CS-E4760: Platform Security (2023)
Exercise 2: Access Control

Section 1: Access control in general [7 points total]

Q1. Access control on computer platforms: Terminology

a. How do you characterize a reference monitor in an operating system? [1 point]

b. To what does Linux SecComp control access? How does it relate to BPF? [1 point]

Q2. Access control models

a. How does the Bell-LaPadula access control model differ from the Biba model? How do they relate to each other? [1 point]

b. What is the difference between a permission and capability (for access control)? [1 point]

Q3. SEAndroid and SELinux

a. How does the application of Domain-Type access control affect the size of the access control matrix, and why? [1 point]

b. What is the difference between using an attribute and a type in an SELinux allow rule? [1 point]

c. What purpose does the file mac_permissions.xml serve for SEAndroid policy? [1 point]

Section 2: SELinux in particular

Q4. SELinux allow rules [4 points]

The subject in SELinux is often (mostly) a process. Also a command line shell is a process in its own right. When the user launches a new program / process from a shell there will be a (subject) transition from the domain of the shell to the domain the launched program (`mc_t` in this example). The four SELinux allow rules that can make this happen are

```
allow shell_t  mc_exec:  file execute;

allow mc_t  mc_exec:  file entrypoint;

allow shell_t  mc_t:  process transition;

type_transition shell_t  mc_exec:  process mc_t;
```

In your own words, explain what happens in the operating system when a shell launches a executable (from the command line) and how each of the necessary rules above relate to the individual steps in this activity.

Q5. SELinux classes and permissions [5 points]

As mentioned in the lecture, the classes + permissions for SELinux are not a configurable thing, but really derive from an understanding of the kernel operation, and sometimes require some analysis to determine what a specific permission really gives access to, and how (and on what premises).

As an example, let us consider Linux's logging functionality (syslog), whose access control checks are in the function check_syslog_permissions in kernel/printk/printk.c in version 6 of the Linux kernel (online viewer here, or download from kernel.org).

The SELinux permission controlling access to syslog is part of the `system` class, (https://selinuxproject.org/page/ObjectClassesPerms). Syslog is also protected using Linux capabilities.

Question:

a) Which are the functions in LSM / SELinux that in order will be invoked in order to reach a decision whether syslog access is granted (notional example available in lecture slides). [4 points]

b) How are the results of the SELinux and Linux capability access control checks combined into a final access control decision? [1 points]

Q6. Emulating access control mechanisms [6 points]

Consider a Linux system with the SELinux domains

- executive_app_t
- employee_app_t

and types

- company_strategy_file_t
- worked_hours_file_t
- daily_joke_t

a) Propose a Biba access control model for the system by assigning an integrity level to each domain and type. [2 points]

b) Propose an equivalent Domain/Type-enforcement policy in the form of SELinux allow rules. [2 points]

c) SELinux allow rules alone cannot implement a low-water-mark policy. Can you propose some way of implementing it within an SELinux policy? [2 points]

Section 3: Hands-on

Deleted: (Do it yourself)

Q7. SELinux policy analysis and new policy construction [8 points]

Starting modifying policy in a SELinux environment easily can cause unexpected side-effects if done in your host machine, so our proposal is that you run the following hands-on exercise in a virtual machine or Docker image, whichever you feel comfortable with after the first lecture exercises. The instructions / help below have been tested with a Fedora Linux 37 server edition (network install) virtual machine running with virt-manager in Linux, but you are free to do the experiment with any SELinux-enabled platform of your choosing. Useful references for setting up your environment are

- https://getfedora.org/en/server/download/
- https://docs.fedoraproject.org/en-US/quick-docs/changing-selinux-states-and-modes
- https://www.thegeekdiary.com/list-of-selinux-utilities/

You will need to install some additional packages, possibly including (non-exhaustively)

- rpmdevtools
- rpmbuild
- policycoreutils-devel
- gcc
- (your preferred text editor)

(a) [2 points]: After having set up a running SELinux-enabled environment, provide screen-dumps / screenshots as evidence to show

a) What is the current status of the running SELinux (permissive or enforcing)?
b) A listing of the SELinux allow rules that pertain to Bluetooth access (note: you don't need to show every rule, but you should make sure that the command that you used to do this is visible in your screenshot, e.g. by piping the results through the head command).
c) Which target OS / version was used to complete this question.

(b) [6 points]: Write a simple SELinux policy for a password manager, that would operate by receiving a URL, and by returning a password for the URL (which is stored previously, by some other tool, no software needed for that).

From an SELinux perspective what we want is

(1) to have an application written by you (reference example below) launched into a new domain of its own,
(2) to configure the file to-be-loaded by the new application to be of a new (dedicated) type, and
(3) to ensure that only the domain of the application can read the file (since SELinux stops all other domains from accessing that new file type).

You can ignore any peculiarities of the unconfined_t domain in your answer, as it is used in e.g. the Fedora policy. Submit

(a) screenshots of the policy and context file additions for your new application/file pair,
(b) /var/log/message file snippets (AVC denial logs) that show that other binaries could not access the file in question OR similar logs produced by SELinux tools (there are a few parsers that produce more clean logs of the same result).

See below for hints.

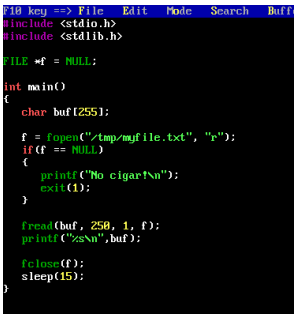
The following notes are not necessary (maybe helpful) for completing exercise 7.2.

1. At least in Fedora, you can leverage the mechanism of policy modules to complete your task, see e.g. reference below. The OS provides tools for constructing a default policy, given a compiled binary. These scripts will in fact produce a good starting point for policy configuration (although they are not complete)

https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/using_selinux/writing-a-custom-selinux-policy_using-selinux

2. To observe the currently applied set of Domains and Types, use the –Z attribute of ls and ps and other applicable Unix commands. These will show what are the currently applied domains and types. Setools will let you observe the policy itself (e.g. sesearch)

3. You can find an example C file to compile either in (1) reference above, or in the picture below (or you can make your own). In the default Fedora installation yum install gcc was needed to install a compiler. The sleep() is convenient, since it gives you time to do ps –Z on the launched program



4. The scripts in (1) will not help you with configuring the file type for the file to be read by the 'password manager'. But those rules are easily added, since the script will in fact configure a file type '_exec' for the binary file - use that as guidance.

5. To 'inherit / include' types in your *.te policy file (from mainline policy) you can use the **require { type xxx; }** pattern.

6. In order to get you application to transition into the new domain when launched, you can choose to make it into a daemon (as the RedHat example). If you do not, then the macros provided by the script are wrong. Consider using domain_auto_trans (), but even if you use that, you will have to manually add the entrypoint definition

allow yourdomain your_exec_t:file entrypoint

(note that there will be a few other allow rules needed to get the executable to read the file, but figuring them out are part of the exercise. The entrypoint rule was just especially hairy to figure out).

Alternatively, you could use the "runcon" application to run your program in a specified context.

7. The OS-generated scripts will try to construct a manpage and an rpm beyond constructing and adding the SELinux policy into the running kernel. These are not needed for this exercise (those errors can be safely ignored or code commented out from *.sh).