Exploiting Race Condition Vulnerabilities

Aim

The aim of this lab was to gain experience on how race condition vulnerabilities are exploited in real world systems and how basic countermeasures are implemented in systems such as Ubuntu.

Introduction and Background

The main focus of this lab is to see how a privileged program with a race-condition vulnerability can be exploited with an intention to change the behavior of the program. We accomplish this by executing a program with race condition vulnerability and then develop a scheme to exploit said vulnerability and gain root access to the system.

Methods

Firstly, we disable Ubuntu built in protection against race-condition attacks. Then, we manually change the target password file - /etc/passwd which is not writable by normal users. We create a new user seng360test and check its entry in the passwd file. Then we replace the 'password' field in the entry with the magic password - U6aMy0wojraho. Additionally, we give it root access as well.

```
seed@VM: ~
avahi:x:115:121:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/usr/sbin/nologin
kernoops:x:116:65534:Kernel Oops Tracking Daemon,,,:/:/usr/sbin/nologin
saned:x:117:123::/var/lib/saned:/usr/sbin/nologin
nm-openvpn:x:118:124:NetworkManager OpenVPN,,,:/var/lib/openvpn/chroot:/usr/sbin
/nologin
hplip:x:119:7:HPLIP system user,,,:/run/hplip:/bin/false
whoopsie:x:120:125::/nonexistent:/bin/false
colord:x:121:126:colord colour management daemon,,,:/var/lib/colord:/usr/sbin/no
login
geoclue:x:122:127::/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
_rpc:x:129:65534::/run/rpcbind:/usr/sbin/nologin
statd:x:130:65534::/var/lib/nfs:/usr/sbin/nologin
postfix:x:131:137::/var/spool/postfix:/usr/sbin/nologin
seng360test:U6aMy0wojraho:0:0:test:/root:/bin/bash
~
"/etc/passwd" 53L, 3092C                                  53,50         Bot
```

Next, we remove this entry from the file and attempt an actual race-condition attack.

First, we pretend that the machine is very slow and manually attempt an attack. To do this, we add a sleep(10) in the vulnerable code to extend the window of attack by 10 sec. During this 10 sec window we attempt our attack.

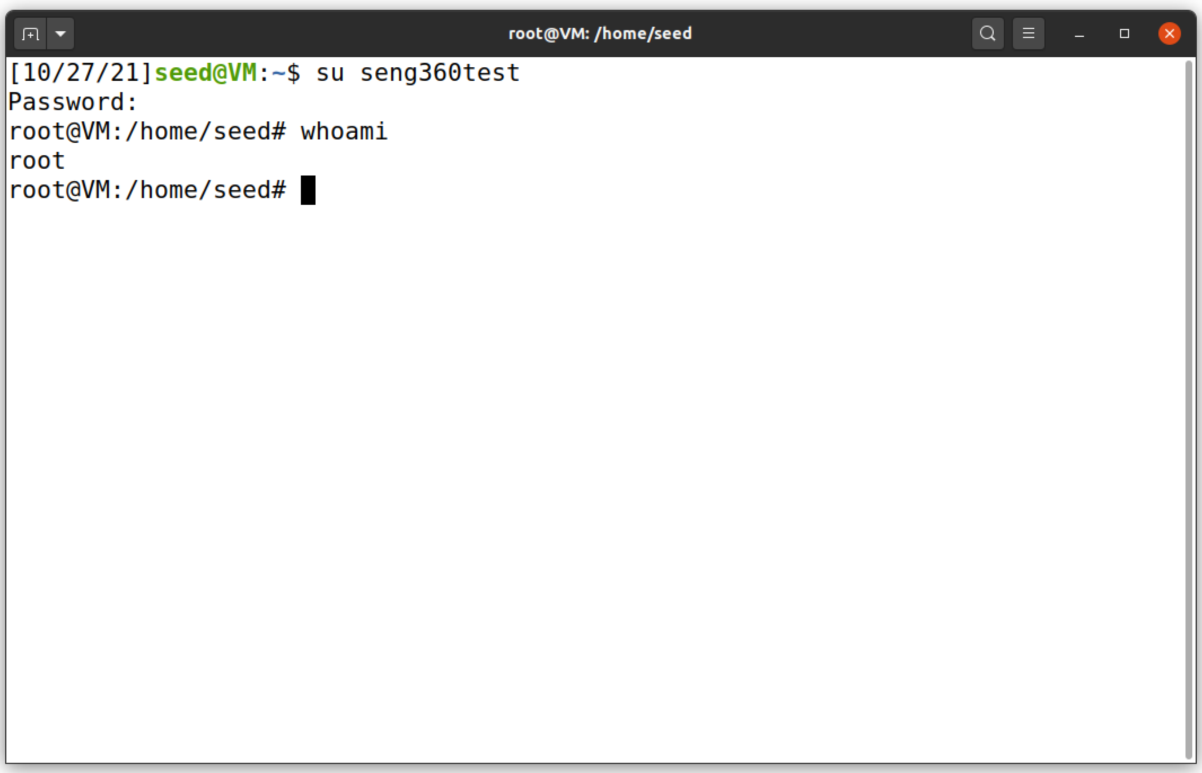
```
seed@VM: ~/.../Labsetup-2
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

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

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

    if (!access(fn, W_OK)) {
        sleep(10);
        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");
    }
-- INSERT --                                        16,12-19      Top
```

