INCS-745 Intrusion Detection and Hacker Exploits: Assignment 3

Hedao Tian
1256221

Race Condition Vulnerability Lab

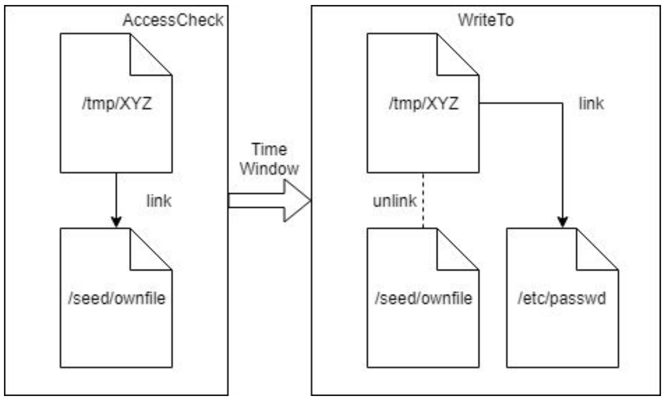
Choosing Our Target => /etc/passwd

By injecting our own string into this file, we can add any new user with any privileges.

Vulnerability

Vulp.c is a root owned Set-UID program which allow user with lower privilege to execute with escalated privilege. For example, the program will run under effective user ID when the real user ID is on hold. The weakness of this program is the time window between checking access right to a public file /tmp/XYZ and writing to it. The goal is to use /tmp/XYZ as a pointer, and let it points to the target file after access checking is passed. To do that within the time window, we use two programs on our side and run them in parallel.

check.sh //This program write whatever in our input file to the vulp.c everytime when checking the /etc/passwd file is not modified. The process happens in a loop. So it will run until change.
exploit.c // This program uses unlin and symlink to repeatedly delete and create the symbolic link from /tmp/XYZ to our own file and /etc/passwd. So it will create a chance to let /tmp/XYZ points to /etc/passwd file right after the access checking is bypassed.



Observation

When running both program together, the /etc/passwd file can finally be changed after certain attempts since the time window is not very big. And our new user can be inserted.

CounterMeasure-Least Privilege

Since the vulp.c program is a Setuid program that actually runs under root effectiveness while the real user is seed, and it only do one single check by **access()**. So what if the whole program runs under the real user's privilege but not the root? The answer is we can prevent race condition by changing the effective id to the real id before anything happens=> **seteuid(uid);** So the program can never write to the /etc/passwd file.

CounterMeasure-Using Ubuntu's Built-in Scheme

$ sudo sysctl -w kernel.yama.protected_sticky_symlinks=1

eUID	Directory Owner	Symlink Owner	Decision
seed	seed	root	Denied
root	**root**	**seed**	**Denied**

INCS-745 Intrusion Detection and Hacker Exploits: Assignment 3

Hedao Tian
1256221

Dirty COW Attack Lab

Task 1: Modify a Dummy Read-Only File

We firstly created a file under root and make it read-only for all other users including seed. Then we have a program with three threads main(maps /zzz to memory, create duplicate file into the physical memory and points the virtual memory to it when writing is not permitted), write(overwrite the predefined part of the memory), madvise(close the writing and remove the duplicated part of memory, points back to the original physical memory storing the file /zzz). By running the program containing those three threads, the designated part of the read-only file can be modified by us.

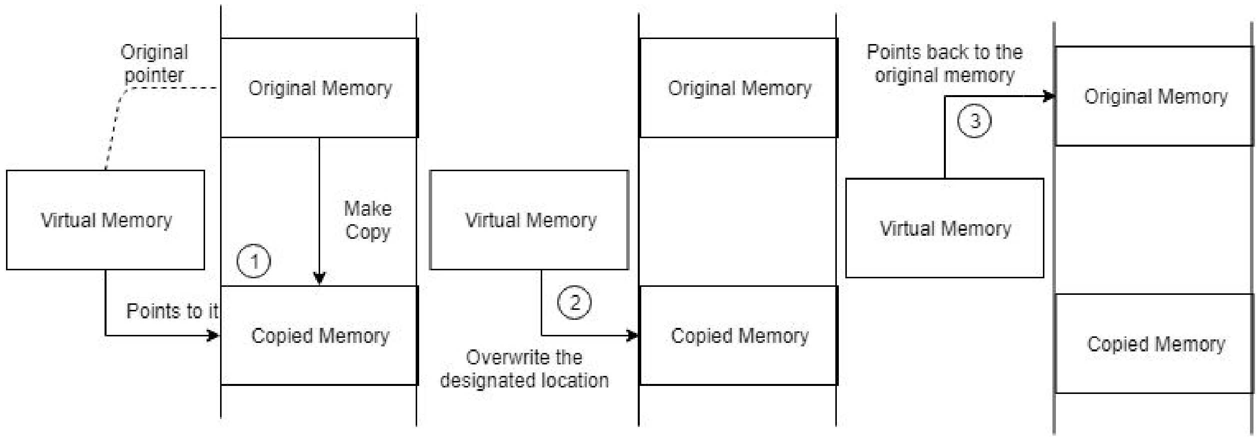
To find the designated part of the memory, we use **char *position = strstr(map, "222222");**

And then write to this location by **pthread_create(&pth2, NULL, writeThread, position);**

The content that is used to write into this memory is defined in the writeThread.

Vulnerability

As the name, COW(copy on write) shown, the vulnerability happens during the copy, write and terminate processes. As shown below there are 3 processes which are copying the memory, pointing to the copied location to write and pointing back to the original location. The weakness of this setting is process 1 and 2 are not atomic(especially when copying large memory of a file) which means we have a chance to insert process 3 between them while running process 3 before 2 repeatedly, thus the original memory can be modified. And the processes are run in 1, 3, 2.



Observation

We use pthread_join to run those processes in parallel so we need to add gcc option lpthread while compiling the exploit.c program to let the shell knows we use certain libraries. By executing the program for few seconds I could see the read-only file was modified. *We need to cancel the execution manually since those last two processes were running repeatedly.

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

In this task I was asked to modify my own user privilege to root. Since the /etc/passwd file is readable for every user, I could seed which syntax I need to change. So just simply modify below parts of the code and then execute the program, attack can be achieved:

char *position = strstr(map, "1000"); //inside main thread, define the location of my privilege.

char *content= "0000"; //inside write thread, change the overwrite content to root privilege.