Comp 4108

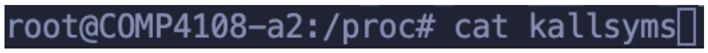
Assignment 2

Alexandre Berube (101054165)

October 15, 2024

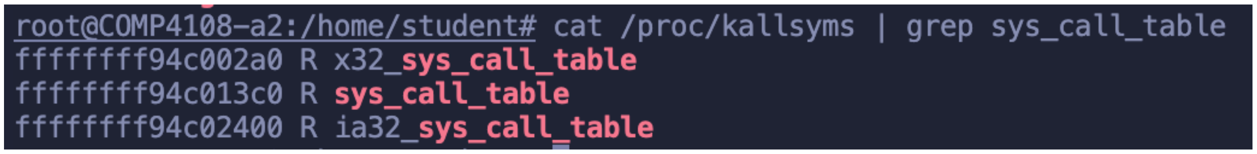
Part A

3) For this question, I first went to the /proc/ file, and then used the following to search for the address for /proc/kallsyms :



```
root@COMP4108-a2:/proc# cat kallsyms
```

This ends up creating an incredibly long output, so I had to redo the command, using grep this time to find only the results that actually had sys_call_table, which resulted in the following command:

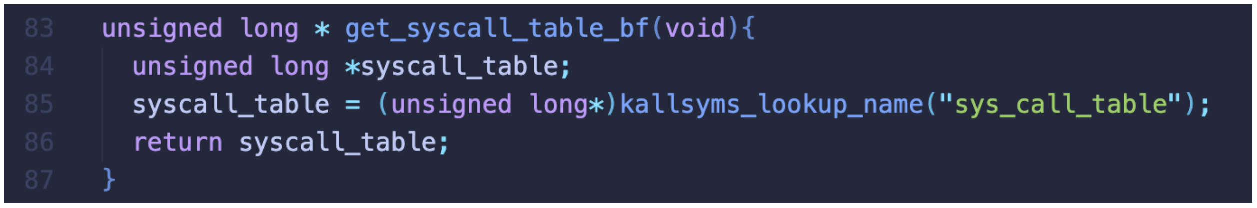


```
root@COMP4108-a2:/home/student# cat /proc/kallsyms | grep sys_call_table
ffffffff94c002a0 R x32_sys_call_table
ffffffff94c013c0 R sys_call_table
ffffffff94c02400 R ia32_sys_call_table
```

The following 3 lines were the output for the above command as well. I think the top and bottom might have to do with if we were using a 32 bit system perhaps, though I am not sure and am opting to primarily use the middle one for this assignment.

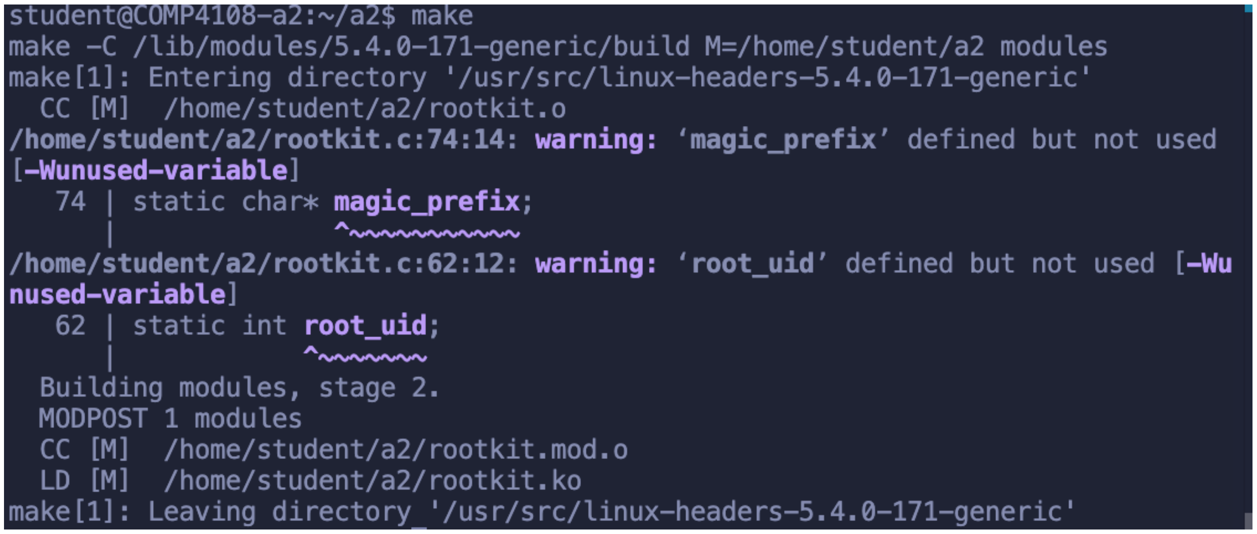
So, the address of the sys_call_table is ffffffffa9e013c0 in this case.

