Computer System Security **CS3312**

计算机系统安全

2024年 春季学期

主讲教师：张媛媛 副教授

上海交通大学 计算机科学与技术系

第十三章

条件竞争

Race Condition

目录/CONTENTS

01

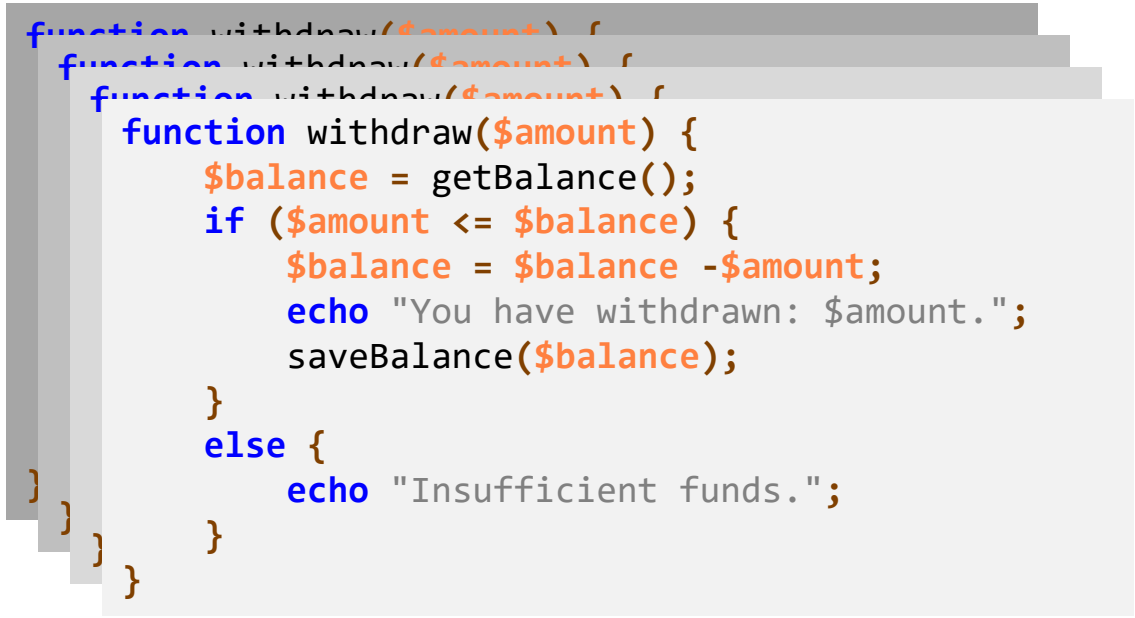
条 件 竞 争

Race condition

条件竞争

竞争条件漏洞是多个进程访问同一资源时产生的时间或者序列的冲突，并利用这个冲突来对系统进行攻击。一个看起来无害的程序如果被恶意攻击者利用，将发生竞争条件漏洞。

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        echo "You have withdrawn: $amount.";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



