reverse

encrypt

IDA打开，一堆Windows api

```
ProcessHeap = GetProcessHeap();
v5 = HeapAlloc(ProcessHeap, 0, v7);
if ( v5 )
{
  if ( BCryptGetProperty(phAlgorithm, L"BlockLength", v26, 4u, &pcbResult, 0) >= 0 )
  {
    v9 = *v26;
    v10 = GetProcessHeap();
    v11 = HeapAlloc(v10, 0, v9);
    v6 = v11;
    if ( v11 )
    {
      memcpy(v11, &unk_7FF7963034A0, *v26);
      v12 = 8i64;
      *pbInput = _mm_xor_si128(_mm_load_si128(&xmmword_7FF796303500), _mm_loadu_si128(pbInput));
      do
        *&pbInput[2 * v12++] ^= 0x55u;
      while ( v12 < 15 );
      if ( BCryptSetProperty(phAlgorithm, L"ChainingMode", pbInput, 0x20u, 0) >= 0
        && BCryptGenerateSymmetricKey(phAlgorithm, &phKey, v5, *pbOutput, &pbSecret, 16u, 0) >= 0
        && BCryptExportKey(phKey, 0i64, L"OpaqueKeyBlob", 0i64, 0, &cbOutput, 0) >= 0 )
      {
        v13 = cbOutput;
        v14 = GetProcessHeap();
        v15 = HeapAlloc(v14, 0, v13);
        if ( v15 )
        {
          if ( BCryptExportKey(phKey, 0i64, L"OpaqueKeyBlob", v15, cbOutput, &cbOutput, 0) >= 0 )
```

首先先解释一下各个api的作用：

BCryptOpenAlgorithmProvider: 打开一个密码算法提供者。

BCryptGetProperty: 获取密码算法提供者的属性。在这里，它获取了`ObjectLength`和`BlockLength`两个属性。`ObjectLength`属性用于分配内存，`BlockLength`属性用于设置密码算法的块长度。

BCryptSetProperty: 设置密码算法提供者的属性。在这里，它设置了对称加密的链接模式（Chaining Mode）

BCryptGenerateSymmetricKey: 生成对称密钥。

BCryptExportKey: 导出密钥。

BCryptEncrypt: 加密数据。

BCryptDestroyKey: 销毁密钥对象。

BCryptCloseAlgorithmProvider: 关闭算法提供者。

GetProcessHeap: 获取当前进程的堆句柄。

HeapAlloc: 在堆中分配内存。

HeapFree: 释放堆中的内存。

通过对这些api的解释，我们可以知道这个程序就是通过使用Windows api来生成一个加密体系来对flag进行加密

看到最后：

```
          && BCryptDestroyKey(phKey) >= 0 )
        {
          phKey = 0i64;
          v20 = GetProcessHeap();
          HeapFree(v20, 0, v3);
          v3 = 0i64;
          if ( !memcmp(v4, &unk_7FF796305050, v28) )
            puts("right flag!");
        }
      }
    }
  }
```

那么这个`unk_7FF796305050`应该就是密文了

现在还需要知道加密模式，动调起来应该可以知道

动调后进入到算法提供的`pszAlgId`

```
v34 = 83;
*pszAlgId = 4522049;
*pbInput = _mm_load_si128(&xmmword_7FF78E6634F0);
si128 = _mm_load_si128(&xmmword_7FF78E6634E0);
if ( BCryptOpenAlgorithmProvider(&phAlgorithm, pszAlgId, 0i64, 0) >= 0
  && BCryptGetProperty(phAlgorithm, L"ObjectLength", pbOutput, 4u, &pcbResult, 0) >= 0 )
{
  v7 = *pbOutput;
  ProcessHeap = GetProcessHeap();
```

```
AF db    0
B0 db  41h ; A
B1 db    0
B2 db  45h ; E
B3 db    0
B4 db  53h ; S
B5 db    0
B6 db    0
B7 db    0
B8 db  43h ; C
B9 db    0
BA db  68h ; h
BB db    0
BC db  61h ; a
```

```
DCE db   65h ; e
DCF db    0
DD0 db   43h ; C
DD1 db    0
DD2 db   42h ; B
DD3 db    0
DD4 db   43h ; C
DD5 db    0
```

