Q1

a.

A reference monitor is a system which grants permission in an operating system. Mainly, it necesites two traits:

- It is not possible to receive certain permissions without the RM granting it, speaking, it is not possible to circumvent the RM's decision
- The reference monitor needs to be tamper-proof

b.

SecComp is able to filter system calls after it is activated. After activation only *exit(), sigreturn(), read() and write()* to already-open file descriptors are allowed.

seccomp-bpf is an extension allowing custom filtering of syscalls.

Q2

a.

The Bell-LaPadula model has a completely different goal than the Biba model. The Bell-LaPadula model enforces a "no write-down, no read-up" policy in order to protect confidentiality.

On the other hand, the biba model protects integrity by enforcing a "no write-up, no read-down" policy.

b.

The main difference is that a permission is assigned (see also MAC) and a capabilty can be forwarded.

Q3

a.

The size of the AC-matrix is decreased since multiple subjects/objects can be merged into a domain and do not need its own columns/rows.

b.

A type is used to summarize multiple system ressources. These types can then be grouped into attributes. A type can have 0, 1 or more attributes. An attribute can then be used just like a type in rules.

c.

This file saves the configuration for SEAndroid on Android.

Q4

First, the executable is executed by forking the shell and the child running the new process. This is done by the child using the *exec()* system call to replace its own process image with the new one.

By default a new process inherits the context of the parent process. Clearly, this is not always wanted, hence, selinux has transitions allowing a new process to switch to a new context. Acording to [https://wiki.gentoo.org/wiki/SELinux/Tutorials/How_does_a_process_get_into_a_certain_context] the following three requierements have to be fulfilled:

1. The origin domain has execute permission on the file
2. The file context itself is identified as an entry point for the target domain
3. The origin domain is allowed to transition to the target domain

Now, inspecting the given four selinux rules we can identify their purposes.

1. This rule satisfies the first requieremnt by giving the shell*t type execute permissions for the executable file with _mc_exec* attribute(?).
2. Continuing, the *mc_exec* attribute of the binary file has to be explicitly mapped to the mc_t of the to-be-spawned process.
3. Finally the last requierement is fulfilled by giving the shell_t type transition permission to the mc_t type.
4. Lastly, the transition itself is applied.

Q5

a.

Inside printk/printk.c

```
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);
}
```

Inside security.c

```
int security_syslog(int type)
{
    return call_int_hook(syslog, 0, type);
}
```

Inside selinux/hooks.c

```
selinux_syslog(int type)
```

b.

If either one of SELinux and Linux capabilty AC fails, the final result is also fail. As we can see in the previous exercise, there are first leightweight checks for capabilites. If this check fails, the access is denied. On the other hand, on success, the lsm modules (inlcuding selinux) are invoked to check for permission.

Q6

a.

We start by considering two integrity levels high and low.

executive_app_t -> high : An executive should be able to change any information as they are the ones with the most influence.
employee_app_t -> low : An emloyee should be able to modify data in his own scope but these rights should genereally be restricted.

company_strategy_file_t -> high: Clearly, changes to this type should onyl be allowed by executives.
worked_hours_file_t -> low: An employee and an executive need to be able to edit their amount of worked hours. However, this file shows the limitations of a simple MLS scheme with Biba. The low setting allows the employee to edit the amount of worked hours of executives and vice versa. Using a high label would still

allow an manger to edit the amount of worked hours of an employee while he would not be even able to enter those himself. Here, using a fine grained AC like SELinux would allow us to solve these problems.
daily_joke_t -> low: This information is not relvant in the grand scheme and may be edited by anyone

b.

allow employee_app_t company_strategy_file_t: file {read}
allow employee_app_t worked_hours_file_t: file {read write}
allow employee_app_t daily_joke_t: file {read write}
allow executive_app_t company_strategy_file_t: file {read write}
allow executive_app_t worked_hours_file_t: file {read write}
allow executive_app_t daily_joke_t: file {read write}

c.

This may be acheived with *type_change* rules. The downside would be that for every scenario/combination where low-watermark policy changes the type, we would need an explicit rule.

Q7

a.

a.

```
sestatus

SELinux status:                 enabled
SELinuxfs mount:                /sys/fs/selinux
SELinux root directory:         /etc/selinux
Loaded policy name:             targeted
Current mode:                   enforcing
Mode from config file:          enforcing
Policy MLS status:              enabled
Policy deny_unknown status:     allowed
Memory protection checking:     actual (secure)
Max kernel policy version:      33
```

As we can see the current mode is **enforcing**.

b.