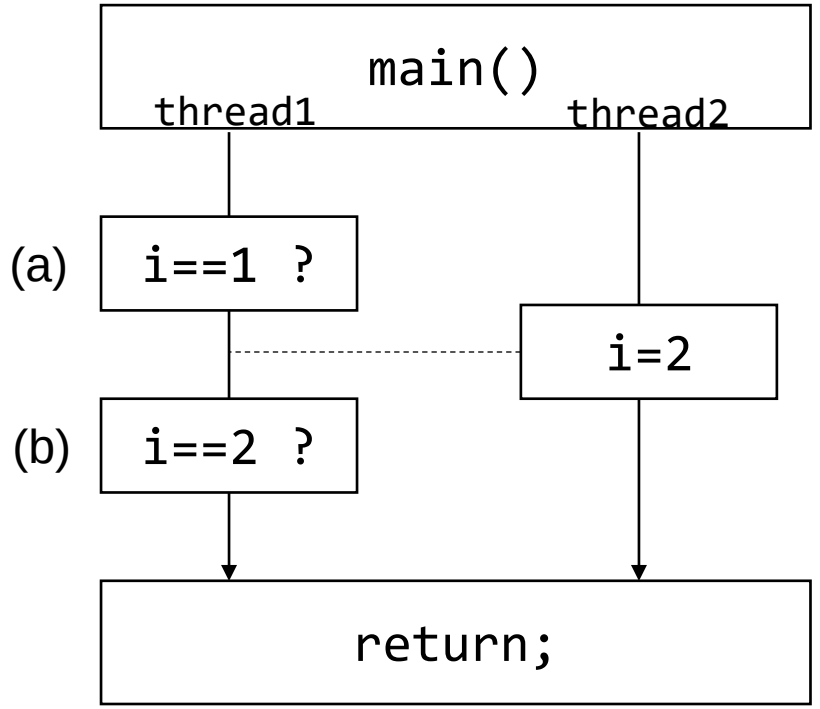
多进程情况

条件竞争代码展示

```c
//myThreadTest
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>

int i = 1;
void *mythread1() {
    if(i == 1){                    (a)
        sleep(3);
        if(i == 2)                 (b)
            printf("hack it!\n");
        else
            printf("you can try again!\n");
    }
}

void *mythread2() {
    sleep(1);
    i=2;
}

int main(int argc, const char *argv[]) {
    pthread_t id1,id2;
    pthread_create(&id1, NULL, (void *)mythread1,NULL);
    pthread_create(&id2, NULL, (void *)mythread2,NULL);
    pthread_join(id1,NULL);
    pthread_join(id2,NULL);
    return 0;
}
```

单独看 mythread1()函数，应输出 “you can try again!\n”。但由于 i 是全局共享的资源，其他线程执行的方式可能使 i 值在（a）执行之后且（b）执行之前发生改变，从而改变函数的流程。

```
root@ubuntu: /home/IS308# ./a.out
hack it!
root@ubuntu: /home/IS308#
```



漏洞成因

条件竞争发生在如下情况：

- 多个进程(多个线程)同时访问和操作相同的数据
- 执行的结果取决于特定的顺序

本节将介绍条件竞争类别下的两类代表性漏洞：

- Dirty Copy-on-Write (Dirty Cow, 脏牛)
- TOCTTOU

注意：　当具有条件竞争漏洞的程序是一个特权程序（SetUID） 时，攻击者可以通过对不可控事件施加影响来改变特权程序的输出，达到更严重的攻击效果。

02

Dirty CoW

脏 牛 攻 击

Copy-on-Write 写时复制

写入时复制（Copy-on-write，简称COW）是一种计算机程序设计领域的优化策略。

其核心思想是，如果有多个调用者（callers）同时要求相同资源（如内存或磁盘上的数据存储），他们会共同获取相同的指针指向相同的资源，直到某个调用者试图修改资源的内容时，系统才会真正复制一份专用副本（private copy）给该调用者，而其他调用者所见到的最初的资源仍然保持不变。

