

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]

In operating systems architecture, a reference monitor is a secure, always-used module that enforces access control system and prevents unauthorized access to protected or confidential resources. The reference monitor verifies the nature of the request against a table of allowable access types for each process on the system, known as the access control matrix. In other words, the reference monitor mediates all inquired access and enforces the security policies. However, defining the policies is not the job of the reference monitor.

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

Linux Seccomp (secure computing mode) is a security system of the Linux kernel that controls access to system calls made by different processes. A system call is a method for a computer program to request a service or resources from the operating system's kernel on which it is running. The role of SecComp is to restrict the possible system calls that a process can make, which can efficiently reduce the number of surface attacks or prevent unwanted/ambiguous modifications from programs that are not intended for the system calls.

The BPF (Berkeley Packet Filter) is a filter of system calls. BPF is an extension to SecComp that allows the filtering of system calls using a configurable policy implemented using Berkeley Packet Filter rules. Simply put, SecComp uses BPF to filter system calls made by the processes. Their combination is widely used in security, such as sandboxing and virtualization.

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]

The Bell-LaPadula and Biba access control models are used to enforce security policies in computer OS. However, they differ in their approach to data protection and thus have different policies.

The Bell-LaPadula model is a data confidentiality model where it prioritizes the policy that unauthorized subjects are not allowed to read confidential data/objects. It defines two policies: no read-up rule and no write-down rule. The "No read up" policy believes that lower security level subjects are not allowed to read data at higher security levels, thus protecting the data

from being misused. "No write down" policy means anything produced by high-security level subjects is confidential, thus preventing them from writing to lower levels.

The Biba model is a data integrity model where it prioritizes the policy that unauthorized subjects are not allowed to write to or modify protected data. It defines two policies: no read-down rule and no write-up rule. The "No read down" policy believes that lower security level subjects are not allowed to read data at higher security levels, thus protecting integrity by preventing wrong information from moving up from lower integrity levels. "No write up" policy means lower security subjects are not allowed to write to confidential data at higher levels, which protects its integrity.

Relationship between these two access control models: they aim to protect data from unwanted actions by unauthorized subjects and can be used in complement to enhance security, where the Biba model protects data integrity, and the Bell-Lapadula model protects data confidentiality

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

A permission is a specific action that a subject is allowed to perform towards an object, such as read, write and execute permissions in Linux system. On the contrary, a capability is usually a physical token or secret information that proves that the subjects are what they claim to be. In other words, a capability is a tool required from the subjects to prove they can perform a permitted action on the restricted objects.

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]

The Domain-Type access control model categorizes subjects and objects into groups (domains) so that the access control matrix can be kept at reasonable complexity with thousands and millions of subjects and objects (from lecture slides). This model's permissions are based on the domains to which subjects and objects belong.

In the access control matrix, the rows are the subjects, and the columns are the objects. Therefore, its size is determined by the formula "objects x subjects". Because of this, if the number of objects is, for example, one million, the number of subjects is thousands, and there are many permissions, this matrix size could grow too large, which is inefficient lookup.

The application of domain-type access thus aims to reduce the size of the access control matrix. This is because the access control matrix is not required to enumerate all possible combinations of permissions, subjects, and objects. Instead, this access control model only needs to specify the permissions for each domain and the types of objects, which is much smaller than the product of subjects and objects in the traditional access control matrix. For example, if there are 10 domains and 3 types of objects, the access control matrix managed would have at most $10 \times 3 = 30$ cells, which is significantly smaller than the AC matrix in the usual approach.

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

An SELinux (Security-Enhanced Linux) policy is a collection of SELinux allow rules. In SELinux, allow rules grant the permissions for different processes to perform actions on different objects. Thus, allow rules are basically always between 2 entities: a domain (processes) and a type (objects). These rules are based on either attributes or types.

The type in SELinux is a domain for subjects/processes or a type for objects/files. Attributes are groups of types (or domains). Rules can also be defined using attributes. By this definition, types in SELinux allow rule specifies the permissions based on the types of the subject/object. For example, the type "all can read" can be applied to an object, and everyone in the system can read that file based on its type.

On the other hand, attributes in SELinux specify the permissions for a specific subject/object based on its attribute values, such as permissions for reading, writing, and executing. For example, the file may have the attribute "anyone in group A can write, " allowing all group A members to edit. However, this attribute is not necessarily available in other files. This provides fine-grained control over processes accessing the objects, but it simultaneously can complicate the rules and add more complexity to security analysis.