According to bluetooth_selinux(8) the follwing process types are defined for bluetooth
bluetooth_helper_t, bluetooth_t

We can inspect a subset of those with

```
sesearch -A -dt -t=bluetooth_t /etc/selinux/targeted/policy/policy.33 |
head

allow NetworkManager_t bluetooth_t:dbus send_msg;
allow bluetooth_helper_t bluetooth_t:dbus send_msg;
allow bluetooth_helper_t bluetooth_t:socket { read write };
allow bluetooth_t bluetooth_t:alg_socket { accept append bind connect
create getattr getopt ioctl listen lock read setattr setopt shutdown write
};
allow bluetooth_t bluetooth_t:association sendto;
allow bluetooth_t bluetooth_t:bluetooth_socket { accept append bind connect
create getattr getopt ioctl listen lock read setattr setopt shutdown write
}; [ deny_bluetooth ]:False
allow bluetooth_t bluetooth_t:capability { dac_read_search ipc_lock
net_admin net_bind_service net_raw setpcap sys_admin sys_tty_config };
allow bluetooth_t bluetooth_t:dbus send_msg;
allow bluetooth_t bluetooth_t:dir { getattr ioctl lock open read search
watch };
allow bluetooth_t bluetooth_t:fifo_file { append getattr ioctl lock open
read write };
```

c.

Fedora37 server edition

b.

We start by creating the benign/cleared application:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

FILE *f = NULL;

int main() {
    char buf[255];

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

    fread(buf, 256, 1, f);
    printf("%s\n",buf);

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

```
    printf("Sleep is over!\n");
}
```

The adversarial counterpart will be compiled with the same source code.

We compile both applications and move them to the destination

```
gcc main.c -o main
gcc main.c -o adversary

cp main /usr/local/bin/
cp adversary /usr/local/bin/
```

Following the example from redhat we create two services

main.service:

```
[Unit]
Description=Simple testing daemon

[Service]
Type=simple
ExecStart=/usr/local/bin/main

[Install]
WantedBy=multi-user.target
```

adversary.service

```
[Unit]
Description=Simple testing daemon

[Service]
Type=simple
ExecStart=/usr/local/bin/adversary

[Install]
WantedBy=multi-user.target
```

and copy them to their destination

```
cp main.service /usr/lib/systemd/system
cp adversary.service /usr/lib/systemd/system
```

In order to create the security contexts and necessary rules we use **sepolgen**

```
sepolgen --init /usr/local/bin/main
sepolgen --init /usr/local/bin/adversary
```

We leave the adversarial configuration as-is, we modify the one for *main* to give the necessary privileges. The following lines are added to **main.te**:

```
type password_var_t;
files_type(password_var_t)

allow main_t password_var_t:file { open read getattr};
```

To conclude we apply both configurations with

```
./main.sh
./adversary.sh
```

Now, to set the correct context to the password file at */tmp/passwords* we use chcon

```
chcon -t password_var_t /tmp/passwords
```

When inspecting the security contexts when running both services we see that they are correctly labeled:

```
ps -efZ | grep main
system_u:system_r:main_t:s0 root 5832 1 0 16:50 ? 00:00:00
/usr/local/bin/main
```

```
ps -efZ | grep adversary
system_u:system_r:adversary_t:s0 root 5835 1 0 16:50 ? 00:00:00
/usr/local/bin/adversary
```

and the access log confirms that access was blocked for the adversary

```
ausearch -m AVC -ts recent
----
time->Mon Jan 23 16:50:13 2023
type=AVC msg=audit(1674492613.758:671): avc:  denied  { read } for
pid=5835 comm="adversary" name="passwords" dev="tmpfs" ino=100
```

```
scontext=system_u:system_r:adversary_t:s0
tcontext=system_u:object_r:password_var_t:s0 tclass=file permissive=1
----
time->Mon Jan 23 16:50:13 2023
type=AVC msg=audit(1674492613.758:672): avc:  denied  { open } for
pid=5835 comm="adversary" path="/tmp/passwords" dev="tmpfs" ino=100
scontext=system_u:system_r:adversary_t:s0
tcontext=system_u:object_r:password_var_t:s0 tclass=file permissive=1
----
time->Mon Jan 23 16:50:13 2023
type=AVC msg=audit(1674492613.758:674): avc:  denied  { getattr } for
pid=5835 comm="adversary" path="/tmp/passwords" dev="tmpfs" ino=100
scontext=system_u:system_r:adversary_t:s0
tcontext=system_u:object_r:password_var_t:s0 tclass=file permissive=1
```