COW技术使用场景举例：

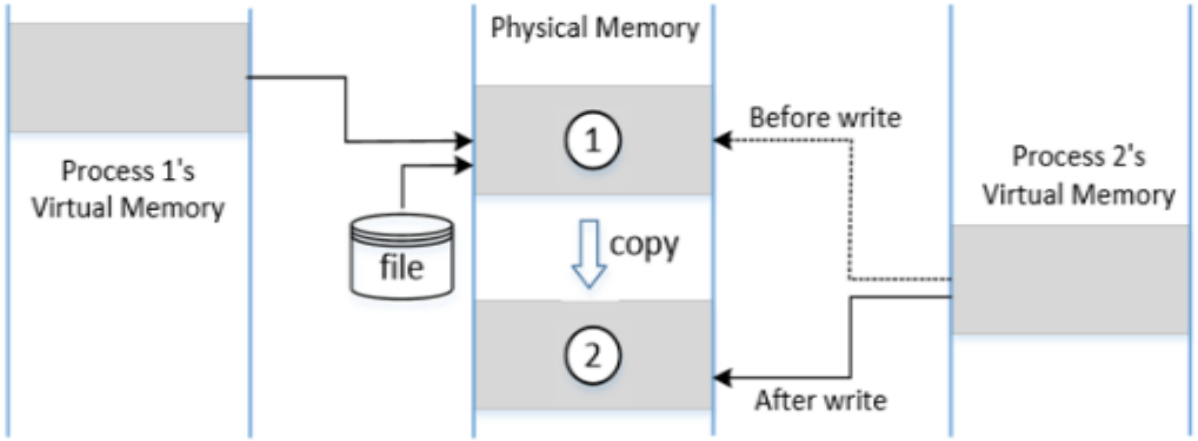
- 文件系统
- Linux轻量级子进程创建
- 高级程序语言

Dirty CoW

写时复制，包含三个步骤：

- 制作映射内存的副本
- 更新页表，使得虚拟内存指向新创建的物理内存
- 写入内存

三个步骤都不是原子态的，
任何一个步骤都可能被其他进程/线程打断，从而产生潜在的竞争条件，导致Dirty Cow漏洞。




```
// map the entire file to memory
map = mmap(NULL, st.st_size, PROT_READ|PROT_WRITE, MAP_SHARED, f, 0)
```

Dirty CoW

2016年10月18日，黑客Phil Oester提交了隐藏长达9年之久的“脏牛漏洞（Dirty COW）”0day漏洞。2016年10月20日，Linux内核团队成员、Linux的创始人Linus修复了这个 0day漏洞。

该漏洞是Linux内核的内存子系统在处理写时拷贝（Copy-on-Write)时存在条件竞争漏洞，导致可以破坏私有只读内存映射。黑客可以获取低权限的本地用户后，利用此漏洞获取其他只读内存映射的写权限，进一步获取root权限。