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

The SEAndroid is the security model for the Android OS, which is based on the SELinux. The file *mac_permissions.xml* serves as an SELinux policy file for SEAndroid to enforce mandatory access control (MAC) over all processes and security policies on Android devices, even processes running with root/superuser privileges (Linux capabilities). The file contains the security policies that define the permissions for different subjects (users or processes) to access different objects (resources or files) on the Android device.

Specifically, *mac_permissions.xml* assigns a *seinfo* tag to apps based on their signature and, optionally, their package name. The *seinfo* tag can then be used as a key in the *seapp_contexts* file to assign a specific label to all apps with that *seinfo* tag. This configuration is read by *system_server* during startup.

More documentation at <https://source.android.com/docs/security/features/selinux>

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.

In Linux OS, a shell is a loop that waits for user input to look for in the PATH of Linux. When the shell finds the program's directory, it will create a process, which is an instance of a program that needs execution. The shell itself is a parent process, and the execution of the program would be a child process. A process can be created with a system called `fork()`.

The `fork()` system call will give the child process a PID or process ID number of 0, which helps differentiate it from the parent process, which will be given a non-zero value PID. The parent process will wait until the child process has been executed and wholly terminated, and to ensure this, the `wait()` system call can be used.

When a user launches a new program or process from *shell_t*, the subject transitions from the shell's domain to the domain of the launched program *mc_t*. This transition is controlled sequentially by the four SELinux allow rules above to ensure a secure transition. They are: allowing the shell to execute the executable domain, declaring the entrypoint of the executable domain, allowing transitioning from the shell to the child program, and finally allowing the transitioning to execute the program.

The syntax of the allow rule is as follows:

ALLOW [domain] [type] : [class] {[allowed permissions]}

where the domain has subjects/processes, type is the objects, class is the resource class, and allowed class-specific permissions that are allowed by this rule.

1. The first rule, "allow shell_t mc_exec: file execute":

It allows the command line shell_t to execute files of the type "mc_exec". This rule is required to ensure that the shell can execute the *mc_exec*.

2. The second rule, "allow mc_t mc_exec: file entrypoint":

It allows processes in the "mc_t" domain to access the entrypoint of files in the "mc_exec" domain. By itself, executing a file with a specific label is insufficient to allow a domain transition because SELinux does not parse contexts. We can ask for what domains the mc_t type is an entrypoint for.

3. The third rule, "allow shell_t mc_t: process transition":

It allows processes in the "shell_t" domain to transition to the "mc_t" domain. This rule allows the shell terminal to transition to the domain of the new program mc_t

4. The fourth rule, "type_transition shell_t mc_exec: process mc_t"

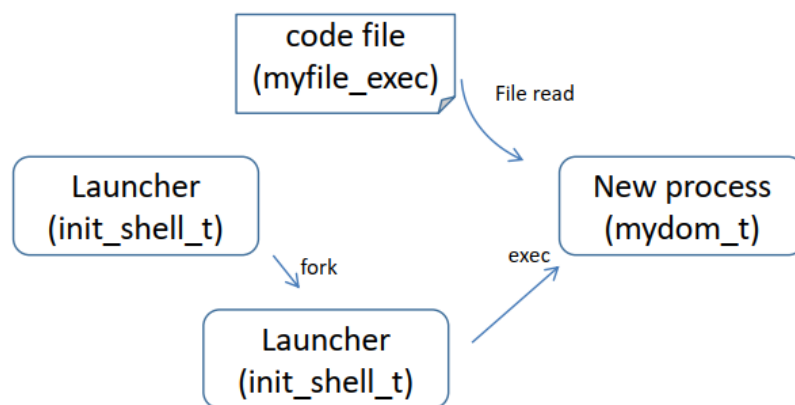
It specifies a type transition from the "shell_t" domain to the "mc_exec" domain for process mc_t. This means that when a shell_t process executes a file with context mt_exec, the resulting process should run in the mc_t context. This last rule finally transitions to the child process of mc_t and executes it.

Documentations on SELinux rules:

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

https://wiki.gentoo.org/wiki/SELinux/Tutorials/How_does_a_process_get_into_a_certain_context

Lecture slide



Intent ("what we want to happen")

```
type_transition init_shell_t myfile_exec: process mydom_t;
```

File execution right

```
allow init_shell_t myfile_exec: file execute;
```

File type is an entrypoint into a domain ("object firewall")

```
allow mydom_t myfile_exec: file entrypoint;
```

Process type needs transition right into a domain ("subject firewall")

```
allow init_shell_t mydomain_t: process transition;
```

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).

