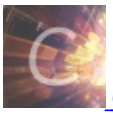
看雪2020 KCTF 子鼠开天writeup复现

原创

zakyi 于 2020-08-23 10:21:04 发布 291 收藏

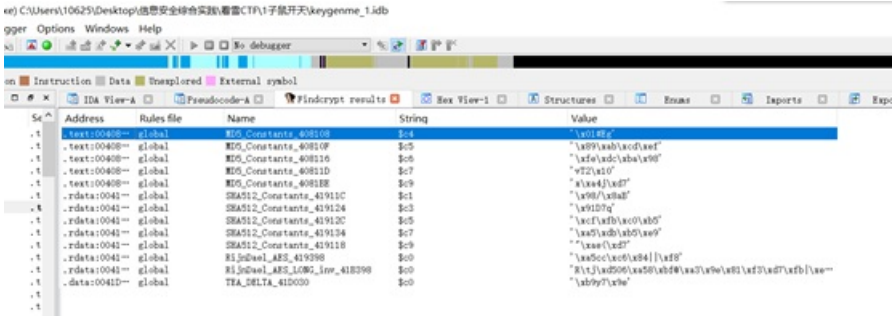
分类专栏： CTF 文章标签： 安全



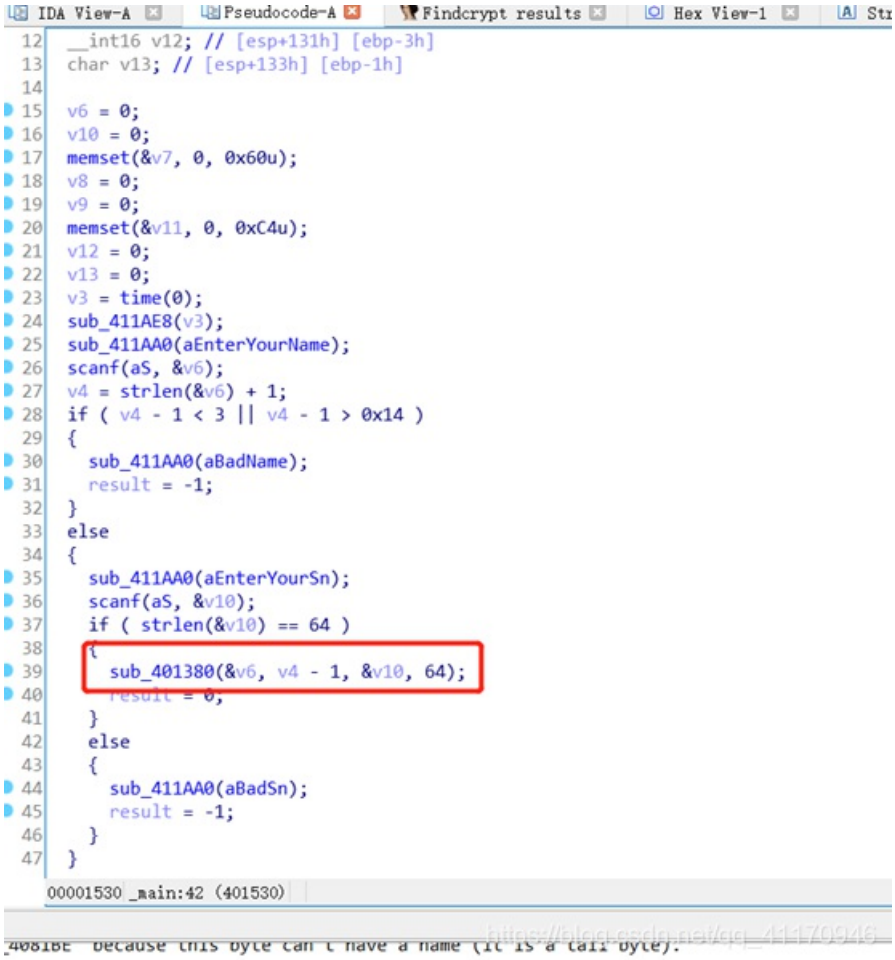
小学期的作业，也是第一次看这么长的writeup，放在这里当纪念。

1.

IDA PRO 7.0安装findcrypt插件后打开keygenme使用插件，



F5找到关键函数，可以看到传入的参数是name,name长度，sn，sn长度。且name的长度在3到20之间，sn长度为64.