漏洞编号：CVE-2016-5195

漏洞名称：脏牛（Dirty COW）

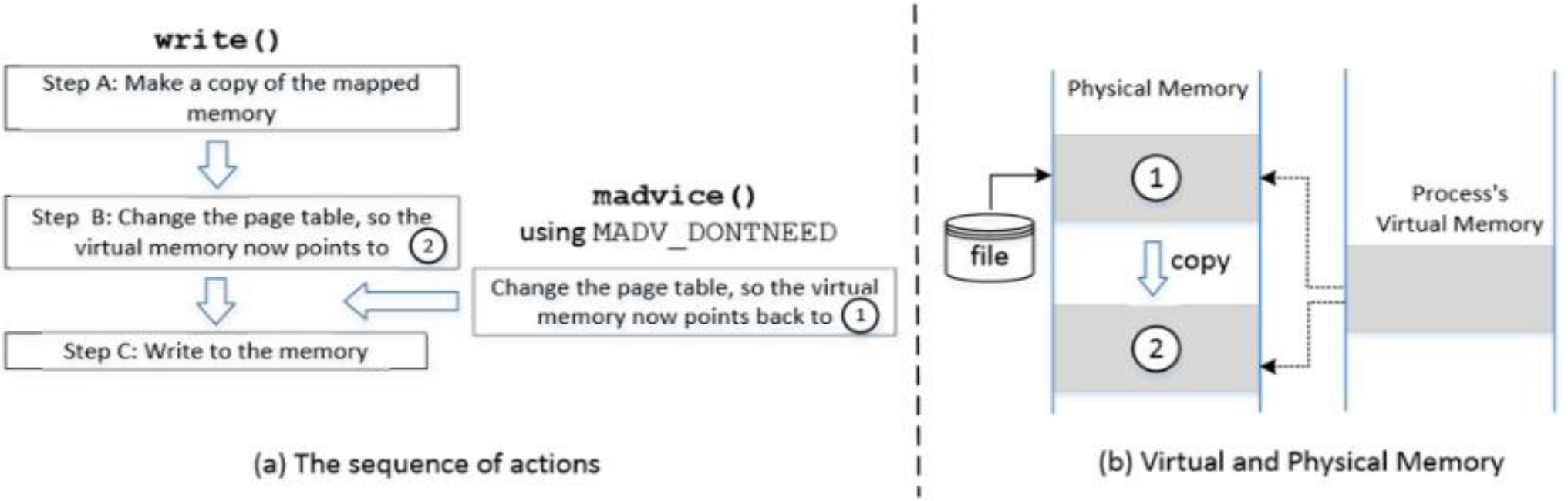
漏洞危害：本地提权

黑客可以通过远程入侵获取低权限用户后，在服务器操作系统本地利用该漏洞实现本地提权，从而获取到服务器root权限。

影响范围：Linux kernel>2.6.22 (released in 2007)的所有Linux系统**

漏洞利用条件：
黑客可以通过远程入侵获取低权限用户后,才能进一步在操作系统本地利用该漏洞。

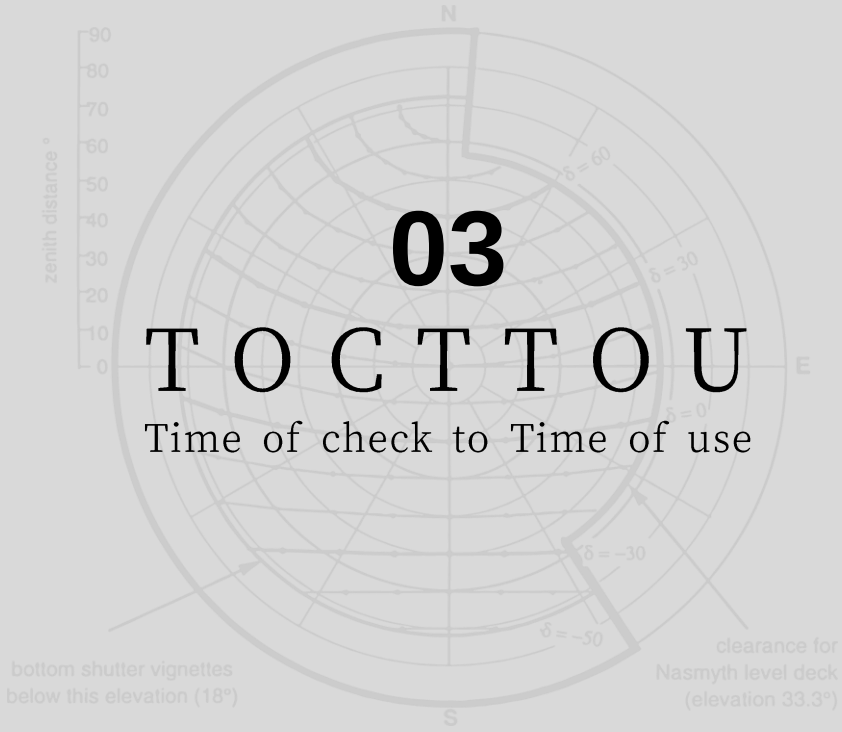
Dirty CoW


write()
Step A: Make a copy of the mapped memory
Step B: Change the page table, so the virtual memory now points to 2
Step C: Write to the memory
madvice()
using MADV_DONTNEED
Change the page table, so the virtual memory now points back to 1
(a) The sequence of actions
Physical Memory
1
copy
2
file
Process's Virtual Memory
(b) Virtual and Physical Memory

03

T O C T T O U

Time of check to Time of use



举个栗子

例程：一个含有条件竞争漏洞的SetUID程序

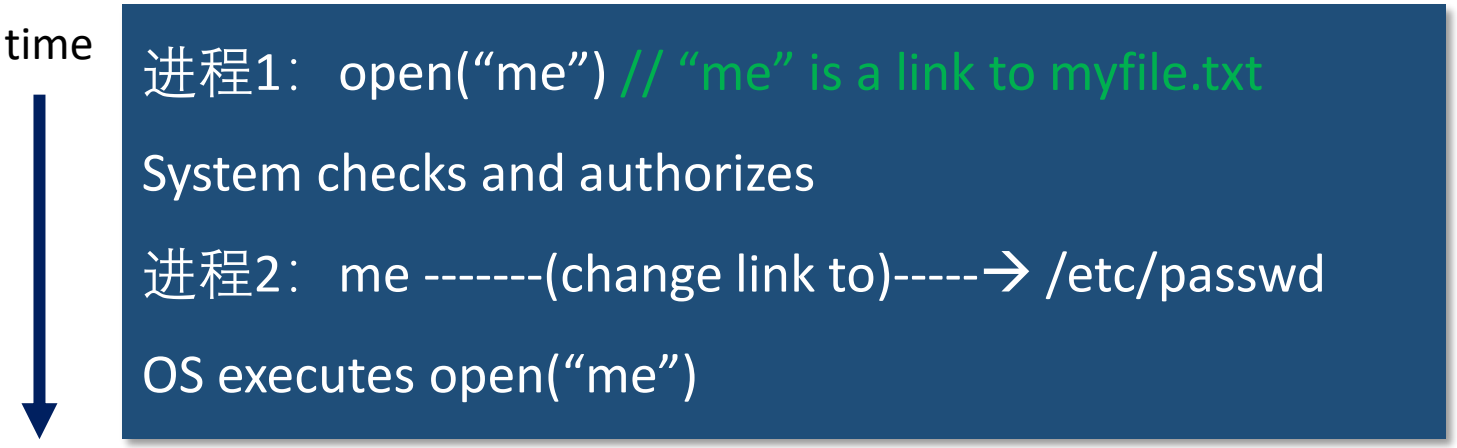
```
> ls -l
rwsrw-r--
```

```
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");
}
```