可以知道该加密算法是aes，模式是cbc

那么现在还需要找到key和iv

`BCryptGenerateSymmetricKey`是生成key

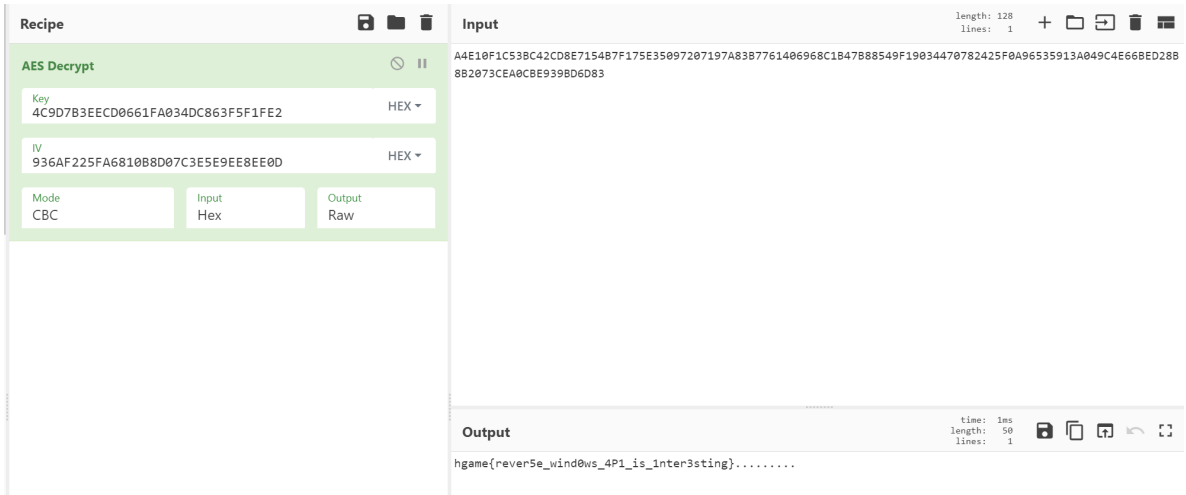
```
if ( BCryptSetProperty(phAlgorithm, L"ChainingMode", pbInput, 0x20u, 0) >= 0
  && BCryptGenerateSymmetricKey(phAlgorithm, &phKey, v5, *pbOutput, &pbSecret, 16u, 0) >= 0
  && BCryptExportKey(phKey, 0i64, L"OpaqueKeyBlob", 0i64, 0, &cbOutput, 0) >= 0 )
{
```

那么key就在`pbSecret`里

```
if ( BCryptGetProperty(phAlgorithm, L"BlockLength", v26, 4u, &pcbResult, 0) >= 0 )
{
  v9 = *v26;
  v10 = GetProcessHeap();
  v11 = HeapAlloc(v10, 0, v9);
  v6 = v11;
  if ( v11 )
  {
    memcpy(v11, &unk_7FF78E6634A0, *v26);
```

iv则猜测在memcpy对`&unk_7FF78E6634A0`的比较中

最后使用cyberchef解aes



mystery

IDA打开，通过strings定位到关键函数

```
LOAD:000055… 0000000A     C     GLIBC_2.4
LOAD:000055… 0000000C     C     GLIBC_2.2.5
LOAD:000055… 0000001C     C     _ITM_deregisterTMCloneTable
LOAD:000055… 0000000F     C     __gmon_start__
LOAD:000055… 0000001A     C     _ITM_registerTMCloneTable
.rodata:000… 00000019     C     please input your flag:\n
.rodata:000… 00000012     C     Congratulations!\n
.rodata:000… 00000018     C     Wrong!please try again!
.eh_frame:0… 00000006     C     :*3$\"
.data:00005… 00000006     C     PB8MLT
.data:00005… 00000007     C     DJVDJV
```

```
int sub_55EA42FEB100()
{
  puts("please input your flag:\n");
  __isoc99_scanf();
  memset(&keybox, 0, 256uLL);
  rc4(&keybox, &key, strlen(&key));
  sub_55EA42FEB500(&keybox, s1, strlen(s1));
  if ( !strcmp(s1, s2) )
    return puts("Congratulations!\n");
  else
    return puts("Wrong!please try again!");
}
```

首先第一个函数就是生成一个keybox

```
  v11 = __readfsqword(0x28u);
  memset(v10, 0, 0x400uLL);
  for ( i = 0LL; i != 256; ++i )
  {
    *(a1 + i) = i;
    v10[i] = *(a2 + i % a3);
  }
  v4 = 0LL;
  v5 = 0;
  do
  {
    v6 = *(a1 + v4);
    v7 = (v10[v4] + v6 + v5) >> 31;
    v5 = (HIBYTE(v7) + LOBYTE(v10[v4]) + v6 + v5) - HIBYTE(v7);
    v8 = (a1 + v5);
    *(a1 + v4++) = *v8;
    *v8 = v6;
  }
  while ( v4 != 256 );
  return __readfsqword(0x28u) ^ v11;
```

