1. **Overview**

The learning objective of this lab is for students to gain the first-hand experience on a race-condition vulnerability by putting what they have learned about the vulnerability in class into action. 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, with an intention to change the behavior of the program.

In this lab, students will be given a program with a race-condition vulnerability; their task is to develop a scheme to exploit the vulnerability and gain root privilege. In addition to the attacks, students will be guided to walk through several protection schemes that can be used to counter the race-condition attacks. Students need to evaluate whether the schemes work or not and explain why. This lab covers the following topics:

This lab covers the following topics:

* Race Condition Vulnerability
* Sticky Symlink Protection
* Principle of Least Privilege

1. **Lab Setup**

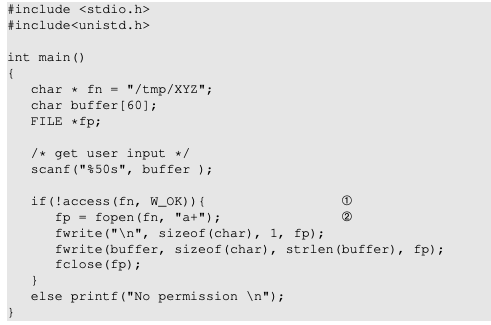
Files needed for this lab are included in the Lab\_04\_RaceCondition folder

**Turning Off Countermeasures**

Ubuntu has a built-in protection against the race condition attacks demonstrated in this lab. This scheme works by restricting who can follow a symlink. According to the documentation, “symlinks in world-writable sticky directories (e.g. /tmp) cannot be followed if the follower and directory owner do not match the symlink owner.” Ubuntu 20.04 introduced another security mechanism that prevents the root from writing to the files in /tmp that are owned by others. In this lab, we need to disable these protections. You can achieve that using the following commands:



The following program is a seemingly harmless program. It contains a race-condition vulnerability.

****

Listing 1 vulp.c

The program above is a root-owned Set-UID program; it appends a string of user input to the end of a temporary file /tmp/XYZ. Since the code runs with root privileges, i.e., its effective user ID is zero, it can overwrite any file. To prevent itself from accidentally overwriting other people’s files, the program first checks whether the real user ID has the access permission to the file /tmp/XYZ; that is the purpose of the access() call in Line ➀. If the real user ID indeed has the correct permissions, the program opens the file in Line ➁ and appends the user input to the file.

At first glance the program does not seem to have any problem. However, there is a race condition vulnerability in this program: due to the time window between the check (access) and the use (fopen), there is a possibility that the file used by access() is different from the file used by fopen(), even though they have the same file name /tmp/XYZ. If a malicious attacker can somehow makes /tmp/XYZ a symbolic link pointing to a protected file, such as /etc/passwd, inside the time window, the attacker can cause the user input to be appended to /etc/passwd, and can thus gain root privileges. The vulnerable program runs with root privileges, so it can overwrite any file.

**Set up the Set-UID program.** We first compile the above code, and turn its binary into a Set-UID program that is owned by the root. The following commands achieve this goal:

****

1. **Lab Tasks**
   1. **Task A:**

We would like to exploit the race condition vulnerability in the program. We choose to target the password file /etc/passwd, which is not writable by normal users. By exploiting the vulnerability, we would like to add a record to the password file, with a goal of creating a new user account that has root privileges. Inside the password file, each user has an entry, which consists of seven fields separated by colons (:). The entry for the root user is listed below:

****

For the root user, the third field (the user ID field) has a value zero. Namely, when the root user logs in, its process’s user ID is set to zero, giving the process root privileges. Basically, the power of the root account does not come from its name, but instead from the user ID field. If we want to create an account with root privileges, we just need to put a zero in this field. Each entry also contains a password field, which is the second field. In the example above, the field is set to "x", indicating that the password is stored in another file called /etc/shadow (the shadow file). If we follow this example, we have to use the race condition vulnerability to modify both password and shadow files, which is not very hard to do. However, there is a simpler solution. Instead of putting "x" in the password file, we can simply put the password there, so the operating system will not look for the password from the shadow file. The password field does not hold the actual password; it holds the one-way hash value of the password. To get such a value for a given password, we can add a new user in our own system using the adduser command, and then get the one-way hash value of our password from the shadow file. Interestingly, there is a magic value used in Ubuntu live CD for a password-less account, and the magic value is 6aMy0wojraho (the 6th character is zero, not letter O). If we put this value in the password field of a user entry, we only need to hit the return key when prompted for a password.

To verify whether the magic password works or not, we manually (as a superuser) add the following entry to the end of the /etc/passwd file. Please report whether you can log into the the account without typing a password, and check whether it has root privileges.

test:U6aMy0wojraho:0:0:test:/root:/bin/bash

After this task, please remove this entry from the password file. In the next task, we need to achieve this goal as a normal user. Clearly, we are not allowed to do that directly to the password file, but we can exploit a race condition in a privileged program to achieve the same goal.

**NOTE:** At this point please make a copy of the passwd file in case something bad happens to it.

sudo cp /etc/passwd /etc/passwd\_back

* 1. **Task B: Launch Race Condition**

The goal of this task is to exploit the race condition vulnerability in the vulnerable Set-UID program listed earlier. The ultimate goal is to gain the root privilege. The most critical step of the attack, making /tmp/XYZ point to the password file, must occur within the window between check and use; namely between the access and fopen calls in the vulnerable program.

1. **Submission**

Once you have finished all of the tasks, email your submission to [brian.swenson@gtri.gatech.edu](mailto:brian.swenson@gtri.gatech.edu). You submission should include your observations where requested in the lab and byte offsets used in badfile to obtain a root shell for the tasks in section 3. Include in your submission your name and the name of your partner if you worked on the lab in pairs.

1. **References**

This lab is based on the seed 2.0 labs created by Wenliang Du and is protected by the following license:

