading the Audit Log

- When SELinux is enabled, it will log every permission check that was denied to the audit.log file
- When evaluating access, SELinux does not go over the entire policy, but uses the Access Vector Cache in which it stores results of previous attempts
- If auditd is not enabled, log messages are stored in the kernel ring buffer: use **dmesg**
- If the SELinux troubleshooting daemon is installed and enabled, the audit daemon dispatched events to the **sedispatch** command
- **sedispatch** next sends it through D-Bus to the SELinux Troubleshooting Daemon: use **sealert** to see messages logged there

Using **sealert**

- Created to make human readable reports based on /var/log/audit/audit.log
  - Displays analysis as well as recommended action
- Matches against an event database and gives every solution a probability score
- When applied to /var/log/messages, **sealert** is used with a UUID on specific events
- Or use **sealert –a /var/log/audit/audit.log** to generate messages for all events that have happened
- Use **sealert –b** for a graphical interface

Reading Boolean Content

- **sesearch –b ftpd_anon_write –A**
  - First character in output shows state in the policy (Disabled or Enabled)
  - Second character shows if the displayed rule is enabled or disabled (True or False)
- Search rules with a specific permission
  - **sesearch –b ftpd_anon_write –p read -AC**

Reading SELinux Labels

- **sesearch –s httpd_t –t user_home_t –p read –A**
  - shows all allow rules where httpd_t gets access to user_home_t

Finding out what an application can do

- First, find the source context type set to your applications
  - **ps Zaux | grep http** would give **httpd_t**
- Use **sesearch –A | grep httpd_t** to see all allow rules and to what specifically access is allowed.
  - These are *existing* rules, not just *effective* rules!

Understanding SELinux Aware Applications

- Many applications don’t know they are running in an SELinux enabled context
- SELinux aware applications change their behavior if SELinux is enabled
- SELinux aware applications will show the **libselinux** library in the output of the **ldd** command
- Running SELinux-aware applications in permissive mode will generally not result in the application running as if SELinux were not active: using **disabled** mode is the only solution in that case

Disabling SELinux for Specific Context

- Using permissive domains allows you to define exceptions, the domain is the source context type (like httpd_t)
- Permissive domains are mainly used for development and not for production
- 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
- Don't use this in production, use unconfined instead!

Handling Unconfined Domains

- An unconfined domain is an applications 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

Managing SELinux

Customizing the SELinux Policy

Customing the SELinux Policy

- Booleans provide an easy interface to customize policy
- Modules can be used to provide basic support for SELinux functionality
  - **semodule –l** to list
  - **semodule –d** to shut off all rules for a part of the system
  - **audit2allow** is provided as easy-to-use interface to compile your own

Using **audit2allow**

- audit2allow is dangerous if you don't know what you're doing!
- Example: **grep http /var/log/audit/audit.log | audit2allow –M mypolicy**
  - Creates mypolicy.te and mypolicy.pp
  - Read mypolicy.te to see what it is doing
  - Use **semodule –i modulename.pp** to run the newly created policy module

Understanding Policy Modules - 1

- *.te files contain all rules that are compiled into the policy (these files are key)
- *.if files define how other policy modules get access to this policy modules. They provide an interface to other policy files
- *.fc files contain labeling instructions
- *.pp files are the binary policy modules

Understanding Policy Modules - 2

- After using audit2allow a *.te file and a *.pp file are created
- If refpolicy is installed, find these files in /etc/selinux/refpolicy/modules/services
- Manual changes are allowed but not recommended
  - After making modifications, run make && make install && make load

Managing SELinux

Creating Custom Rules

Creating Custom Rules

- In a modular policy, the source files of the policy modules are where you want to apply modifications
- In particular, look at the *.te files that contain what exactly has to be done
- audit2allow is a reactive interface to generate some *.te files

Understanding Custom Rules

- Default rule 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**
- Which translates into the following rule if you want to allow: **allow sshd_t http_port_t : tcp_socket { name_bind };**
- Don't do this manually, use **audit2allow** instead!
  - **grep ssh /var/log/audit/audit.log | audit2allow –M mypolicy**