第二步则是使用生成的keybox对我们的输入进行加密

```
if ( a3 )
{
  i = &a2[a3];
  LOBYTE(v4) = 0;
  LOBYTE(v5) = 0;
  do
  {
    v5 = (v5 + 1);
    v6 = (a1 + v5);
    v7 = *v6;                           // v7 = key[i + 1]
    v4 = (*v6 + v4);
    v8 = (a1 + v4);
    *v6 = *v8;                          // key[i + 1] = key[i + v4]
    *v8 = v7;                           // key[i + v4] = key[i + 1]
    result = *(a1 + (*v6 + v7));        // reslut = key[i+1]+key[i+v4]
    *a2++ -= result;
  }
  while ( i != a2 );
}
return result;
```

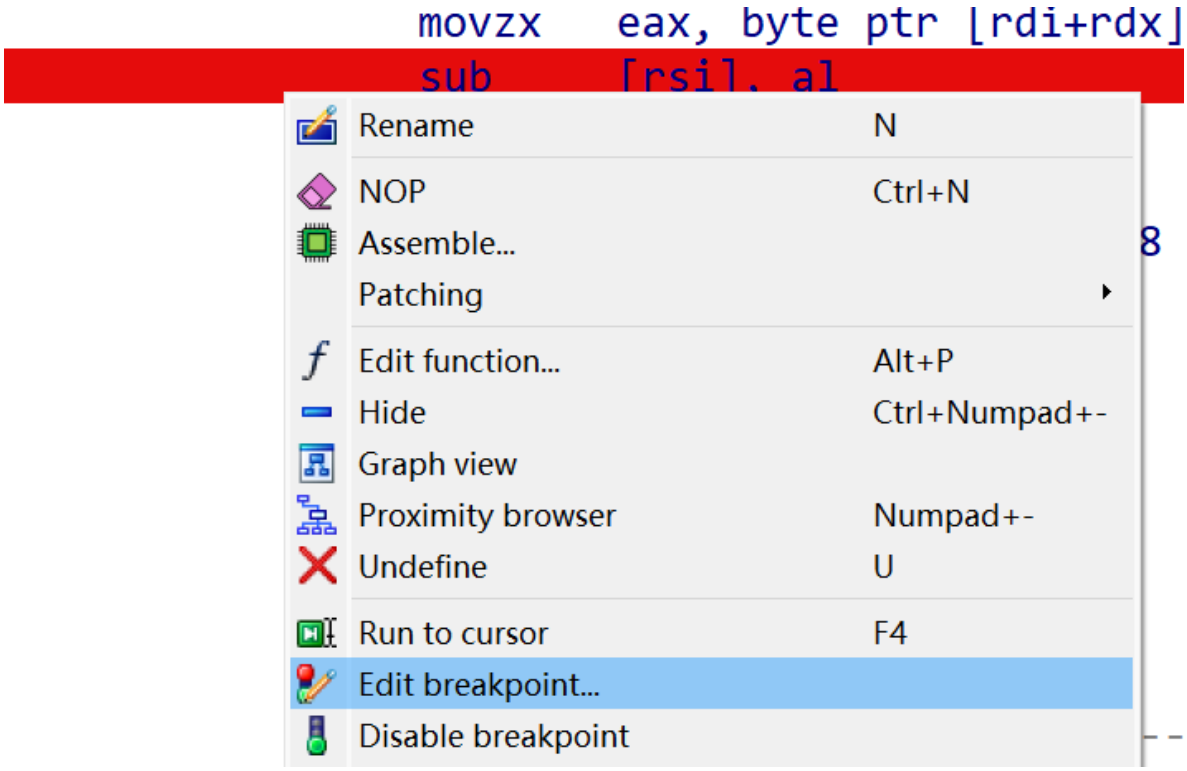
两个函数合起来就是一个清晰的rc4，加密和解密都是同样的脚本，只不过最后被改为`*a2++ -= result;`