After completing the above attack, we remove the sleep() from the code and try to launch a real attack. To do this, we run the vulnerable program in parallel to the attack program. Since we would need to run the vulnerable program multiple times, we write a script to automate this process. The script exits as soon as the target file gets modified.

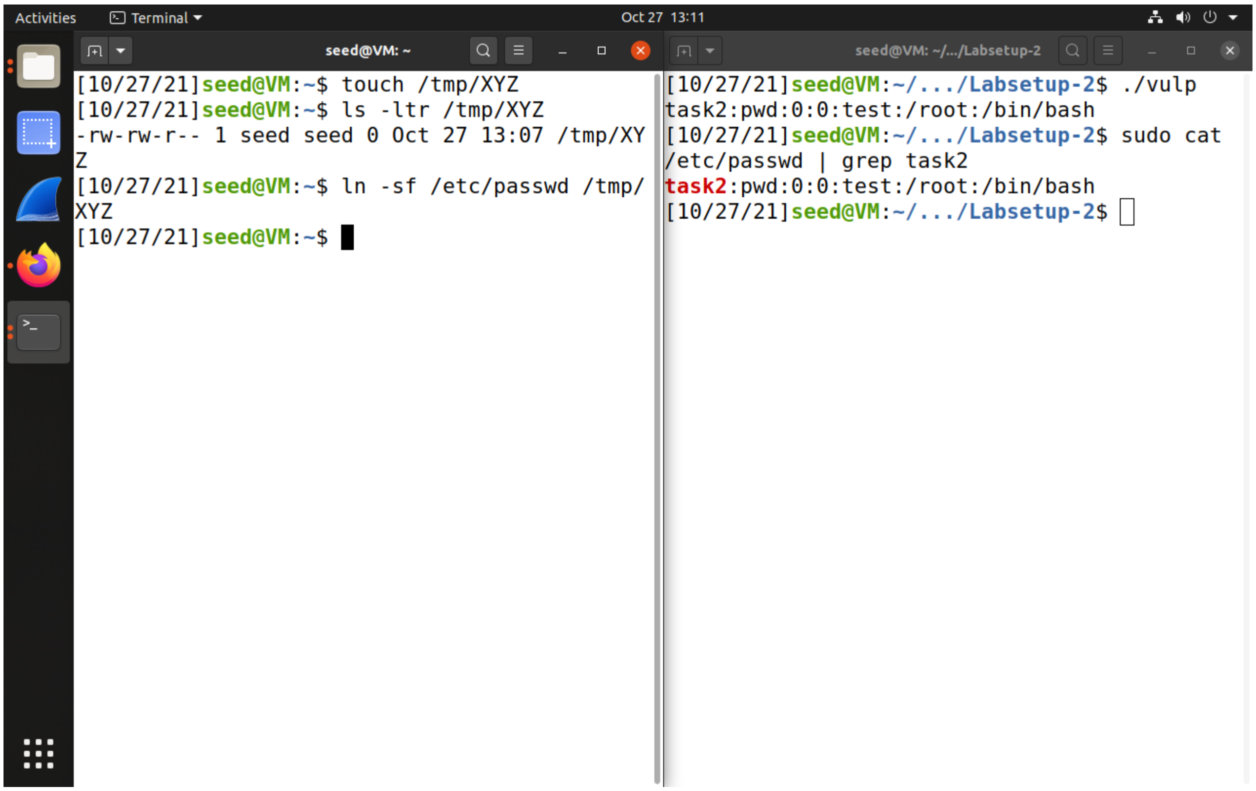
Lastly, we modify our attack to improve its effectiveness and counter a vulnerability in our own attack program. Then we again run the vulnerable program and the improved attack in parallel.