4) For this question, I had to find the appropriate function (kallsyms_lookup_name()) where we are supposed to input the symbol for the right address found in the previous question. Here is the code block where this takes place and the symbol has been input into the appropriate spot:



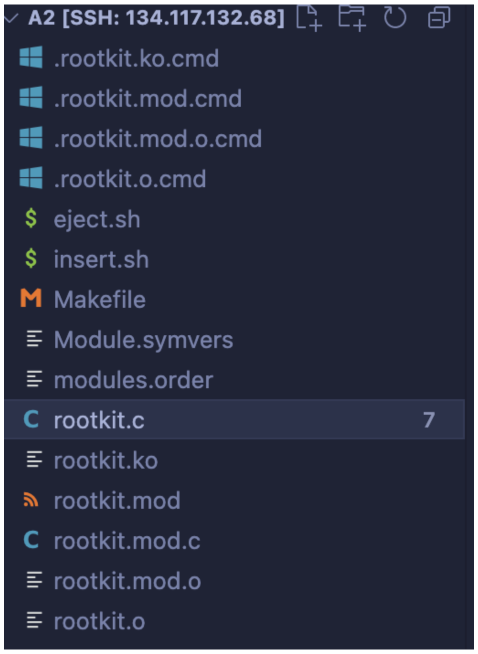
```
unsigned long * get_syscall_table_bf(void){
  unsigned long *syscall_table;
  syscall_table = (unsigned long*)kallsyms_lookup_name("sys_call_table");
  return syscall_table;
}
```

5) I was able to confirm the rootkit works by running the command make:



```
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
/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 [-Wu
nused-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'
```

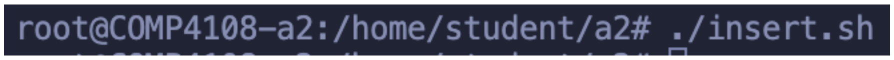
This resulted in the following file structure being created:



```
A2 [SSH: 134.117.132.68]
.rootkit.ko.cmd
.rootkit.mod.cmd
.rootkit.mod.o.cmd
.rootkit.o.cmd
eject.sh
insert.sh
Makefile
Module.symvers
modules.order
rootkit.c                7
rootkit.ko
rootkit.mod
rootkit.mod.c
rootkit.mod.o
rootkit.o
```

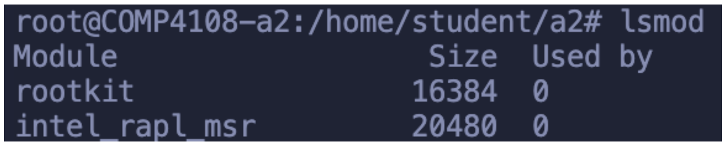
As stated in the assignment spec, the few warnings about defined but unused variables are meant to be there.

6) I was able to use ./insert.sh as the root user in order to insert the rootkit framework using the following:



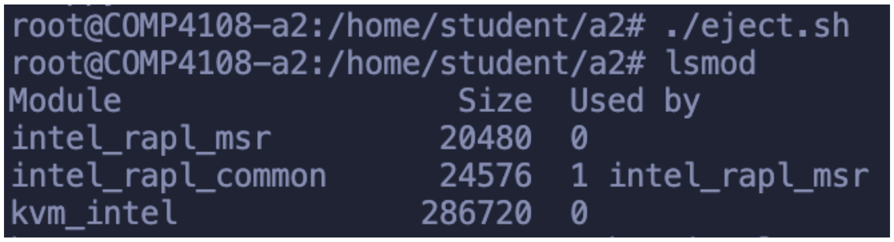
```
root@COMP4108-a2:/home/student/a2# ./insert.sh
```

Then, we can verify that it has been added by using lsmod, which shows rootkit at the very top of the list since it is the most recent addition (entire list not included):



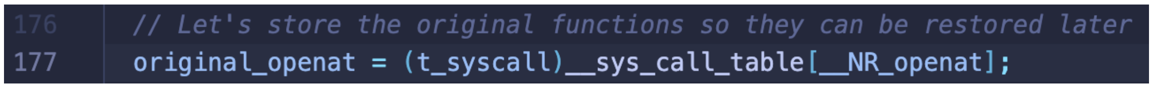
```
root@COMP4108-a2:/home/student/a2# lsmod
Module                  Size  Used by
rootkit                16384  0
intel_rapl_msr         20480  0
```