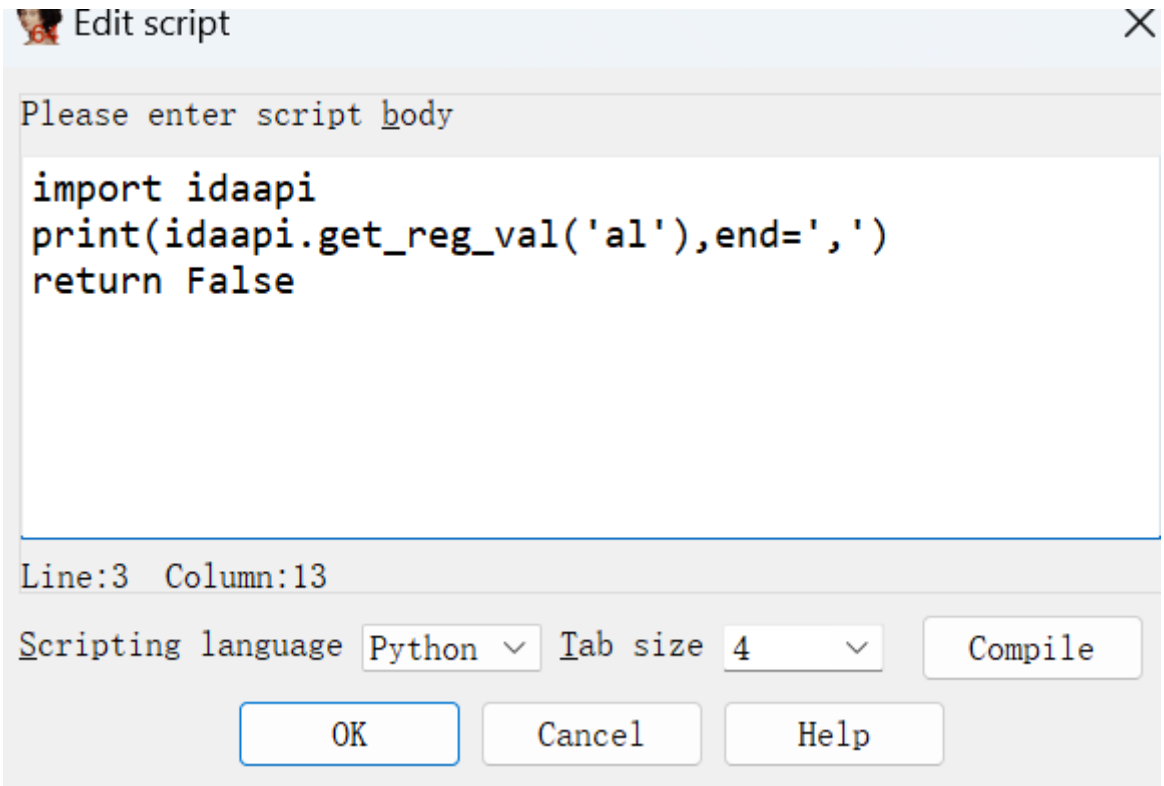
但是`result`可以通过动调获取，于是可以编写一个idapython的脚本，输出result

```
.text:000055EA42FEB542 0F B6 D2                 movzx   edx, dl
.text:000055EA42FEB545 0F B6 04 17              movzx   eax, byte ptr [rdi+rdx]
.text:000055EA42FEB549 28 06                    sub     [rsi], al
.text:000055EA42FEB54B 48 83 C6 01              add     rsi, 1
.text:000055EA42FEB54F 49 39 F2                 cmp     r10, rsi
.text:000055EA42FEB552 75 C4                    jnz     short loc_55EA42FEB518
```

首先在result转到指令下断点

然后选edit breakpoint





写下如下脚本

```
import idaapi
print(idaapi.get_reg_val('al'),end=',')
return False
```

最后动调，随便给一个输入，就自动输出result了

然后就是解密，s2是密文，key也给出

exp：

```
v = [24, 37, 41, 32, 25, 39, 185, 201, 52, 199, 113, 201, 172, 23, 180, 30, 229,
233, 252, 42, 74, 1, 234, 121, 199,
     130, 254, 81, 231, 177, 174, 40, 21, 172, 45, 155, 21, 111, 57, 15, 200,
235, 72, 160, 41, 248, 25, 63, 1, 231,
     139, 31, 143, 15, 189, 246]
s = [0x50, 0x42, 0x38, 0x4D, 0x4C, 0x54, 0x90, 0x6F, 0xFE, 0x6F,
     0xBC, 0x69, 0xB9, 0x22, 0x7C, 0x16, 0x8F, 0x44, 0x38, 0x4A,
     0xEF, 0x37, 0x43, 0xC0, 0xA2, 0xB6, 0x34, 0x2C]
for i in range(len(s)):
    print(chr((s[i] + v[i])%256),end='')
```