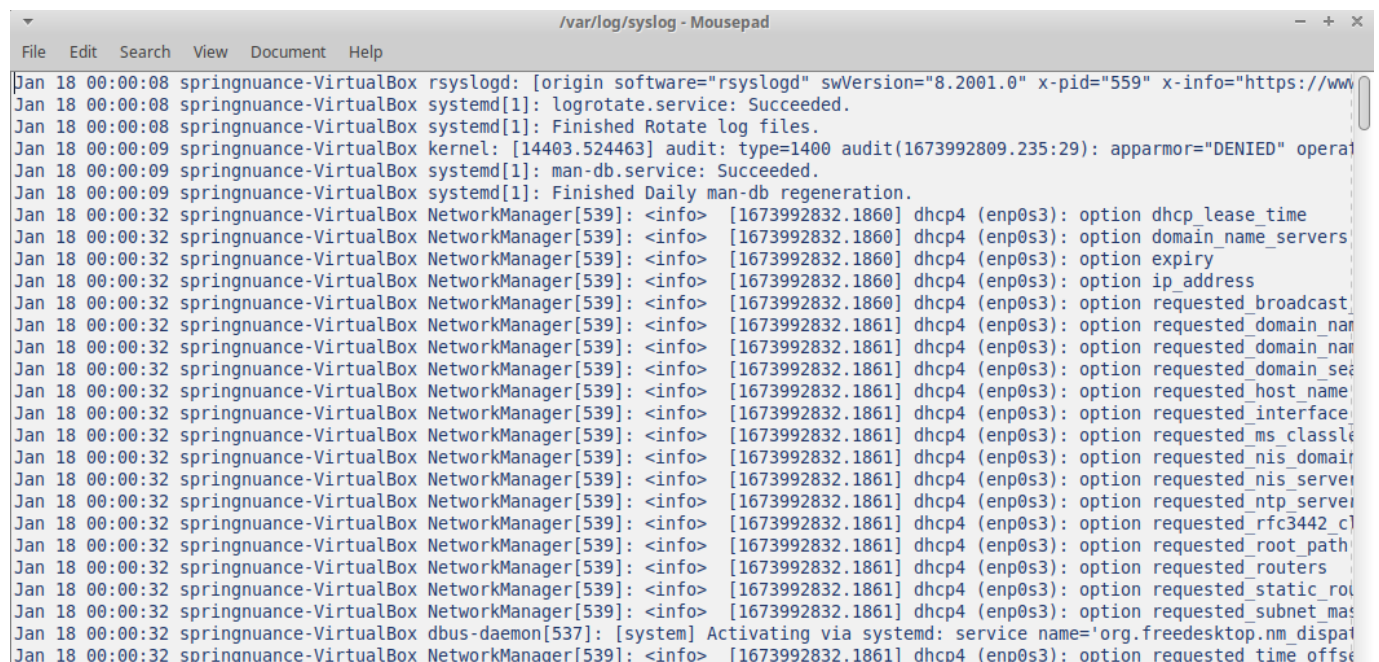
The SELinux permission controlling access to syslog is part of the system class, (<https://selinuxproject.org/page/ObjectClassesPerms>). Syslog is also protected using Linux capabilities.

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]

An object is an OS primitive, which is not a policy issue. An object, therefore, belongs to one or more classes predefined by the system properly. A class can be e.g., a file, socket, and character device. The SELinux security policy is consulted to determine whether the access request is allowed or denied (lecture slides).