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 };
```

Manually Adding Policy Files - 2

- File system resources can be defined using a .fc file
- Like the sander.fc file below:
  /opt/sander(/.*)? system_u:object_r:httpd_sys_content_t:s0

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**

Managing SELinux

Using SELinux Users and Roles

Understanding SELinux Users

- SELinux users are not the same as Linux users
- Multiple Linux users can be mapped to one SELinux user
- The current SELinux user ID cannot be changed during a session
- The policy can place additional restrictions on what an SELinux user can do
- This makes SELinux users a solution to limit what the root user can do
- By default, Linux users are mapped to unconfined_u
- Use **semanage user -l** for an overview of current users
- Users have access to SELinux Roles and the roles determine what it is a user can do
  - sysadmin_r allows use of **su** and **sudo** commands
  - xguest_r restricts command access and only allows network access through firefox

Understanding SELinux Roles

- SELinux Roles define the allowed types a process can run with
- Within a role, specific permissions are defined
- SELinux users are connected to roles to determine what they can and cannot do

Understanding SELinux Users

- Multiple SELinux users exist, below are the most common accounts:
  - **user_u**: standard non-admin users. Users mapped to user_u can NOT use **su** or **sudo**, neither SUID and SGID permissions
  - **sysadm_u**: these users can use **su** and **sudo**, but are not allowed to log in using **ssh** unless the **ssh_sysadm_login** boolean is set
  - **staff_u**: these users can use **sudo**, but NOT **su**.
  - **xguest_x**: can start Firefox in a GUI
  - **guest_u**: can only login from a console
  - Other users exist, consult the *SELinux Users and Administrators Guide* at access.redhat.com for more details, or use **semanage user -l**
- Some user-related booleans exist as well
  - **ssh_sysadm_login**: allows SSH access
  - **user_exec_content**: allows executables in home directories
  - **sysadm_exec_content**
  - **staff_exec_content**

Understanding Unconfined

- 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
- Linux users on login by default are all mapped to the unconfined_u user
  - **semanage login –l** shows current mappings

Demo: Managing SELinux Users

- **semanage user -l**
- **useradd linda; echo password | passwd --stdin linda**
- **useradd -Z sysadm_u -G wheel lisa**
- **semanage login -a -s user_u linda**
- **semanage login -l**
- login as linda (no su)
- **su -** # will be denied on RHEL7, allowed on RHEL8
- on RHEL8: **id -Z** will show that root user is in **user_u**
- From the **su -** shell on RHEL8: **useradd anna** will fail

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**
- login as anna
- **id -Z**
- **getsebool -a | grep user**
- **getsebool -a | grep sysadm**

Managing SELinux

Understanding Sensitivities

Understanding Sensitivities

- Sensitivities are an optional component that classify resources based on a security clearance
- Sensitivity values consist of a confidentiality value (starting with s) and a category value (starting with c)
- Sensitivities are used in a Multilevel Security (MLS) policy
- Multi-category Security (MCS) policies are a light version of MLS: they set the s value to 0, and just work with categories
- Use **command -Z** to show current sensitivity levels

Managing SELinux

Further Exploring the Policy

Understanding Policies

- On RHEL, 3 policies are offered
- **targeted** is the default. It controls all daemons, but processes started by users run in the **unconfined_t** domain
- **minimum** is a minimal policy, in which almost everything is running under the **unconfined_t** domain
- **MLS** allows for differentiation of security levels and is used in military environments

Exploring Context Types

- Attributes define main categories of functionality
- The domain (context) type is using attributes to get stuff done
  - Use **seinfo --type** to see all domain (context) types
- To categorize, SELinux associates attributes with types. For instance, the logfile attribute applies to all log related types.
  - Use **seinfo --attribute** for a list of attributes
  - Use **seinfo --attribute=exec_type -x** to list all context (domain) types that use the exec_type attribute
  - Or **seinfo --attribute=httpd_content_type -x** to find all context types that do something with the httpd_content_type attribute

Analyzing Policy Rules

- Rules are in the policy and determine which operations are allowed by which domain type
  - Use **sesearch -A** to display all rules
- Use filtering for more specific results:
  - **sesearch -A -s httpd_t -t httpd_config_t -c file** will show the rule(s) that allows the httpd_t source type to access files with the httpd_config_t target type
- Useful **sesearch -A** options:
  - **-s name**: source domain type or attribute
  - **-t name**: target domain type or attribute
  - **-c name:** class of the target object (use **seinfo -c** for a complete list)
  - **-p perm,perm**: list of permissions
  - **-C**: prints the name of the boolean that enables this rule

Advanced Policy Queries

- **sesearch -b ftpd_anon_write -A** shows what a Boolean is doing
- **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

Understanding dontaudit Rules

- When SELinux doesn't allow an action, but a **dontaudit** rule exists for that action, then SELinux doesn't log the denial
- This is useful to ensure that the audit log does not store logs for access violations that are expected