findme

IDA打开，进入到main函数

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  sub_140001010("hgame{It_is_a_fake_flag!HaHaHa}\n");
  sub_140001010("you should try to decrypt it:\n");
  sub_140001010("aGdhbWV7SXRfaXNfYWxzb19hX2Zha2VfZmxhZyFIYUhhSGFIYX0=");
  puts(Buffer);
  return 0;
}
```

下面的base64解码也是一个假的flag

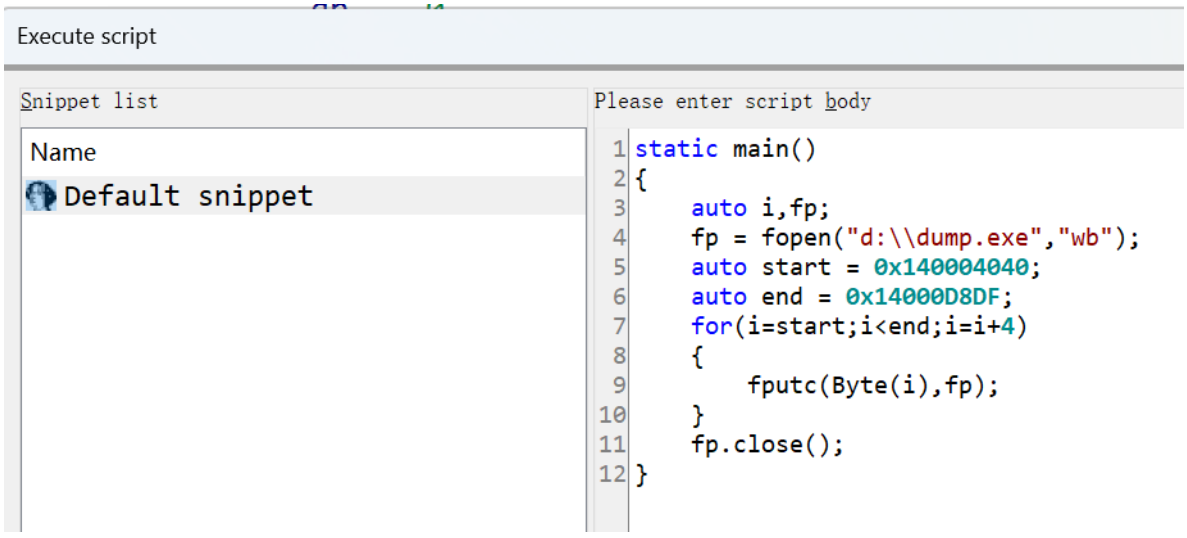
往下看 看到buffer

```
; char Buffer[2]
Buffer db 'M',0
align 4
aZ db 'Z',0
align 8
db  90h
db    0
db    0
db    0
db    0
db    0
db    0
db    0
db    3
db    0
db    0
db    0
db    0
db    0
db    0
db    0
```

`mz.....`像是一个exe文件头，根据题目描述，一堆奇怪的数据

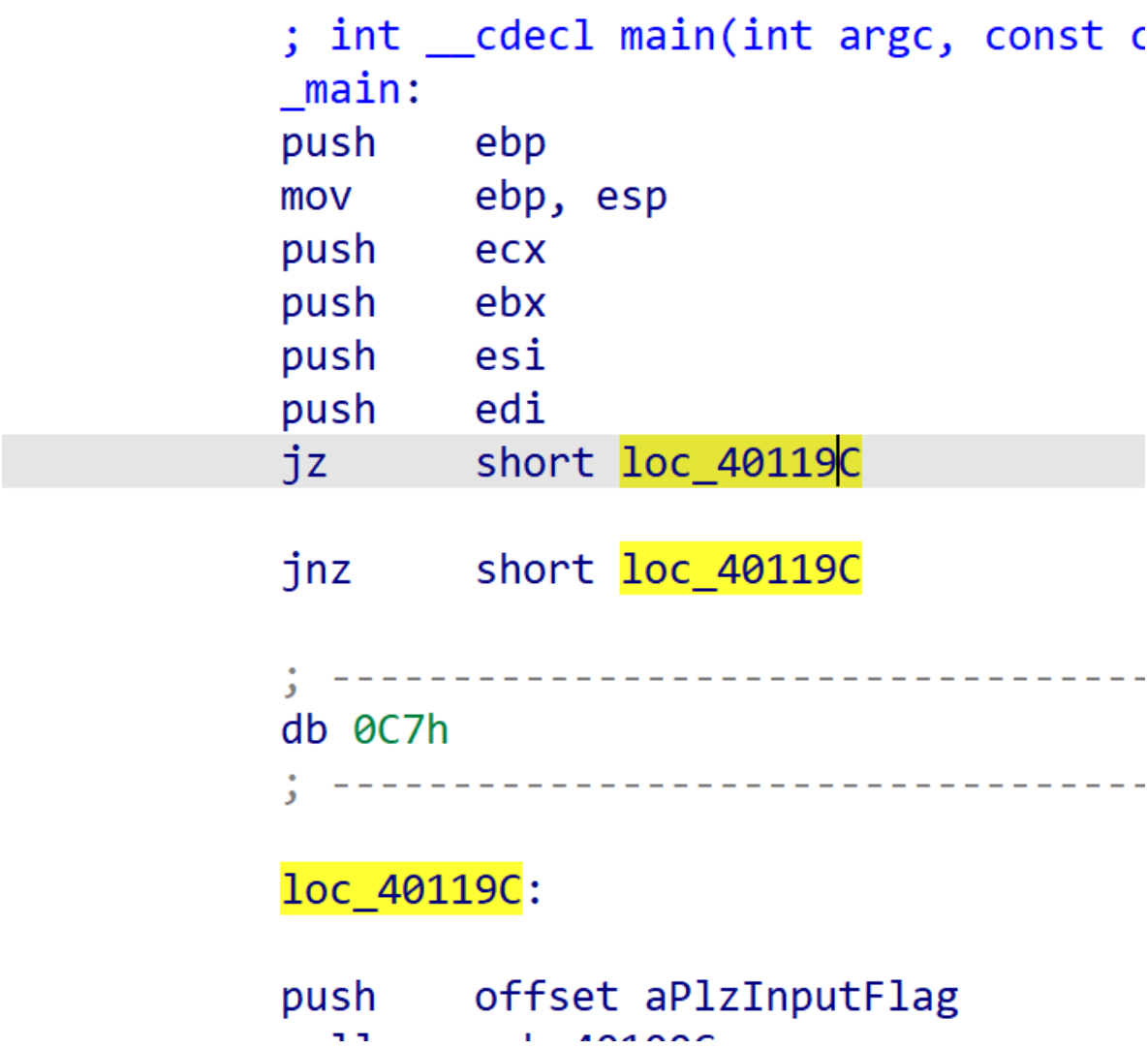
那么有可能这段数据就是题目的真正的exe，现在把这段数据dump出来

编写脚本