One of the most important logs contained within `/var/log` is syslog. This particular log file logs everything except auth-related messages. Syslog is a standard for message logging, which has been the standard logging mechanism on Linux/Unix systems.



```
Jan 18 00:00:08 springnuance-VirtualBox rsyslogd: [origin software="rsyslogd" swVersion="8.2001.0" x-pid="559" x-info="https://www.rsyslog.com/doc/v8-release-notes.html"]
Jan 18 00:00:08 springnuance-VirtualBox systemd[1]: logrotate.service: Succeeded.
Jan 18 00:00:08 springnuance-VirtualBox systemd[1]: Finished Rotate log files.
Jan 18 00:00:09 springnuance-VirtualBox kernel: [14403.524463] audit: type=1400 audit(1673992809.235:29): apparmor="DENIED" operation="open" profile="systemd" name="/var/log/syslog" pid=559
Jan 18 00:00:09 springnuance-VirtualBox systemd[1]: man-db.service: Succeeded.
Jan 18 00:00:09 springnuance-VirtualBox systemd[1]: Finished Daily man-db regeneration.
Jan 18 00:00:32 springnuance-VirtualBox NetworkManager[539]: <info> [1673992832.1860] dhcp4 (enp0s3): option dhcp_lease_time
Jan 18 00:00:32 springnuance-VirtualBox NetworkManager[539]: <info> [1673992832.1860] dhcp4 (enp0s3): option domain_name_servers
Jan 18 00:00:32 springnuance-VirtualBox NetworkManager[539]: <info> [1673992832.1860] dhcp4 (enp0s3): option expiry
Jan 18 00:00:32 springnuance-VirtualBox NetworkManager[539]: <info> [1673992832.1860] dhcp4 (enp0s3): option ip_address
Jan 18 00:00:32 springnuance-VirtualBox NetworkManager[539]: <info> [1673992832.1860] dhcp4 (enp0s3): option requested_broadcast_address
Jan 18 00:00:32 springnuance-VirtualBox NetworkManager[539]: <info> [1673992832.1861] dhcp4 (enp0s3): option requested_domain_name_servers
Jan 18 00:00:32 springnuance-VirtualBox NetworkManager[539]: <info> [1673992832.1861] dhcp4 (enp0s3): option requested_interface
Jan 18 00:00:32 springnuance-VirtualBox NetworkManager[539]: <info> [1673992832.1861] dhcp4 (enp0s3): option requested_ip_address
Jan 18 00:00:32 springnuance-VirtualBox NetworkManager[539]: <info> [1673992832.1861] dhcp4 (enp0s3): option requested_ip_netmask
Jan 18 00:00:32 springnuance-VirtualBox NetworkManager[539]: <info> [1673992832.1861] dhcp4 (enp0s3): option requested_host_name
Jan 18 00:00:32 springnuance-VirtualBox NetworkManager[539]: <info> [1673992832.1861] dhcp4 (enp0s3): option requested_nis_domain_name
Jan 18 00:00:32 springnuance-VirtualBox NetworkManager[539]: <info> [1673992832.1861] dhcp4 (enp0s3): option requested_nis_server
Jan 18 00:00:32 springnuance-VirtualBox NetworkManager[539]: <info> [1673992832.1861] dhcp4 (enp0s3): option requested_ntp_server
Jan 18 00:00:32 springnuance-VirtualBox NetworkManager[539]: <info> [1673992832.1861] dhcp4 (enp0s3): option requested_rfc3442_client_id
Jan 18 00:00:32 springnuance-VirtualBox NetworkManager[539]: <info> [1673992832.1861] dhcp4 (enp0s3): option requested_root_path
Jan 18 00:00:32 springnuance-VirtualBox NetworkManager[539]: <info> [1673992832.1861] dhcp4 (enp0s3): option requested_routers
Jan 18 00:00:32 springnuance-VirtualBox NetworkManager[539]: <info> [1673992832.1861] dhcp4 (enp0s3): option requested_static_routes
Jan 18 00:00:32 springnuance-VirtualBox NetworkManager[539]: <info> [1673992832.1861] dhcp4 (enp0s3): option requested_subnet_mask
Jan 18 00:00:32 springnuance-VirtualBox dbus-daemon[537]: [system] Activating via systemd: service name='org.freedesktop.nm_dispatcher' already running / started
Jan 18 00:00:32 springnuance-VirtualBox NetworkManager[539]: <info> [1673992832.1861] dhcp4 (enp0s3): option requested_time_offset
```

From the documentations:

capability2

Permission	Description	Kernel Version/Capability
mac_override	Override MAC restrictions - Ignored by SELinux	2.6.25+
mac_admin	Change MAC configuration - For SELinux, get/set raw security context values unknown to the current policy.	2.6.25+
syslog	Configure kernel syslog subsystem	
wake_alarm	Trigger something that will wake the system	
block_suspend	Prevent system suspends	

system

Permission	Description	Kernel Version/Capability
ipc_info	Get info for an ipc socket.	
syslog_mod	Perform syslog operation other than syslog_read or console logging.	
syslog_read	Perform syslog read.	
syslog_console	Perform syslog console.	

So syslog is part of the the system class.