TOCTTOU：
Time of Check to Time of Use
仍然是一个打时间差的攻击，发生时机在check之后，use之前

Notes

Warning: Using **access**() to check if a user is authorized to, for example, open a file before actually doing so using ***open***(2) creates a security hole, because the user might exploit the short time interval between checking and opening the file to manipulate it. **For this reason, the use of this system call should be avoided**. (In the example just described, a safer alternative would be to temporarily switch the process's effective user ID to the real ID and then call ***open***(2).)


time
进程1： open(“me”) // “me” is a link to myfile.txt
System checks and authorizes
进程2： me -------(change link to)-----→ /etc/passwd
OS executes open(“me”)

另一个例子

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

if (fileExist == False) {
    /* The file does not exist, create it. */
    f = open(file, C_CREAT);
    // write to file
}
```

A Flawed Program（SetUID）

```
#include <unistd.h>

int main() {
    while (1) {
        unlink("/tmp/X");
        symlink("/home/jess/UserOwnedFile", "/tmp/X");
        usleep(10000);
        unlink("/tmp/X");
        symlink("/etc/passwd", "/tmp/X");
        usleep(10000);
    }
    return 0;
}
```

Attacking Script

讨论

The following programs are Set-UID programs that run with the root's privileges.
Do they have an exploitable race condition vulnerability?
If yes, please describe your attack; otherwise, please explain why.

```
01  if (!access("/etc/shadow", W_OK)) {
02      f = fopen("/etc/shadow", O_WRITE);
03      write_to_file(f);
04  } else {
05      fprintf(stderr, "Permission denied. \n");
06  }
```

```
01  int flag;
02  //......
03  if (flag == 0) {
04      write_to_file(f);
05  } else {
06      //print out error
07  }
```

04

条件竞争漏洞缓解

Countermeasures

增加检查次数

```c
#include <fcntl.h>
#include <stdio.h>

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

    //Three TOCTOU Windows:
    if (access("tmp/XYZ", O_RDWR)) {
        fprintf();
        return -1;
    }
    else fd1 = open("/tmp/XYZ", O_RDWR);

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

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

    //check whether fd1, fd2, fd3 has the same i-node (using fstat)

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

    if (stat1.st_ino == stat1.st_ino && stat1.st_ino == stat1.st_ino) {
        write_to_file(fd1);
    } else {
        fprintf(stderr, "Race condition detected. \n");
    }
}
```

操作原子化

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

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

使用这个选项 exclusively create only when this file doesn't exist.
因此，程序中无需再check existence，而消除了攻击窗口。

```c
/* disable the root privilege */

uid_t real_uid = getuid(); //get the 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);
```

讨论

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?

*The code to the left is a set-UID program.

```
01  int main()
02  {
03      Disable privilege(seteuid(ruid));
04      foo();
05      Enable privilege(seteuid(euid));
06  }
07
08  int foo()
09  {
10      //...
11      strcpy(buffer, ...);
12      //...
13      return 0;
14  }
```

本章要点

- 条件竞争是逻辑漏洞的一种
  - 由于需要处理共享资源且多方操作之间有时间差，使得攻击者能从中获利

- 条件竞争中有两类知名漏洞：
  - Dirty CoW
    - 对共享内存的Copy-on-Write加以利用，需要内核对操作进行原子化处理
  - TOCTTOU
    - 对文件系统检验和使用的时间差，可通过程序编写技巧避免