Results and Discussion

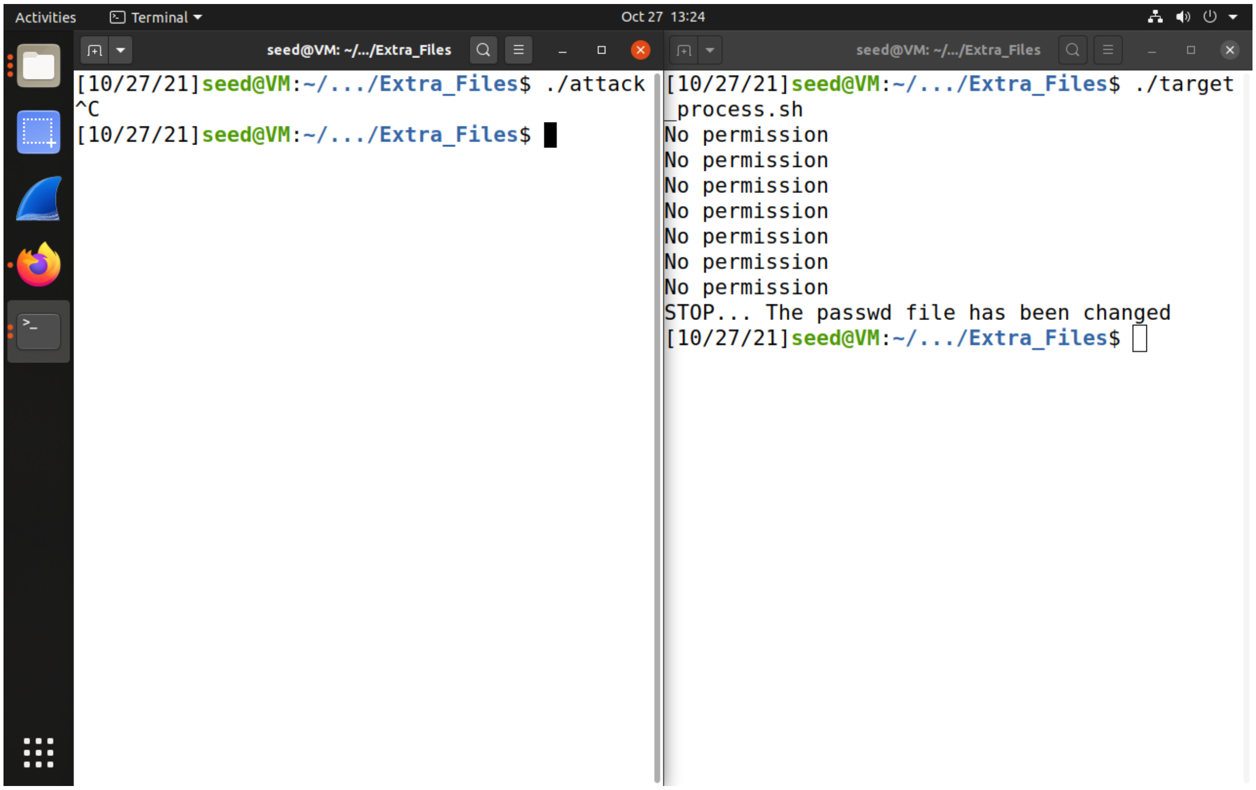
After modifying the passwd file, we login onto the seng360test account and notice do not have to enter any password. We also confirm we have root privileges for the user.



Now we attempt a manual attack. For this we let the vulnerable program sleep between the access and fopen command. While the program is sleeping the control is yielded to the OS, thus allowing us to manually execute the attack.



The above attack assumes that the system is slow, which would rarely be the case. Thus, we try to simulate an actual attack by running the automated script and the attack in parallel.