```
static main()
{
    auto i,fp;
    fp = fopen("d:\\dump.exe","wb");
    auto start = 0x140004040;
    auto end = 0x14000D8DF;
    for(i=start;i<end;i=i+4)
    {
        fputc(Byte(i),fp);
    }
    fp.close();
}
```

dump出来后打开得到的exe



果然这才是真正的exe

发现有许多jz jnz构成的花指令，手动patch掉后对main函数重新定义，则可以f5查看伪代码

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int v3; // ecx
  char v5; // [esp+4h] [ebp-10h]
  char v6; // [esp+4h] [ebp-10h]

  sub_64100C("plz input flag:\n", v5);
  sub_64103A("%32s", input);
  sub_641068(strlen(aDeadbeef));
  sub_64110C(strlen(input));
  v3 = 0;
  while ( input[v3] == byte_642148[v3] )
  {
    if ( ++v3 >= 32 )
    {
      sub_64100C("Congratulations!", v6);
      return 0;
    }
  }
  sub_64100C("Sry...try again", v6);
  return 0;
```

查看 `sub_641068` 函数和 `sub_64110C` 函数

```
int i; // ecx
int v2; // ebx
int j; // esi
unsigned __int8 v4; // dl
char result; // al
int v6[256]; // [esp+Ch] [ebp-400h]

memset(v6, 0, sizeof(v6));
for ( i = 0; i < 256; ++i )
{
  byte_643390[i] = -i;
  v6[i] = aDeadbeef[i % a1];
}
v2 = 0;
for ( j = 0; j < 256; ++j )
{
  v4 = byte_643390[j];
  v2 = (v4 + v6[j] + v2) % 256;
  result = byte_643390[v2];
  byte_643390[j] = result;
  byte_643390[v2] = v4;
}
return result;
```

```
int v1; // ebx
unsigned int v2; // edi
int v3; // esi
unsigned __int8 v4; // cl
char result; // al

v1 = 0;
v2 = 0;
if ( a1 )
{
  v3 = 0;
  do
  {
    v1 = (v1 + 1) % 256;
    v4 = byte_643390[v1];
    v3 = (v4 + v3) % 256;
    byte_643390[v1] = byte_643390[v3];
    byte_643390[v3] = v4;
    result = input[-(v4 + byte_643390[v1])];
    input[v2++] += result;
  }
  while ( v2 < a1 );
}
return result;
```

又是一个rc4，类似上面`mystery`题

那么用idapython脚本导出result

```
import idaapi
print(idaapi.get_reg_val('al'),end=',')
return False
```