7) For this question now, we run the command ./eject.sh as the root user, and the confirm that the rootkit module was removed by again checking with lsmod. These steps are all seen in this screenshot:



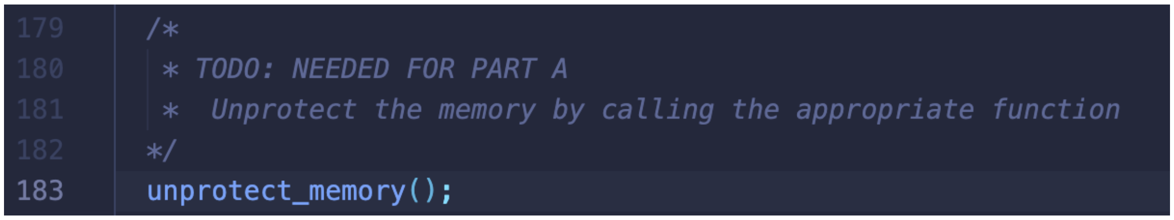
```
root@COMP4108-a2:/home/student/a2# ./eject.sh
root@COMP4108-a2:/home/student/a2# lsmod
Module                  Size  Used by
intel_rapl_msr         20480  0
intel_rapl_common      24576  1 intel_rapl_msr
kvm_intel             286720  0
```

8) For this question, I am looking for all the sections of the rootkit.c file that have TODO in the comments. The first is to uncomment the following line:



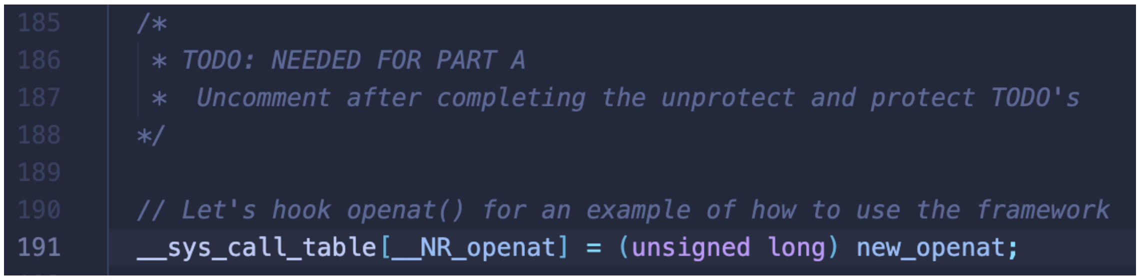
```c
// Let's store the original functions so they can be restored later
original_openat = (t_syscall)__sys_call_table[__NR_openat];
```

Then, this function call to unprotect_memory() is added:



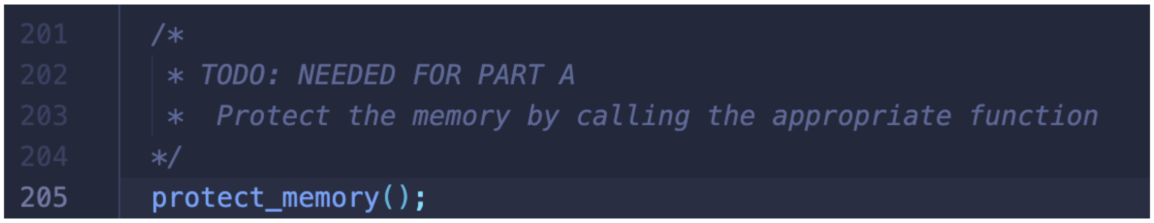
```c
/*
 * TODO: NEEDED FOR PART A
 *  Unprotect the memory by calling the appropriate function
*/
unprotect_memory();
```

Followed by this step:



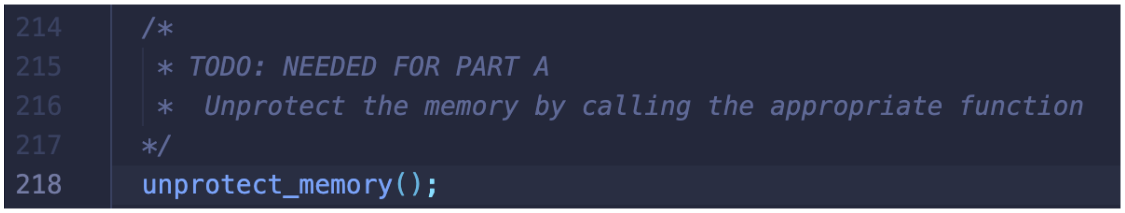
```c
/*
 * TODO: NEEDED FOR PART A
 *  Uncomment after completing the unprotect and protect TODO's
*/

// Let's hook openat() for an example of how to use the framework
__sys_call_table[__NR_openat] = (unsigned long) new_openat;
```