The example in the lecture slide

Class/ Permission illustration

Inside fs/open.c (sys_open syscall)

```
static int do_dentry_open(struct file *f, int (*open)(struct inode *, struct file *),
                        const struct cred *cred)
```

...

```
error = security_file_open(f, cred);    ← class 'file', permission 'open'
if (error) goto cleanup_all;
```

Inside security.c (LSM)

```
int security_file_open(struct file *file, const struct cred *cred) {
    int ret;
    ret = security_ops->file_open(file, cred);
    if (ret) return ret;
    return fsnotify_perm(file, MAY_OPEN);
}
```

Inside selinux/hooks.c

```
.file_open = selinux_file_open,
```

The checking syslog permission source code

```
static int check_syslog_permissions(int type, int source)
{
    /*
     * If this is from /proc/kmsg and we've already opened it, then we've
     * already done the capabilities checks at open time.
     */
    if (source == SYSLOG_FROM_PROC && type != SYSLOG_ACTION_OPEN)
        goto ok;

    if (syslog_action_restricted(type)) {
        if (capable(CAP_SYSLOG))
            goto ok;

        /*
         * For historical reasons, accept CAP_SYS_ADMIN too, with
         * a warning.
         */
        if (capable(CAP_SYS_ADMIN)) {
            pr_warn_once("%s (%d): Attempt to access syslog with "
                        "CAP_SYS_ADMIN but no CAP_SYSLOG "
                        "(deprecated).\n",
                        current->comm, task_pid_nr(current));
            goto ok;
        }
        return -EPERM;
    }
ok:
    return security_syslog(type);
}

static int devkmsg_open(struct inode *inode, struct file *file)
{
    struct devkmsg_user *user;
    int err;

    if (devkmsg_log & DEVKMSG_LOG_MASK_OFF)
        return -EPERM;

    /* write-only does not need any file context */
    if ((file->f_flags & O_ACCMODE) != O_WRONLY) {
        err = check_syslog_permissions(SYSLOG_ACTION_READ_ALL,
                                       SYSLOG_FROM_READER);

        if (err)
            return err;
    }
    return 0;
}
```


The functions in LSM / SELinux in order that will be invoked in order to reach a decision on whether syslog access is granted are:

1. `devkmsg_open`

This function invokes the process of accessing the syslog file. It will check the capabilities and permissions in the policies in SELinux/LSM.

2. `check_syslog_permissions`

This function is invoked by `devkmsg_open` to check for permissions to access the syslog file

3. `syslog_action_restricted`

This function is invoked by `check_syslog_permissions` if the applied action of the process is permitted towards the syslog

4. `capable`

If the action is allowed for the syslog, the `check_syslog_permissions` continues to check if the process has the capability to perform this action

5. `security_syslog`

Finally, if everything is cleared, access to the syslog is granted by `security_syslog`. This returns to the original `devkmsg_open` function

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

The access control decision-making process in Linux is finalized by combining the outcomes of two security components: SELinux and Linux capabilities. When a process requests to perform an action on the objects, the Linux kernel checks both SELinux and Linux capabilities to determine whether the access should be granted or denied.

Like the example above, the kernel first checks the SELinux policy to see if access is allowed or denied based on the security context. If access is granted, it proceeds to check the capabilities of the process. If it has, access is granted. In summary, the final allowance decision is the truth value "AND" of the results of both checks: If either SELinux or capabilities deny the access, the access is denied. It is only granted if both conditions are satisfied.

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`
- `dailyjoke_t`

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

According to the naming semantics, executive levels should have higher clearance levels than employees. For the types, company strategy is important data that should be confidential (such as being protected from competitors). So it is logical that only the executives can access it, but the employees are not allowed to write. For the working hours, it is a schedule that everyone should know (such as working shifts). This file, therefore, should be able to be read by everyone, but only the managers are allowed to change it. Finally, daily joke sounds like something informal that employees exchange and the executives do not need it. From these deductions, the Biba access control model for the system is:

For the domains (subjects):

- *executive_app_t*: High integrity level

Naturally, the executives are superior, so they should be able to modify important data

- *employee_app_t*: Low integrity level

Employees are not allowed to edit essential or confidential files in the organization.

For the types (objects):

- *company_strategy_file_t*: High integrity level

Only the executives are allowed to edit this type, as it is confidential data. However, the employees can still read it because reading up is allowed in the Biba model.

- *worked_hours_file_t*: High integrity level.

Everyone can read this file type. However, employees should not be able to tamper with it. They can only inform the managers of the working hours, and the managers will be the ones who edit it. This prevents employees from changing the schedule of others due to personal interests

- *dailyjoke_t*: Low integrity level

This can be edited by the employees for personal entertainment. According to the Biba model, reading down is not allowed, so the executives do not need to read this file.

