Race Condition Vulnerability

Overview

The learning objective of this lab if for you to gain the first-hand experience on the race-condition vulnerability. A race condition occurs when multiple processes access and manipulate the same data concurrently, and the outcome of the execution depends on the particular order in which the access takes place. If a privileged program has a race-condition vulnerability, attackers can run a parallel process to "race" against the privileged program in order to change the behavior of the privileges program.

In this lab, you will be given a program with a race-condition vulnerability; your task is to develop a scheme to exploit the vulnerability and gain the root privilege. In addition to the attacks, you will be guided to walk through several protection schemes that can be used to counter the race-condition attacks.

Submission

As you complete the following tasks, you will compose a lab report that documents each step you take, including screenshots to illustrate the effects of commands you type, and describing your observations as well as answering any questions posed. Feel free to use google to gather more information on the tasks and answers the questions. Please bring a printed version of your report to class the day it is due.

Task 0: lab setup

Linux has a built-in protection that will stop our race condition attack. To disable this protection login as root and then enter

```
sysctl -w kernel.yama.protected_sticky_symlinks=0
```

Log out as root.

Task 1: Run vulp

The first step is to simply run the ***vulp*** program with input from a file ***hackerAccountInfo.txt*** that it appends to ***/tmp/XYZ***, which is linked to a file ***dummy.txt***. There are many places where things can go wrong so it is best to do this in stages, testing after each step. We'll walk you through the process.

a. Create the file ***vulp.c*** using the following code and compile it.

```
/* vulp.c                                          */
/* This program reads user input and appends */
/* it as a new line to the file /tmp/XYZ     */
#include <stdio.h>
#include <unistd.h>
#include <string.h>
int main()
{
  char *fname = "/tmp/XYZ";          // we append to this
file
  char buffer[512];
  FILE *fp;
  scanf("%s", buffer );              // get string to
append
  if(!access(fname, W_OK)){          // check write access
of /tmp/XYZ
    fp = fopen(fname, "a+");            // open file fr
appending
    fwrite("\n", sizeof(char), 1, fp);  // add newline
    fwrite(buffer, sizeof(char), strlen(buffer), fp); //
add text
    fclose(fp);                         // close file
   }
   else printf("No permission \n");
}
```

b. First you should make sure that the program works. It expects ***/tmp/XYZ*** to exist. To create an empty file with that name enter the command:

```
touch /tmp/XYZ
```

Run ***vulp***. It will wait for input. Enter some random text, e.g. "abcdefg hijklmnop" and then type return. Check ***/tmp/XYZ*** to confirm the text has been appended. Note you should see a blank line from the "\n" and then some text without a return; i.e. the shell prompt should immediately follow the text you entered. Does it? (We do this so vulp will work appropriately with ***/etc/passwd*** and ***/etc/shadow***.) Also, how does the scanf in the ***vulp*** program handle whitespace?

c. Next we want to link ***/etc/XYZ*** to a file, e.g. ***dummy.txt***. First remove ***/tmp/XYZ*** by entering:

```
rm /tmp/XYZ
```

Now create the file dummy.txt by entering the following command:

```
echo "hello" >  dummy.txt
```

Finally create a symbolic link from ***/tmp/XYZ*** to ***dummy.txt***. Use absolute paths (or regret it.) Now list that file (with -l flag) and be sure you see a link indicated by ***/tmp/XYZ -> dir/dummy***, where dir is your directory for this lab. (Note: the order of arguments to the link command is somewhat counterintuitive. Make sure you have linked in the right direction.)

d. To make sure the permissions and links are set appropriately enter the command:

```
echo "hey there" >> /tmp/XYZ
```

By using >> we are appending the text "hey there" to the file ***/tmp/XYZ***. (If we used > we would overwrite /***tmp/XYZ*** with the text "hey there.")

Print out dummy.txt. Has the line "hey there" been appended?

e. Run ***vulp***. The program will wait for input. Enter some text; e.g. "this_is_a_test,_this_is_only_a_test". You should now find that text appended to dummy.txt. Do you?

f. Next you need to create the file ***hackerAccountInfo.txt*** that has the text you want to (eventually) append to ***/etc/passwd***. Open /etc/passwd and locate the record for root. You want to create a line of text that is similar but for a user called hacker with your own home director. Put this text in a file called ***hackerAccountInfo.txt***. Now run the command

```
vulp < hackerAccountInfo.txt
```

Check to see be sure the hacker account info has been appended to ***dummy.txt***.

g. Finally make ***vulp*** a setuid program. Repeat the last step to be sure everything is working.

h. How long did this task take? Do you have any suggestions on how to improve it?

Task 2: Create runVulp.sh

The next step is to create the shell script that runs ***vulp*** repeatedly.

a. Create a shell script ***runVulp.sh*** with the following code (this assumes /bin/sh points to /bin/bash):

```
#!/bin/sh
```

```
i=0
while :
do
./vulp < hackerAccountInfo.txt
let i=$i+1
echo $i
done
```

Make ***runVulp.sh*** executable (i.e. use the chmod command).

b. Run the script. After a few seconds enter control c to kill the process. Check that the hacker account info has been appended (many time) to ***dummy.txt***.

c. Now create an empty file ***myPasswd.txt***. Link (symbolic) ***/tmp/XYZ*** to it. Run you ***runVulp.sh*** script again for a few seconds. Check that the hacker account info has been appended (many times) to ***myPasswd.txt***.

d. How long did this section take? Did you have any problems? Do you have any suggestions for improving this task?