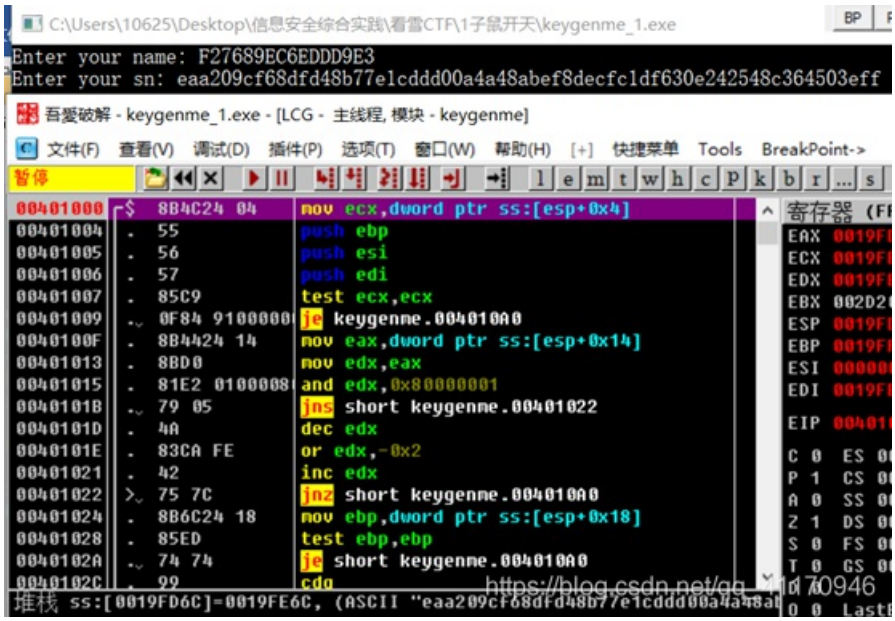


```
12  __int16 v12; // [esp+131h] [ebp-3h]
13  char v13; // [esp+133h] [ebp-1h]
14
15  v6 = 0;
16  v10 = 0;
17  memset(&v7, 0, 0x60u);
18  v8 = 0;
19  v9 = 0;
20  memset(&v11, 0, 0xC4u);
21  v12 = 0;
22  v13 = 0;
23  v3 = time(0);
24  sub_411AE8(v3);
25  sub_411AA0(aEnterYourName);
26  scanf(aS, &v6);
27  v4 = strlen(&v6) + 1;
28  if ( v4 - 1 < 3 || v4 - 1 > 0x14 )
29  {
30    sub_411AA0(aBadName);
31    result = -1;
32  }
33  else
34  {
35    sub_411AA0(aEnterYourSn);
36    scanf(aS, &v10);
37    if ( strlen(&v10) == 64 )
38    {
39      sub_401380(&v6, v4 - 1, &v10, 64);
40      result = 0;
41    }
42    else
43    {
44      sub_411AA0(aBadSn);
45      result = -1;
46    }
47  }
00001530 _main:42 (401530)
```

进入sub_401380函数（此图为分析到rsa算法后截，所以重命名了部分函数）

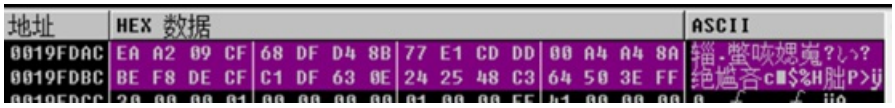
```
void __cdecl sub_401380(int name, unsigned int namelen, int sn, int snlen)
{
  char v4; // [esp+4h] [ebp-70h]
  char v5; // [esp+14h] [ebp-60h]
  char v6; // [esp+15h] [ebp-5Fh]
  char v7; // [esp+1Eh] [ebp-56h]
  char v8; // [esp+23h] [ebp-51h]
  char v9; // [esp+24h] [ebp-50h]
  char sn_; // [esp+34h] [ebp-40h]
  char aesout; // [esp+54h] [ebp-20h]

  if ( namelen >= 3 && namelen <= 0x14 && snlen == 64 )
  {
    if ( string_2_hex((_WORD *)sn, 64, (int)&sn_) != 32// 此函数将字符串转化为十六进制整数
      || (aes((int)&sn_, 32, (int)&aesout, (int)&unk_4190D0, 128, 0), rsa((int)&aesout, 32, (int)&v5), v5)// 此行调用了两个函数，另外v5需要=0
      || v6 != 2
      || v8
      || v7 != -120 )
    {
      sub_411AA0(aBadSn);
    }
    else
    {
      sub_401190(name, namelen, &v4);
      if ( !memcmp(&v4, &v9, 0x10u) )
        sub_411AA0(aCongratulation);
    }
  }
}
```

