1 RaceCondition

```c
//gcc chall.c -m32 -pie -fstack-protector-all -o chall
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include <pthread.h>

unsigned int a = 0;
unsigned int b = 0;
unsigned int a_sleep = 0;
int flag = 1;
int pstr1 = 1;
int ret1;
pthread_t th1;
void * th_ret = NULL;

void menu_go(){
      if(a_sleep == 0){
            a = a + 5;
     }else{
            a_sleep = 0;
     }
      b = b + 2;
}

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

void menu_test(){
      if( b>a ){
            puts("Win!");
            system("/bin/sh");
```

```
45              exit(0);
46      }else{
47              puts("Lose!");
48              exit(0);
49      }
50  }
51
52  void menu_exit(){
53      puts("Bye");
54      exit(0);
55  }
56
57  void menu(){
58      printf("***** race *****\n");
59      printf("*** 1:Go\n*** 2:Chance\n*** 3:Test\n*** 4:Exit \n");
60      printf("************************************\n");
61      printf("Choice> ");
62      int choose;
63      scanf("%d",&choose);
64      switch(choose)
65      {
66              case 1:
67                      menu_go();
68                      break;
69              case 2:
70                      ret1 = pthread_create(&th1, NULL, menu_chance, &pstr1);
71                      break;
72              case 3:
73                      menu_test();
74                      break;
75              case 4:
76                      menu_exit();
77                      break;
78              default:
79                      return;
80      }
81      return;
82  }
83
84  void init(){
85      setbuf(stdin, 0LL);
86      setbuf(stdout, 0LL);
87      setbuf(stderr, 0LL);
88      while (1)
89      {
90              menu();
91      }
```

```
}

int main(){
      init();
      return 0;
}
```

这是一道条件竞争题目，最终的目标是启动shell，我们从目标反推攻击方法。

要想启动shell，就需要执行 `menu_test` ，并且通过第一个if检测，其检测条件为 `b>a` 。 `b` 与 `a` 是两个全局变量，且初始值为0，观察源代码，发现只有 `menu_go` 函数对这两个变量进行了修改。