Task 3: Prototype exploit.sh

The next step is to prototype the exploit shell that changes the link ***/tmp/XYZ***. Before we attack ***/etc/passwd*** we'll debug our script on a different (i.e. safer) file.

a. Create a shell script ***exploit.sh*** that does the following

- creates a symbolic using the -f flag from /tmp/XYZ to dummy.txt
- lists /tmp/XYZ with details (i.e. ls -l /tmp/XYZ)
- creates a symbolic link from -f flag /tmp/XYZ to myPasswd.txt
- lists /tmp/XYZ with details

b. The -f flag forces the link to be established even when ***/tmp/XYZ*** exists. Make the script executable and run it. The listings should verify that the linking is done correctly; e.g. after linking ***/tmp/XYZ*** to dummy.txt the listing should show ***/tmp/XYZ -> /dir/myPasswd.txt***, where, as before, dir, is your directory for this lab.

c. Next augment your script so that it repeats the linking/listing process in an infinite loop. (Hint: see how the loop was done in Task 2.) Run it for a few seconds and confirm it links/unlinks appropriately. Hit control c to stop the process.

d. How long did this section take? Did you have any problems? Do you have any suggestions for improving this task?

Task 4: Testing the scripts in tandem

Now you are going to dry run the scripts by running them both simultaneously. You have write permission for both ***dummy.txt*** and ***myPasswd.txt*** so you should see ***hackerAccountInfo.txt*** appended whichever file is currently linked to ***/tmp/XYZ.***

a. Open a second terminal window. Login as root and re-enter the command in task 0.

b. Start ***runVulp.sh*** in one terminal and ***exploit.sh*** in the other. After a few seconds stop both processes. You should find that ***hackerAccountInfo*** has been appended to both ***dummy.txt*** and many times.

c. How long did this section take? Did you have any problems? Do you have any suggestions for improving this task?

Task 5: Halting on success

We want both script to halt when ***hackerAccountInfo.txt*** has been appended to myPasswd.txt. In practice, a sys admin would notice if ***/etc/passwd*** grew significantly.

a. First delete both dummy.txt and myPasswd.txt and then touch each to recreate them. (We want to start with them empty.)

b. At the command line enter

```
ls -l myPasswd.txt
```

Do you know what all of the fields mean? Will any of them change when we have successfully appended ***hackerAccountInfo.txt***?

c. Modify the while loops in your scripts so that they stop when the exploit has succeeded. Repeat the test run you did in the last task to make sure the scripts stop when myPasswd.txt has been modified.

d. How long did this section take? Did you have any problems? Do you have any suggestions for improving this task?

Task 6: Trial run

Now we'll run the full exploit on myPasswd.txt.

a. Login as root. Change the owner to ***myPassword.txt*** to root. Make it readable but not writable by others. Exit from the root account.
b. Repeat the same experiment you did in the last two tasks. This time it will take awhile for a context switch to happen at the right time so that ***vulp*** checks the permissions on ***dummy.txt*** but actually opens ***myPasswd.txt***.
c. How long did this section take? Did you have any problems? Do you have any suggestions for improving this task?

Note: Some times things can go wrong. If everything worked in the last task, this one should be fine. But you should monitor the size of ***dummy.txt*** to be sure it is growing and monitor the ***exploit.sh*** process for error messages. Occasionally the owner of ***/tmp/XYZ*** gets changed to root. If so, you'll see lots of errors in the exploit.sh process. You need to stop both processes, login as root, remove the ***/tmp/XYZ*** file, and restart everything. Note: errors in the ***runVulp.sh*** process are to be expected as you are often trying to write to a file you don't have permission for.

Task 7: Let's hack!

Finally we will use the exploit to create a new account with root privilege. Note, there are reports of things going very wrong with this step. In particular someone (reportedly) wiped out their ***/etc/passwd*** and/or ***/etc/shadow*** files. If that happens you cannot login and will have to reinstall the vm. We don't want that to happen so we'll be sure to take necessary cautions.

a. Open a new terminal (this should be the third!). Login as root. Create backup copies of ***/etc/passwd*** and ***/etc/shadow*** files. Do NOT log out as root and do not close this terminal. If something goes awry in the next steps you'll be able to restore the ***/etc/passwd*** and ***/etc/shadow*** files in this terminal.
b. Edit both scripts replacing ***myPasswd.txt*** with ***/etc/passwd***.
c. Run both scripts in the first two terminal windows as you did in the last tasks. When the scripts stop, check ***/etc/passwd***. You should find the record for the new hacker account.
d. Next you need to replicate the attack but adding an appropriate password record for the new hacker account to the ***/etc/shadow*** file. Create a new file ***hackerPasswd.txt*** with an appropriate line of text (hint: you might copy and modify the password record for the

seed account.) Modify your scripts to append this new file to ***/etc/shadow***. Run the scripts until ***/etc/shadow*** is modified.

e. Login into the hacker account. Confirm you have root privilege!

f. How long did this section take? Did you have any problems? Do you have any suggestions for improving this task?