2.首先分析sub_401000（重命名后为string_2_hex），打开OD在00401000处设置断点，随机输入长度为64的sn和长度为3到20的name以后断在了401000



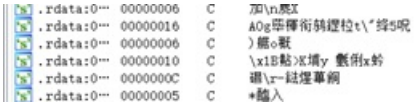
由于sub_4010F0使用了401000的返回值，又因为401000返回值是4010F0的第一个参数而且4010F0的调用约定是_cdecl，所以从右向左压栈，只要跟踪最后一个push进去的参数，就可以找到处理后的数据存在下图栈中,此图为调用sub_4010F0前eax指向的地址的值

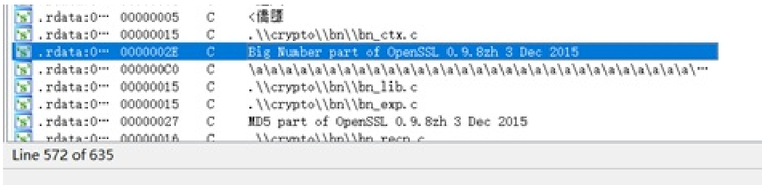
与401000函数内处理的数据存储的位置相同，下图为在401000内跟踪得到的堆栈数据



所以我们可以知道401000传入了一个字符型sn，处理后得到一个整数型sn_，
并且此sn_在004010F0中被使用。

3.接下来分析sub_4010F0（重命名后为aes）
首先搜索字符串，看看此程序是否用到了某些加密函数的开源库，首先想到了OPENSSL，使用字符串搜索函数果然搜索到了OPENSSL的踪迹





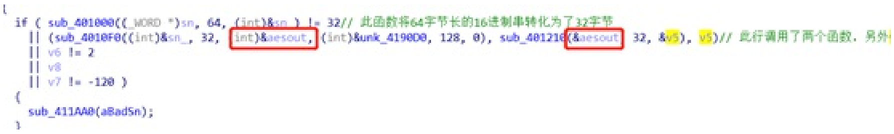
去github下载openssl 0.9.8的sig文件，放到ida根目录的sig-pc文件夹里，然后点view-subview-signatures，右键点击添加sig，搜索到openssl添加，然后进入sub_4010F0函数，查看到已经有一个函数被解析了

google找到AES_ecb_encrypt的定义

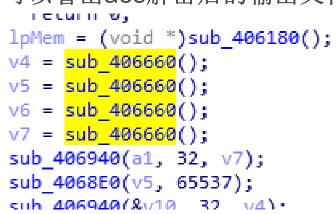
```
void AES_ecb_encrypt(const unsigned char *in, unsigned char *out,
                     const AES_KEY *key, const int enc) {

        assert(in && out && key);
    assert((AES_ENCRYPT == enc)||(AES_DECRYPT == enc));

    if (AES_ENCRYPT == enc)
        AES_encrypt(in, out, key);
    else
        AES_decrypt(in, out, key);
}
```

可知若a6 encAES_ENCRYPT1时，函数执行AES加密，a60时函数执行AES解密。

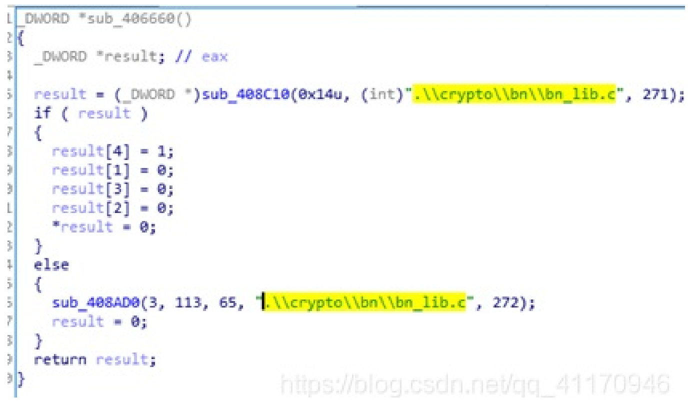
由于我们传进4010F0的enc为0，所以执行解密，解密后的数据放在a3里，回到main函数后成为了v11，我们重命名v11为aesout