The following function call to protect_memory() is added:



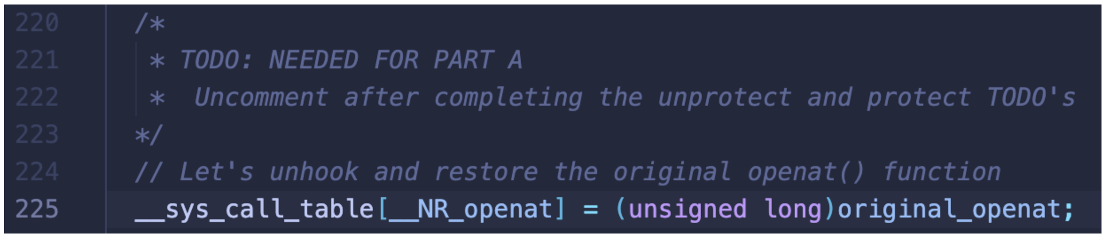
```c
/*
 * TODO: NEEDED FOR PART A
 *  Protect the memory by calling the appropriate function
*/
protect_memory();
```

Then further down, we do a few similar steps, again adding the unprotect_memory() function call here:



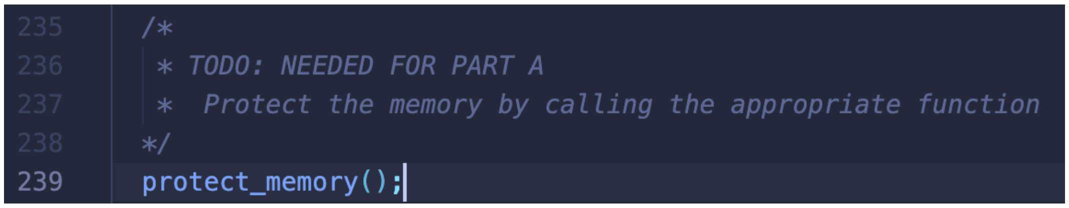
```c
/*
 * TODO: NEEDED FOR PART A
 *  Unprotect the memory by calling the appropriate function
*/
unprotect_memory();
```

Then we uncomment the below line:



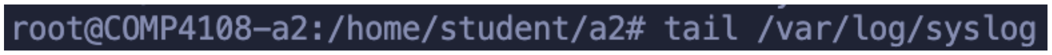
```c
/*
 * TODO: NEEDED FOR PART A
 *  Uncomment after completing the unprotect and protect TODO's
*/
// Let's unhook and restore the original openat() function
__sys_call_table[__NR_openat] = (unsigned long)original_openat;
```

Then finally we add another use of the protect_memory() function here:



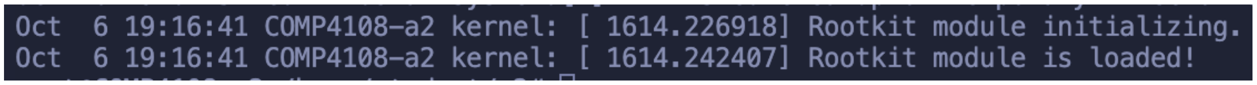
```
/*
 * TODO: NEEDED FOR PART A
 *  Protect the memory by calling the appropriate function
*/
protect_memory();
```

After running the following command:



```
root@COMP4108-a2:/home/student/a2# tail /var/log/syslog
```

We can see the following lines show up at the bottom of the resulting information, indicating the success of the open() hook:



```
Oct  6 19:16:41 COMP4108-a2 kernel: [ 1614.226918] Rootkit module initializing.
Oct  6 19:16:41 COMP4108-a2 kernel: [ 1614.242407] Rootkit module is loaded!
```

9) I think principle P4, Complete Mediation, can help mitigate rootkits. This is because the principle is concerned with the premise that to access objects on a system one must have their authority verified. A rootkit is meant to bypass this verification initially into the system, but if this verification is needed every time to access objects, then the effectiveness of said rootkit will be drastically reduced.

   I also think P5, Isolated-Compartments, is applicable to help mitigate rootkits. If system components are more isolated, then it is harder to access the whole system just because you have access to a part of it which can undermine the use of a rootkit by limiting the range of objects that it can have access to in the first place.

Part B

1)