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

Based on the Biba model above, the allow rules of SELinux in the Domain/Type enforcement policy are proposed as:

- allow *executive_app_t* to read and write *company_strategy_file_t*

- allow *executive_app_t* to read and write *worked_hours_file_t*

- allow *executive_app_t* to write *dailyjoke_t*

- allow *employee_app_t* to read *company_strategy_file_t* and *worked_hours_file_t*

- allow *employee_app_t* to read and write *dailyjoke_t*

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]

The two low-watermark policy that is not reinforced in part (b) are:

- deny *executive_app_t* to read *dailyjoke_t*
- deny *employee_app_t* to write *company_strategy_file_t* and *worked_hours_file_t*

A low-water-mark policy can be implemented within an SELinux policy by using the SELinux Multilevel Security (MLS) model. This model in SELinux is a framework mechanism to support data confidentiality and integrity (Bell LaPadula, Biba)

Multilevel security is a security policy that allows the classification of data and users based on a system of hierarchical security levels combined with a system of non-hierarchical security categories. A multilevel-secure security policy has two primary goals. First, the controls must prevent unauthorized individuals from accessing information at a higher classification than their authorization. Second, the controls must prevent individuals from declassifying information. This can be achieved by defining the SELinux user roles by low-watermark policy and assigning users to their respective roles.

Section 3: Hands-on

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

First, I enable

1) What is the current status of the running SELinux (permissive or enforcing)?

Before enabling SELinux, this is the status of SELinux on my virtual machine, which is disabled

```
springnuance@springnuance-VirtualBox:~/Desktop$ sudo sestatus
SELinux status: disabled
```

I enabled it using the instructions at

<https://www.linode.com/docs/guides/how-to-install-selinux-on-debian-10/>

After rebooting, SELinux is enabled, which is in permissive mode

```
springnuance@springnuance-VirtualBox:~/Desktop$ sudo sestatus
SELinux status: enabled
SELinuxfs mount: /sys/fs/selinux
SELinux root directory: /etc/selinux
Loaded policy name: default
Current mode: permissive
Mode from config file: permissive
Policy MLS status: enabled
Policy deny_unknown status: allowed
Memory protection checking: requested (insecure)
Max kernel policy version: 33
springnuance@springnuance-VirtualBox:~/Desktop$ s
```

To set enforcing mode, edit with `sudo vim /etc/selinux/config`, and change `SELINUX=permissive` into `SELINUX=enforcing`. Additionally, I run `$sudo setenforce 1`

```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
# enforcing - SELinux security policy is enforced.
# permissive - SELinux prints warnings instead of enforcing.
# disabled - No SELinux policy is loaded.
SELINUX=enforcing
# SELINUXTYPE= can take one of these two values:
# default - equivalent to the old strict and targeted policies
# mls      - Multi-Level Security (for military and educational use)
# src      - Custom policy built from source
SELINUXTYPE=default

# SETLOCALDEFS= Check local definition changes
SETLOCALDEFS=0
```

```
springnuance@springnuance-VirtualBox:~/Desktop$ sudo sestatus
SELinux status: enabled
SELinuxfs mount: /sys/fs/selinux
SELinux root directory: /etc/selinux
Loaded policy name: default
Current mode: enforcing
Mode from config file: enforcing
Policy MLS status: enabled
Policy deny_unknown status: allowed
Memory protection checking: requested (insecure)
Max kernel policy version: 33
```

2) 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).

I use the sestatus tool to find SELinux allow rules that pertain to Bluetooth access in Linux.

Documentations for sestatus command: <https://linux.die.net/man/1/sestatus>

The command to search for the allow rules of the bluetooth type is:

```
$ sestatus --allow -t bluetooth_t
```

This will search for all allow rules with a source type of bluetooth_t and a class of bluetooth, and print the results to the console.

```
springnuance@springnuance-VirtualBox:~/Desktop$ sestatus --allow -t bluetooth_t
allow NetworkManager_t bluetooth_t:dbus send_msg;
allow NetworkManager_t domain:dir { getattr ioctl lock open read search };
allow NetworkManager_t domain:file { getattr ioctl lock open read };
allow NetworkManager_t domain:lnk_file { getattr read };
allow acpid_t domain:dir { getattr ioctl lock open read search };
allow acpid_t domain:file { getattr ioctl lock open read };
allow acpid_t domain:lnk_file { getattr read };
allow apt_t domain:process getattr;
allow auditadm_t bluetooth_t:dbus send_msg;
allow auditadm_t daemon:association recvfrom;
allow auditadm_t daemon:peer recv;
allow auditadm_t daemon:tcp_socket recvfrom;
allow auditadm_t daemon:udp_socket recvfrom;
allow auditadm_t domain:process sigkill;
allow auditctl_t domain:dir { getattr ioctl lock open read search };
allow auditctl_t domain:file { getattr ioctl lock open read };
allow auditctl_t domain:lnk_file { getattr read };
allow bluetooth_helper_t bluetooth_t:dbus send_msg;
allow bluetooth_helper_t bluetooth_t:socket { read write };
allow bluetooth_helper_t domain:dir { getattr ioctl lock open read search };
allow bluetooth_helper_t domain:file { getattr ioctl lock open read };
allow bluetooth_helper_t domain:lnk_file { getattr read };
allow bluetooth_t bluetooth_t:association sendto;
```

3) Which target OS/version was used to complete this question.