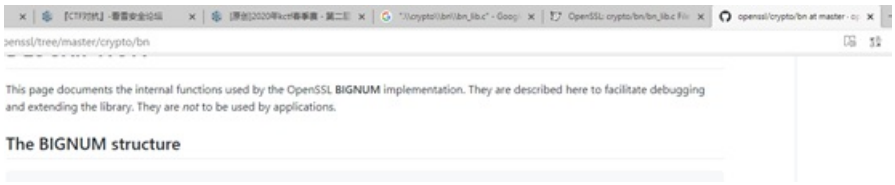
4.接下来分析sub_401210（重命名后为rsa），可以看出aes解密后的输出又传到401210



进入sub_401210的sub_406660，发现字符串".\crypto\bn\bn_lib.c"



谷歌一下，看到此文件存在于openssl的大数库中，那什么加密算法需要用到大数呢？推断此算法为RSA。而且sub_4068E0传入的参数中有65537也就是0x10001是RSA算法中常用的公钥e。



下图v7为传入的sn处理后的结果，猜测为明文。V5是0x10001转换为大整数的值。
V4为v10转换为大整数的值，所以v10应该是n。

```
sub_406940((unsigned __int8 *)sn, 32, v7);
sub_4068E0(v5, 65537);
sub_406940((unsigned __int8 *)&v10, 32, v4);
sub_406D40(v6, v7, v5, v4, lpMem);
```

```
v10 = 0x69;
v18 = 0x39;
v41 = 0x39;
v11 = 0x82u;
v12 = 0x30;
v13 = 0x28;
v14 = 0x57;
v15 = 0x74;
v16 = 0x65;
v17 = 0xABu;
v19 = 0x91u;
v20 = 0xDFu;
v21 = 4;
v22 = 0x51;
v23 = 0x46;
v24 = 0xF9u;
v25 = 0x1D;
v26 = 0x55;
v27 = 0x6D;
v28 = 0xEEu;
v29 = 0x88u;
v30 = 0x70;
v31 = 0x84u;
v32 = 0x5D;
v33 = 0x8Eu;
v34 = 0xE1u;
v35 = 0xCDu;
v36 = 0x3C;
v37 = 0xF7u;
v38 = 0x7E;
v39 = 0x4A;
v40 = 0xC;
```

整理后
n=0x69823028577465AB3991DF045146F91D556DEE8870845D8EE1CD3CF77E4A0C39
分解出两个质因数
p = 0x979BE0C9EECE7426C9FD28C2D6E7772B
q = 0xB22831D15714EB91CD83340B4837182B

d = (p-1)*(q-1)=
0x390A684CB713378FFD5CCE8C4000B5D6A2BB9F29B63D395E6BE6E9DD941527BD

所以只需要求出RSA的私钥(n,d)，加上密文解密后用aes解密就得到最后的答案。
那么密文去哪里找呢？
红框中，sub_401190处理name为v4，v4又和v9比较，若相同则成功。