最后用`密文-result`得到flag

exp：

```
v = [0x7D, 0x2B, 0x43, 0xA9, 0xB9, 0x6B, 0x93, 0x2D, 0x9A, 0xD0,
  0x48, 0xC8, 0xEB, 0x51, 0x59, 0xE9, 0x74, 0x68, 0x8A, 0x45,
  0x6B, 0xBA, 0xA7, 0x16, 0xF1, 0x10, 0x74, 0xD5, 0x41, 0x3C,
  0x67, 0x7D]
s =
[21,196,226,60,84,240,77,193,106,89,21,86,120,242,24,119,65,9,52,224,249,65,72,1
76,127,220,13,99,224,206,243,0]
for i in range(len(v)):
    c = v[i]-s[i]
    if c<0:
        print(chr(c+256),end='')
    else:
        print(chr(c),end='')
```

crypto

exRSA

attachment.py

```
from Crypto.Util.number import *
from secret import flag
m=bytes_to_long(flag)
p=getStrongPrime(1024)
q=getStrongPrime(1024)
phi=(p-1)*(q-1)
e1=inverse(getPrime(768),phi)
e2=inverse(getPrime(768),phi)
e3=inverse(getPrime(768),phi)
n=p*q
c=pow(m,0x10001,n)
print(f'e1={e1}')
print(f'e2={e2}')
print(f'e3={e3}')
print(f'c={c}')
print(f'n={n}')

"""
e1=507704823781196942747311122537087612252896744705655189912361346179268800289678839430419291761056414976625223228157699029348523968414531087693099791896007081696882915037687595340542080958626715317171749619833686108952370183209832228450193114288981757581676170504495170553084932792884984815864303069336314375706322058471492589396558796704213755780726115411791635851947796464529347197506336205069030635362749298086100843976536583762265797795806985328805630725316750988325812294988227702166531780725330890635567047217234617117726768806495939718692610398725955158662796540697911819348552752097674849072846016794905528953 9
e2=125268482983490053905202769239291324634591525749986257572082592978911151336541176482157829453325290813652738603162011307933065707777350765347721689997058956412075353038394550740030576878103811109783209889760113261069199407991609742283118247600463702735055110656192685576971825862592343792394104827844981573233529439567630222641686370934003298761271515191608429182109546262582102313356041532582488534722139149693721324636173636127084674112855759560305271361252845370994840310071127767964121852042987889756565548208641057637997140478921229769755374829243818306550099337504003173382549669279769936242101027159951026940 1
"""
```

```
e3=12985940757578530810519370332063658344046688856605967474941014436872720360444
040464644790980976991393970947023398357422203873284294843401144065013911463670501
155988860114510865196109834825082416669766552841766837440881457295972278902011030
962450762755535058785656035094662207102192600377838492764753972834210687160886380
186994778153542817681963059581651103563578804145156157584336712678882995685632610
568685398017604768332697428389634332298152115021131759757155454248892129015812260
341405711480367328938080641190483288551340547091208778959416701664216648061867100
346824494054783025733475898081247824887967550418509038276279
c=141417606015230184211049709802459718924625917201933541490012745209823394304182
592602851743707531629494335532394745892801055691290913973928292425550664730569680
729078989504731085564173501997831453496910872559262873632869220118411433395308630
300198239231490707393383076174791818994158815857391930802936280447588808440607410
537739133660453344009979384923785724755758230739132932051599602182000035556051420
175056435870269949185883111271435668580366533159851775519638364297285157456468070
123637193259859856630452155138986610272067480257330592146135108190083578873094130
311444005086084419225944109323678700271573793234284714739 9
n=178533037338380661731104178905937044641468248863164567808733525599697426157552
944666644395293527184343995528186353527680335319480097371706975662868487108328000
426311328560924133698481653594007727877031506265706341560810588064209681809146597
572126173303463125668183837840427667101827234752823747483792944536893070188010357
576447851214333201478653969853522013978444031448137146405395476982273840780816194
946943216714729685820896972467020893493349051243983390018762076812868678098172416
646569155028537284640299199579434901583886822168621639659732727311016592278981431
158584620497062552540667240129258151004349538218568545297 53
phi=178533037338380661731104178905937044641468248863164567808733525599697426157552
944666644395293527184343995528186353527680335319480097371706975662868487108328000
426311328560924133698481653594007727877031506265706341560810588064209681809146597
572126173303463125668183837840427667101827234752823747483792944536893802927070181
771490686318736012878592343518366616230003156532617279650769801240788457082470798
285098309787313234015003029546069389983917646339464213270307973083920197245464323
265774603130048384907423910358440436321967414553635119220303427088297964135321132
951478390454424777782514867778927855950169251946270628
"""
```