The target OS/version used to complete this question:

Host machine: Windows

Virtual machine OS: Ubuntu version 18.04 Debian

(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).

```
springnuance@springnuance-VirtualBox:~/Desktop/SELinux$ vi mydaemon.service
springnuance@springnuance-VirtualBox:~/Desktop/SELinux$ cp mydaemon /usr/local/bin/
cp: cannot create regular file '/usr/local/bin/mydaemon': Permission denied
springnuance@springnuance-VirtualBox:~/Desktop/SELinux$ sudo cp mydaemon /usr/local/bin/
[sudo] password for springnuance:
springnuance@springnuance-VirtualBox:~/Desktop/SELinux$ sudo cp mydaemon.service /usr/lib/systemd/system
cp: cannot stat 'mydaemon.service': No such file or directory
springnuance@springnuance-VirtualBox:~/Desktop/SELinux$ sudo cp mydaemon.service /usr/lib/systemd/system
springnuance@springnuance-VirtualBox:~/Desktop/SELinux$ systemctl start mydaemon
springnuance@springnuance-VirtualBox:~/Desktop/SELinux$ systemctl status mydaemon
● mydaemon.service - Simple testing daemon
   Loaded: loaded (/usr/lib/systemd/system/mydaemon.service; disabled; vendor preset: enabled)
   Active: active (running) since Sat 2023-01-21 13:43:35 EET; 14s ago
     Main PID: 3878 (mydaemon)
       Tasks: 1 (limit: 5454)
      Memory: 152.0K
      CGroup: /system.slice/mydaemon.service
              └─3878 /usr/local/bin/mydaemon

tammi 21 13:43:35 springnuance-VirtualBox systemd[1]: Started Simple testing daemon.
```

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).

This is how I set up the application:

First, I compose the password_manager.c. This password manager works like this:

0. It has a memory of passwords using a text file.

1. If an URL is asked for the first time (this URL is not in the text file), it will generate a random password for it and save the password

2. When this URL is asked again, it will check the text file, and it sees that this URL has been asked before, so it returns the password from the text file.

Now, I can compile and run the application with

```
$ gcc -o password_manager password_manager.c
```

```
$ ./password_manager https://helloworld.com
```

```
springnuance@springnuance-VirtualBox:~/Desktop/SELinux$ ./password_manager https://helloworld.com
Password for https://helloworld.com: oiqprisrlddbdggf
```

Now I moved on to SELinux configurations

Step 1: Install and start the password_manager

```
$ sudo cp password_manager /usr/local/bin/  
$ sudo cp password_manager.service /usr/lib/systemd/system  
$ systemctl start password_manager  
$ systemctl status password_manager
```

```
springnuance@springnuance-VirtualBox:~/Desktop/SELinux$ gcc -o password_manager password_manager.c  
springnuance@springnuance-VirtualBox:~/Desktop/SELinux$ sudo cp password_manager /usr/local/bin/  
[sudo] password for springnuance:  
springnuance@springnuance-VirtualBox:~/Desktop/SELinux$ sudo cp password_manager.service /usr/lib/systemd/system  
springnuance@springnuance-VirtualBox:~/Desktop/SELinux$ systemctl start password_manager  
springnuance@springnuance-VirtualBox:~/Desktop/SELinux$ systemctl status password_manager  
● password_manager.service - Password manager  
   Loaded: loaded (/lib/systemd/system/password_manager.service; disabled; vendor preset: enabled)  
   Active: active (running) since Sat 2023-01-21 17:26:04 EET; 6s ago  
 Main PID: 10325 (password_manager)  
    Tasks: 1 (limit: 5454)  
  Memory: 152.0K  
   CGroup: /system.slice/password_manager.service  
           └─10325 /usr/local/bin/password_manager https://helloworld.com  
  
tammi 21 17:26:04 springnuance-VirtualBox systemd[1]: Started Password manager.
```

Step 2: Check that the new password manager is not confined by SELinux

```
$ ps -efZ | grep password_manager
```

```
springnuance@springnuance-VirtualBox:~/Desktop/SELinux$ ps -efZ | grep password_manager  
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 springn+ 10387 10134 0 17:29 pts/3  
00:00:00 grep --color=auto password_manager
```