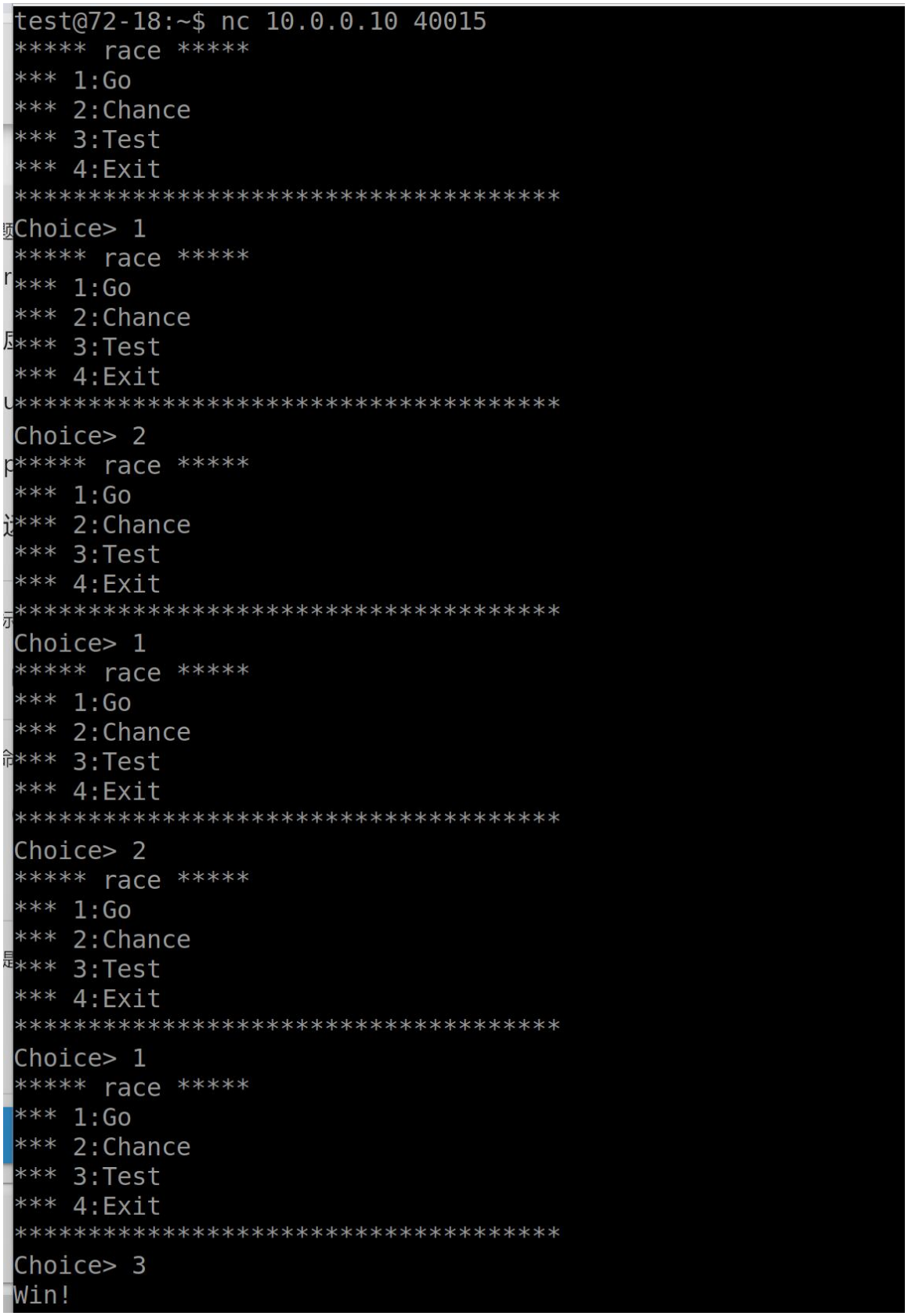
每次调用 `menu_go` 函数， `b` 一定被加2，而 `a` 则与 `a_sleep` 变量有关，如果 `a_sleep` 为0，则 `a` 增加5，否者 `a` 不变并将 `a_sleep` 设置为0。也就是说，如果我们要让 `b>a` 成立，就需要在 `a_sleep==0` 的时候执行 `menu_go` 函数。注意到 `a_sleep` 的初始值为0，所以我们需要找到能修改 `a_sleep` 的函数，也就是 `menu_chance` 。

`menu_chance` 首先会检查 `a<=b` ，如果满足则直接返回，否则会检查 `flag == 1` ，满足条件就把 `a_sleep` 设置为0，并在1秒后将 `flag` 设置为0，之后再调用 `menu_chance` 则会因为 `flag` 不为1而返回。

为了使 `menu_chance` 修改 `a_sleep` ，我们必须先执行一次 `menu_go` ，此时 `a=5,b=2` 。对于单线程的情况，我们在执行过一次 `menu_chance` 后，这个函数就不再有效，所以我们只有一次机会在 `a_sleep==0` 的时候执行 `menu_go` 函数，这显然不足以让我们通过 `b>a` 的条件检测。

但是注意到 `pthread_create(&th1, NULL, menu_chance, &pstr1)` ，这说明 `menu_chance` 函数是在一个新的线程上执行的，在 `menu_chance` 函数线程 `sleep` 期间，我们依然可以在主线程操作。更重要的是， `menu_chance` 函数会先修改 `asleep` 然后在 `sleep` 结束后才修改 `flag` ，也就是说在 `sleep` 期间我们还可以继续执行 `menu_chance` 修改 `a_sleep` ，每次 `a_sleep` 被设置为1后立即执行 `menu_go` 就可以做到只增加 `b` 。

所以我们的执行步骤为首先输入1，然后快速输入 `2 1 2 1` ，再输入3即可完成攻击。进入shell后使用 `cat flag` 查看flag为 `flag{race_r3ce_Race!!!!}` 。


test@72-18:~$ nc 10.0.0.10 40015
***** 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> 2
***** race *****
*** 1:Go
*** 2:Chance
*** 3:Test
*** 4:Exit
****
Choice> 1
***** race *****
*** 1:Go
*** 2:Chance
*** 3:Test
*** 4:Exit
****
Choice> 3
Win!