- For troubleshooting, it may be useful to disable dontaudit rules. Use **semodule -DB** to do so and **semodule -B** to switch them on again
- Use **sesearch --dontaudit** to get a list of all dontaudit rules

Understanding Domain Transitions

- New processes by default inherit the context type of the parent process
- Processes (daemons in particular) sometimes need to change their security context when started: you don't want httpd to run with the same context as systemd
- In these cases, transition rules apply. These are rules that are defined in the SELinux policy
- Use **sesearch -T** to display all transition rules
- DEMO: start httpd, use **pstree -Z | grep -e ^systemd -e httpd** and notice how there is a parent/child relation, but different context types

Analyzing Transition Types

- To find out which subprocesses may issue a specific type of transition, you can use **sepolicy transition**. For instance: **sepolicy transition -s sshd_t -t unconfined_t** shows a list of all sshd subprocesses types that can get to an unconfined state (and thus create a risk)
- Based on the list of context types you find here, you might want to see more details. To figure out which exact rules are involved in the transition, use the same but on the specific type you've found: **sepolicy transition -s sshd_t -t mount_t**

Analyzing File Transitions

- Normally, new files and directories inherit the context of the parent directory
- Sometimes, the policy defines that the child object must be relabeled
- Use **matchpathcon** to find the expected new file context which would be set to **restorecon**
  - **matchpathcon /var/www/myfile**
- To search file transition rules, use **sesearch -T -s httpd_t -t var_log_t -c file**
  - This will show that a new file that was generated with a source context httpd_t in a target context var_log_t, will be labeled httpd_log_t

Managing SELinux

SELinux Enabling Applications

SELinux Enabling Applications Options

- Run the application as unconfined or permissive domain – not recommended
- 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 **sepolgen** to create your own context types and apply these to the application: preferred option

Understanding File Context Type - 1

- File Context is defined by using policy modules
- Look for examples in refpolicy/policy/modules/*/*.fc (**git clone https://github.com/SELinuxProject/refpolicy**)
- A File Context Type is defined using different standard SELinux types that provide standard functionality
  - exec_type
  - file_type
  - http_content_type
  - and many more are provided as macros by SELinux
- By specifying different use types, a specific file context type is created
  - Use **seinfo –t *any_type*_t –x** for an overview
- These file context types are used in rules
  - **allow user_t var_log_t: dir { getattr search open read };**

Understanding File Context Type - 2

- /usr/share/doc/selinux-policy contains the example.te file
- In this file, the **myapp_t** and **myapp_exec_t** context types are defined, based on **domain_type** and **domain_entry_file**
- Next, allow rules are defined to provide access for source contexts to target context, including the permissions
- XXXX Compile this into a running policy module using **make –f /usr/share/selinux/devel/Makefile myapp.pp**
- According to the types, contexts can now be set to files as well as binaries
- This brings one essential component: the file contexts to be used

Analyzing Port Context

- Port Context is not fundamentally different from File Context
- Best approach is to examine a sample file as provided by refpolicy, like ssh.te (notice the complexity of this file)
- You'll find the different context types (as well as users and roles) defined here, which provides a blueprint for using your own
  - Most xxx_port_t contexts are automatically generated from the kernel/corenetwork.te.m4 macro file
  - Use grep –R 'port_t' refpolicy/policy/modules/* for an overview
- Make changes at will, and compile using **make –f /usr/share/selinux/devel/Makefile myapp.pp**

Managing SELinux

SELinux Enabling Applications:
Using **polgen**

DEMO 1: Creating Context with

- **sepolgen** can automate the work of creating .te and .fc files based on analysis of what an application is doing
- To start with, we need a dummy service
  - **git clone https://github.com/sandervanvugt/selinux**
  - **./setup-rot.sh**
- Next, run **sepolgen --application startrot13**
- This generates a startrot13.te file, some other files, as well as startrot13.sh to automatically compile the startrot13.te and related files into a policy module
  - The startrot.te file contains type definitions. Add new context type definitions as required
  - The startrot.fc shows default file context type labeling
- Analyze and modify the .te and .fc files at will, and next set up everything using the shell script that was generated (don't do this yet)

DEMO2 (c'td): Modifying an Application

- Continue on the files generated by **sepolgen --application startrot13**
- Edit the startrot13.te file, and make the following modifications:
  - In # Declarations, include **type public_content_rw_t;**
  - In # startrot13 local policy include
    allow startrot13_t public_content_rw_t:dir { getattr search open read };
    allow startrot13_t public_content_rw_t:file { getattr open read };
- Run **startrot.sh** to generate and insert the policy module. Ignore the **rpmbuild** error
- Use **seinfo –t | grep start** to verify the new context types are available

Managing SELinux

SELinux Enabling Applications:
Using **runcon**

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!!**

Selecting 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