Step 3: Generate a custom policy for the password_manager

```
$ sepolity generate --init /usr/local/bin/password_manager
```

```
springnuance@springnuance-VirtualBox:~/Desktop/SELinux$ sepolity generate --init /usr/local/bin/password_manager  
Failed to retrieve rpm info for selinux-policy  
Created the following files:  
/home/springnuance/Desktop/SELinux/password_manager.te # Type Enforcement file  
/home/springnuance/Desktop/SELinux/password_manager.if # Interface file  
/home/springnuance/Desktop/SELinux/password_manager.fc # File Contexts file  
/home/springnuance/Desktop/SELinux/password_manager_selinux.spec # Spec file  
/home/springnuance/Desktop/SELinux/password_manager.sh # Setup Script
```

Step 4: Rebuild the system policy with the new policy module using the setup script created by the previous command

```
$ sudo ./password_manager.sh
```

```
springnuance@springnuance-VirtualBox:~/Desktop/SELinux$ sudo ./password_manager.sh  
Building and Loading Policy  
+ make -f /usr/share/selinux/devel/Makefile password_manager.pp  
Compiling default password_manager module  
Creating default password_manager.pp policy package  
rm tmp/password_manager.mod tmp/password_manager.mod.fc  
+ /usr/sbin/semodule -i password_manager.pp  
+ sepolity manpage -p . -d password_manager_t  
./password_manager_selinux.8  
+ /sbin/restorecon -F -R -v /usr/local/bin/password_manager
```


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,

```
springnuance@springnuance-VirtualBox:~/Desktop/SELinux$ sudo make -f /usr/share/selinux/devel/Makefile
[sudo] password for springnuance:
Compiling default password_manager module
Creating default password_manager.pp policy package
rm tmp/password_manager.mod tmp/password_manager.mod.fc
```

Policy file

```
password_manager.te X
password_manager.te
1  policy_module(password_manager, 1.0.0)
2
3  require {
4      type unconfined_t;
5      class file { read write };
6  }
7
8  # Declarations
9  S
10 type password_manager_t;
11 type password_manager_exec_t;
12 init_daemon_domain(password_manager_t, password_manager_exec_t)
13
14 permissive password_manager_t;
15
16 # password_manager local policy
17
18 # Allow the password manager to read the password file
19 allow password_manager_exec_t password_manager_t:file read;
20
21 # Deny all other domains access to the password file
22 dontaudit unconfined_t password_manager_t:file read;
23
24 allow password_manager_t self:fifo_file rw fifo_file_perms;
25 allow password_manager_t self:unix_stream_socket create_stream_socket_perms;
26
27 # Define the password file as the entrypoint for the executing domain
28 allow password_manager_t password_manager_exec_t: file entrypoint;
29
30 domain_use_interactive_fds(password_manager_t)
31
32 files_read_etc_files(password_manager_t)
33
34 miscfiles_read_localization(password_manager_t)
```

Context file

```
password_manager.fc X
password_manager.fc
1  /usr/local/bin/password_manager    --  gen_context(system_u:object_r:password_manager_exec_t,s0)
```


(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).

When I run with the correct context, the password manager outputs as normal

```
springnuance@springnuance-VirtualBox:~/Desktop/SELinux$ runcon system_u:object_r:password_manager_exec_t:s0 /usr/local/bin/password_manager https://helloworld.com
Password for https://helloworld.com: oiqpriszlddbdf
```

However, I fail to produce the error that is required in this part. To be honest I am lost at this point. Im sorry as I was overwhelmed with many information about SELinux, so this is my best attempt at this exercise. I hope you can give me detailed feedback on how to do this exercise correctly, as I still vaguely understand SELinux, like what should I write in the context file .fc and policy file .te, and what are actually the application/file pair that you refer to. Is it two different domains, file_t and file_exec_t (subject and object), or two different types of files that interact with each other? Thank you and I look forward to your help.

Additionally, I am using Debian so maybe there is something that can be done on Fedora but cannot be produced on Debian. So I am curious to know if it is absolutely required to use Fedora for this SELinux exercise.

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`)

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

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

```
F10 key => File Edit Mode Search Buff
#include <stdio.h>
#include <stdlib.h>

FILE *f = NULL;

int main()
{
    char buf[255];

    f = fopen("/tmp/myfile.txt", "r");
    if(f == NULL)
    {
        printf("No cigar!\n");
        exit(1);
    }

    fread(buf, 250, 1, f);
    printf("%s\n", buf);

    fclose(f);
    sleep(15);
}
```

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 your 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).