Activities
Terminal
Oct 27 13:24
seed@VM: ~/.../Extra_Files
[10/27/21]seed@VM:~/.../Extra_Files$ ./attack
^C
[10/27/21]seed@VM:~/.../Extra_Files$
seed@VM: ~/.../Extra_Files
[10/27/21]seed@VM:~/.../Extra_Files$ ./target_process.sh
No permission
No permission
No permission
No permission
No permission
No permission
No permission
STOP... The passwd file has been changed
[10/27/21]seed@VM:~/.../Extra_Files$

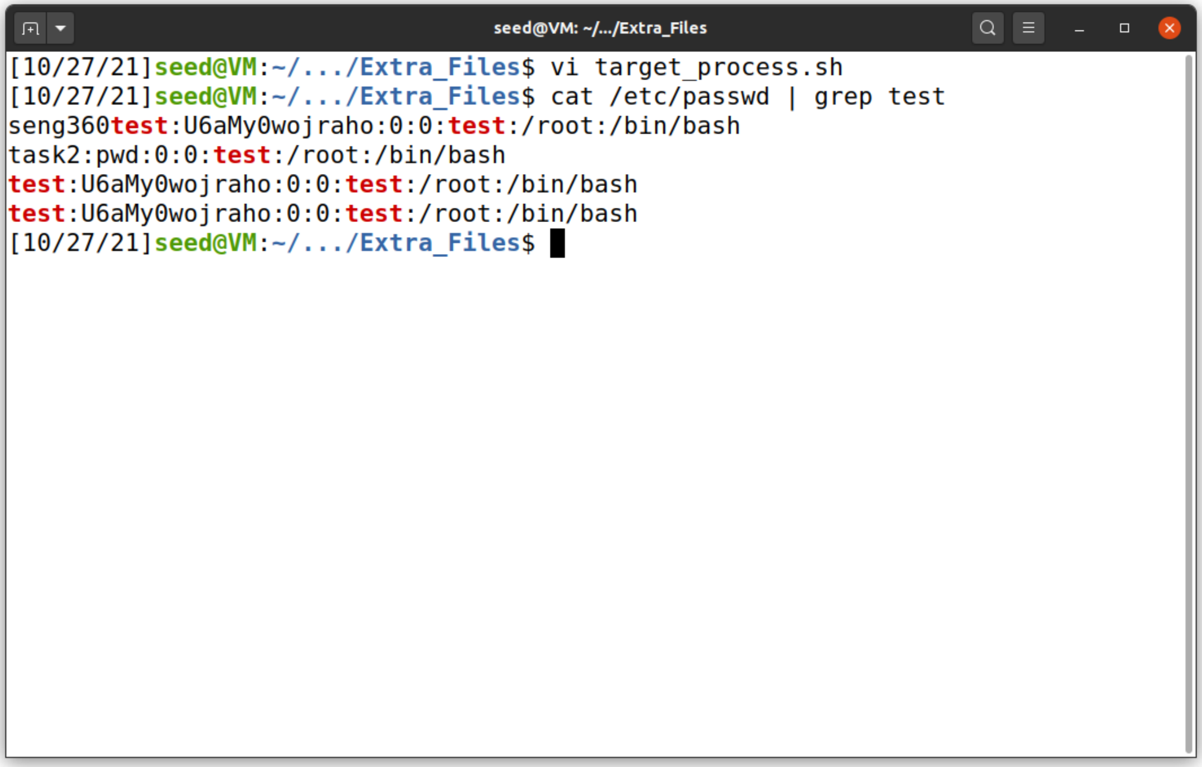

root@VM: /home/seed/Desktop/LAb6 Seng360/Extra_Files
[10/27/21]seed@VM:~/.../Extra_Files$ su test
Password:
root@VM:/home/seed/Desktop/LAb6 Seng360/Extra_Files# whoami
root
root@VM:/home/seed/Desktop/LAb6 Seng360/Extra_Files#

This attack took multiple attempts to succeed, some attempts taking upwards of 5 min before having to manually force exit it.

(Question for report: Explain in your own words, why the attack in Task 2C makes the attack more reliable.)

We blame this behavior on the fact that there is a race condition in our attack program itself. In our attack we unlink() the target file and then create a new symbolic link using symlink() between the target file and /etc/passwd file. But after executing unlink(), the attack program gets context-switched out and the vulnerable program has a chance to fopen(). After which, executing symlink() will have no effect at all.

To counter this we make improvements to our attack program, by making unlink and symlink atomic. We accomplish this by using the system call renameat2. This eliminates the race-condition.



Question for report: Explain how the Ubuntu countermeasures (which you switched off) work against these kinds of attacks.)

Firstly, restricting who can follow a symlink. This helps in situations when the attack program calls symlink before the vulnerable program calls fopen. This countermeasure will block the vulnerable program fopen call, as the symlink owner is different then the follower.

Secondly, preventing root from writing to files in /tmp that are owned by others. The effect of this countermeasure is quite obvious as the entire point of our attack was to get root access to modify a file in /tmp