Week 1 - Dirty COW Attack Lab

1 Lab Overview

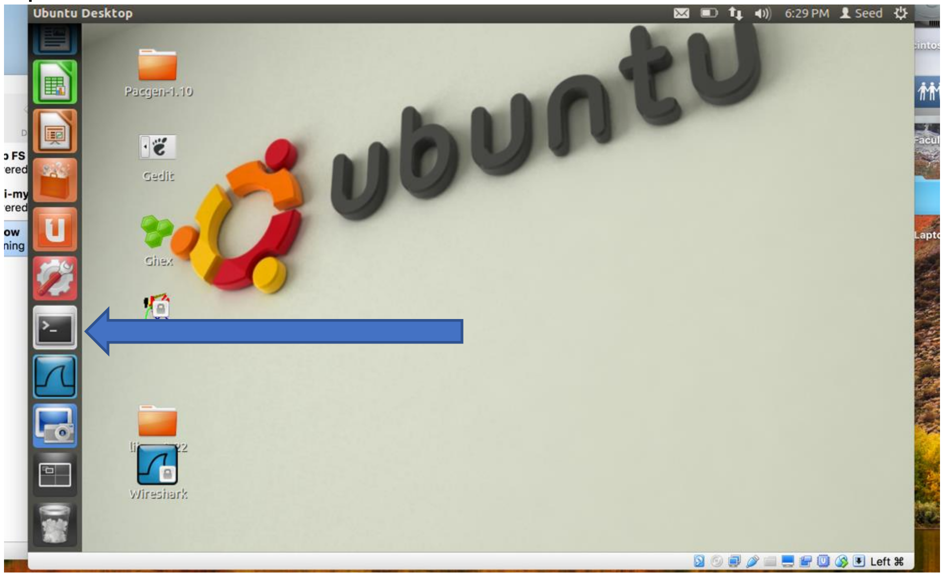
The Dirty COW vulnerability is an interesting case of the race condition vulnerability. It existed in the Linux kernel since September 2007, and was discovered and exploited in October 2016. The vulnerability affects all Linux-based operating systems, including Android, and its consequence is very severe: attackers can gain the root privilege by exploiting the vulnerability. The vulnerability resides in the code of copy-on-write inside Linux kernel. By exploiting this vulnerability, attackers can modify any protected file, even though these files are only readable to them.

The objective of this lab is for students 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. In this lab, students will exploit the Dirty COW race condition vulnerability to gain the root privilege.

Note: This lab is based on the SEEDUbuntu12.04 VM. The VM is 32-bit. Follow the instruction in useVirtualBox.pdf to set up the VM in your VirtualBox.

Drop Box Link for VM:
https://www.dropbox.com/home/MSSD%20Lab/Week%201%20Cow
Run the VM
Login password for username Seed: **dees**

Open the Terminal in the VM



2 Demo: Modify a Dummy Read-Only File

The objective of this task is to write to a read-only file using the Dirty COW vulnerability.

2.1 Create a Dummy File

We first need to select a target file. Although this file can be any read-only file in the system, we will use a dummy file in this task, so we do not corrupt an important system file in case we make a mistake. Please create a file called zzz in the root directory, change its permission to read-only for normal users, and put some random content into the file using an editor such as gedit.

Touch is to create a file

```
$  sudo touch /zzz                          (touch is used to create a new file named “zzz”)
key in the password dees                    (the password does appear on the screen)
$  sudo chmod 644 /zzz             (chmod => change permission 644=>files are readable and writable
$  sudo gedit /zzz
key in “111111222222333333” and save in the gedit
$ cat /zzz
You should see 111111222222333333
$ ls -l /zzz
-rw-r--r—1 root root 19 Oct 18 22:03 /zzz
```

```
$ ls -l /zzz
-rw-r--r—1 root root 19 Oct 18 22:03 /zzz
$ echo 99999 > /zzz (Trying to add 99999 to /zzz but denied as you do not have permission because
you are not sudo superuser)
bash: /zzz: Permission denied
```

Try to alter the file:

```
$ echo 99999 > /zzz                         (Trying to add 99999 to /zzz)
bash: /zzz: Permission denied
```

From the above experiment, 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 system, we can find a way to write to this file. Our objective is to replace the pattern "222222" with "******".

2.2 Set Up the Memory Mapping Thread

You can download the c program cow_attack.c from
https://www.dropbox.com/home/MSSD%20Lab/Week%201%20Cow

The program cow_attack.c has three threads: the main thread, the write thread, and the **madvise thread**. The main thread maps /zzz to memory, finds where the pattern "222222" is, and then creates two threads to exploit the Dirty COW race condition vulnerability in the OS kernel.

In the main thread, we need to find where the pattern "222222"is. We use a string function called strstr() to find where "222222" is in the mapped memory (Line À). We then start two threads: madviseThread (Line Á) and writeThread (Line Â).

2.3 Set Up the **write** Thread

The job of the write thread listed in the following is to replace the string "222222" in the memory with "******". Since the mapped 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 not cause any change to the underlying /zzz file.

2.4 The **madvise** Thread

The madvise thread does only one thing: discarding the private copy of the mapped memory, so the page table can point back to the original mapped memory.

2.5 Launch the Attack

If the write() and the madvise() system calls are invoked alternatively, i.e., one is invoked only after the other is finished, the write operation will always be performed on the private copy, and we will never be able to modify the target file. The only way for the attack to succeed is to perform the madvise() system call while the write() system call is still running. We cannot always achieve that, so we need to try many times. As long as the probability is not extremely low, we have a chance. That is why in the threads, we run the two system calls in an infinite loop. Compile the cow attack.c and run it for a few seconds. If your attack is successful, you should be able to see a modified /zzz file.

```
$ gcc cow_attack.c -lpthread                  (compile the attack file)
$ a.out
... press Ctrl-C after a few seconds ...
```

Report your results in the lab report and explain how you are able to achieve that.

3 Assignment: Modify the Password File to Gain the Root Privilege

Now, let’s launch the attack on a real system file, so we can gain the root privilege. We choose the **/etc/passwd** file as our target file. This file is world-readable, but non-root users cannot modify it. The file contains the user account information, one record for each user. Assume that our user name is seed. The following lines show the records for root and seed:

```
$ cat /etc/passwd                          (View the passwd file)


root:x:0:0:root:/root:/bin/bash
seed:x:1000:1000:Seed,123,,:/home/seed:/bin/bash
```

Each of the above record contains seven colon-separated fields. Our interest is on the third field, which specifies the user ID (UID) value assigned to a user. UID is the primary basis for access control in Linux, so this value is critical to security. The **root** user’s UID field contains a special value **0**; that is what makes it the superuser, not its name. Any user with UID 0 is treated by the system as root, regardless of what user name he or she has. The **seed user’s** ID is only **1000**, so it does not have the root privilege. However, if we can change the value to **0000**, we can turn it into root. We will exploit the Dirty COW vulnerability to achieve this goal.

In our assignment, so that we do not corrupt the user:seed in case we make a mistake, we will create a new normal user account called <YOUR NAME>

Adding a new account can be achieved using the adduser command. After the account is created, a new record will be added to /etc/passwd. See the following:

```
$ sudo adduser <YOUR NAME>                              (create new normal user)
  …
$ cat /etc/passwd | grep <YOUR NAME>
<YOURNAME>:x:1001:1001:,,,:/home/<YOURNAME>:/bin/bash
```

You should turn this normal user <YOUR NAME> into a root user using the Dirty COW attack.

We suggest that you save a copy of the /etc/passwd file just in case you make a mistake and corrupt this file.

Clue: You need to modify the new user’s entry in /etc/passwd, so the third field is changed to 0000. The file is not writable to the new user, but we can use the Dirty COW attack to write to this file. You shall modify the cow_attack.c to achieve this goal.

If your attack is successful, you will be able to notice something unusual and interesting when you switch user to user: <YOUR NAME>.

```
seed@ubuntu$ su <YOUR NAME>                             (Switch user)
Passwd:
```

Use the command “id” in the command line to see the user privileges.

4 Submission

You need to submit a detailed lab report to describe what you have done and what you have observed (Both in demo exercise and assignment exercise). Include a brief explanation of the cow_attack.c code used in both exercises. Please provide details using screen shots and code snippets. You also need to provide explanation to the observations that are interesting or surprising.

You do not need to submit your cow_attack.c code