考点是rsa多组低解密指数攻击，直接上脚本

exp：

```
#sage
from Crypto.Util.number import long_to_bytes
import gmpy2


N =
178533037338380661731104178905937044641468248863164567808733525599697426157552944
666644395293527184343995528186353527680335319480097371706975662868487108328004263
113285609241336984816535940077278770315062657063415608105880642096818091465975721
261733034631256681838378404276671018272347528237474837929445368930701880103576447
851214333201478653969853522013978444031448137146405395476982273840780816194694321
671472968582089697246702089349334905124398339001876207681286867809817241646569155
028537284640299199579434901583886822168621639659732727311016592278981431585846204
970625525406672401292581510043495382185685452975 3
```

```
e1 =
507704823781196942747311122537087612252896744705655189912361346179268800289678839430419291761056414976625223228157699029348523968414531087693099791896007081696882915037687595340542080958626715317171749619833686108952370183209832228450193114288981757581676170504495170553084932792884984815864303069336314375706322058471492589396558796704213755780726115411791635851947796464529347197506336205069030635362749298086100843976536583762265797795806985328805630725316750988325812294988227702166531780725330890635567047217234617117726768806495939718692610398725955158662796540697911819348552752097674849072846016794905528953 9
e2 =
125268482983490053905202769239291324634591525749986257572082592978911151336541176482157829453325290813652738603162011307933065707777350765347721689997058956412075353038394550740030576878103811109783209889760113261069199407991609742283118247600463702735055110656192685576971825862592343792394104827844498157323352943956763022264168637093400329876127151519160842918210954626258210231335604153258248853472213914969372132463617363612708467411285575956030527136125284537099484031007112776796412185204298788975656554820864105763799714047892122976975537482924381830655009933750400317338254966927976993624210102715995102694019 e3 =
129859407575785308105193703320636583440466888566059674749410144368727203604440404646447909809769913939709470233983574222038732842948434011440650139114636705015598886011451086519610983482508241666976655284176683744088145729597227890201103962450762755535058785656035094662207102192600377838492764753972834210687160886381869947781535428176819630595816511035635788041451561575843367126788829956856326156868539801760476833269742838963433229815211502113175975715545424889212901581226341405711480367328938080641190483288551340547091208778959416701664216648061867103468244940547830257334758980812478248879675504185090382762791 c =
141417606015230184211049709802459718924625917201933541490012745209823394304182592602851743707531629494335532394745892801055691290913973928292425550664730569687290789895047310855641735019978314534969108725592628736328692201184114333953086330019823923149070739338307617479181899415881585739193080293628044758880844060741537739133660453344009979384923785724755758230739132932051599602182000035556051421750564358702699491858831112714356685803665331598517755196383642972851574564680712363719325985985663045215513898661027206748025733059214613510819008357887309413311444005086084419225944109323678700271573793234284714739 9



alpha2 = 815./2048
M1 = int(gmpy2.mpz(N)**(3./2))
M2 = int( gmpy2.mpz(N) )
M3 = int(gmpy2.mpz(N)**(3./2 + alpha2))
M4 = int( gmpy2.mpz(N)**(0.5) )
M5 = int( gmpy2.mpz(N)**(3./2 + alpha2) )
M6 = int( gmpy2.mpz(N)**(1.+alpha2) )
M7 = int( gmpy2.mpz(N)**(1.+alpha2) )
D = diagonal_matrix(ZZ, [M1, M2, M3, M4, M5, M6, M7, 1])
B = Matrix(ZZ, [ [1, -N,   0,  N**2,   0,      0,      0,    -N**3],
                 [0, e1, -e1, -e1*N, -e1,      0,   e1*N,  e1*N**2],
                 [0,  0,  e2, -e2*N,   0,   e2*N,      0,  e2*N**2],
                 [0,  0,   0, e1*e2,   0, -e1*e2, -e1*e2, -e1*e2*N],
                 [0,  0,   0,     0,  e3,  -e3*N,  -e3*N,  e3*N**2],
                 [0,  0,   0,     0,   0,  e1*e3,      0, -e1*e3*N],
                 [0,  0,   0,     0,   0,      0,  e2*e3, -e2*e3*N],
                 [0,  0,   0,     0,   0,      0,      0, e1*e2*e3] ]) * D

L = B.LLL()
```

```
v = Matrix(ZZ, L[0])
x = v * B**(-1)
phi_ = (e1*x[0,1]/x[0,0]).floor()
d = inverse_mod( 65537, phi_)
m = hex(power_mod(c, d, N))[2:]
print(bytes.fromhex(m))
```

misc

与ai聊天

misc签到题

进网页让ai输出flag