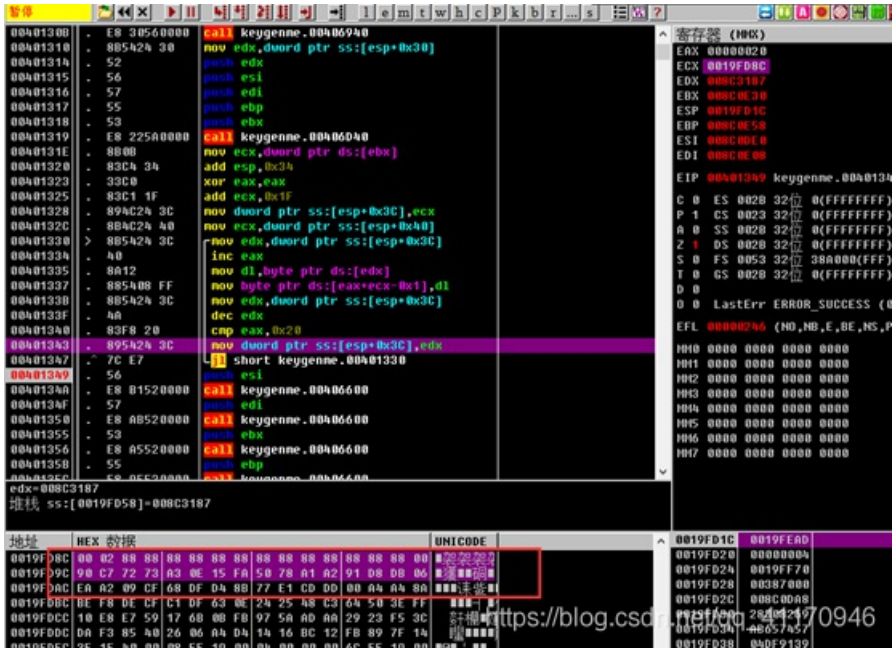
```
void __cdecl sub_401380(int name, unsigned int namelen, int sn, int snlen)
```

```
{
  char v4; // [esp+4h] [ebp-70h]
  char v5; // [esp+14h] [ebp-60h]
  char v6; // [esp+15h] [ebp-5Fh]
  char v7; // [esp+1Eh] [ebp-56h]
  char v8; // [esp+23h] [ebp-51h]
  char v9; // [esp+24h] [ebp-50h]
  char sn_; // [esp+34h] [ebp-40h]
  char aesout; // [esp+54h] [ebp-20h]

  if ( namelen >= 3 && namelen <= 0x14 && snlen == 64 )
  {
    if ( string_2_hex((_WORD *)sn, 64, (int)&sn_ ) != 32// 此函数将字符串转化为十六进制整数
      || (aes((int)&sn_, 32, (int)&aesout, (int)&unk_419000, 128, 0), rsa((int)&aesout, 32, (int)&v5), v5)// 此行调用了两个函数，另外v5需要=0
      || v6 != 2
      || v8
      || v7 != -120 )
    {
      sub_411AA0(aBadSn);
    }
    else
    {
      sub_401190(name, namelen, &v4);
      if ( !memcmp(&v4, &v9, 0x10u) )
        sub_411AA0(aCongratulation);
    }
  }
}
```

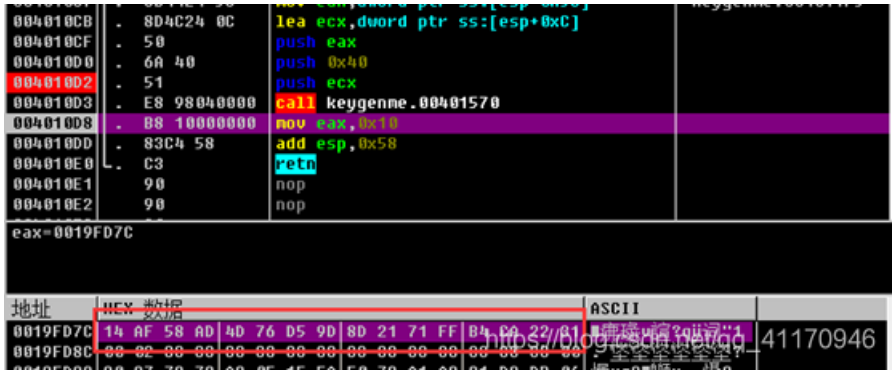

那么v9又是什么呢？

查看sub_401380的参数定义，v5是RSA加密后的密文（共256位），存放在esp+14h,v9存放在esp+24h，也就是密文的后128位。

跟踪rsa函数，得到处理后的密文为



5、接下来分析sub_401190，也就是处理name的函数,直接丢进OD动态调试，把处理后的name找出来，下图是跟踪到sub_401190的最后一个函数返回的eax指向的地址，即处理后的KCTF字符串。此字符串和RSA加密之后的后128位比较，若相同就成功。



6.获得sn

所以我们用上图红框内标注的后128位加上第四步rsa密文的前128位就可以构建出我们需要的密文了。也就是C=

288888888888888888888888880014AF58AD4D76D59D8D2171FFB4CA2231

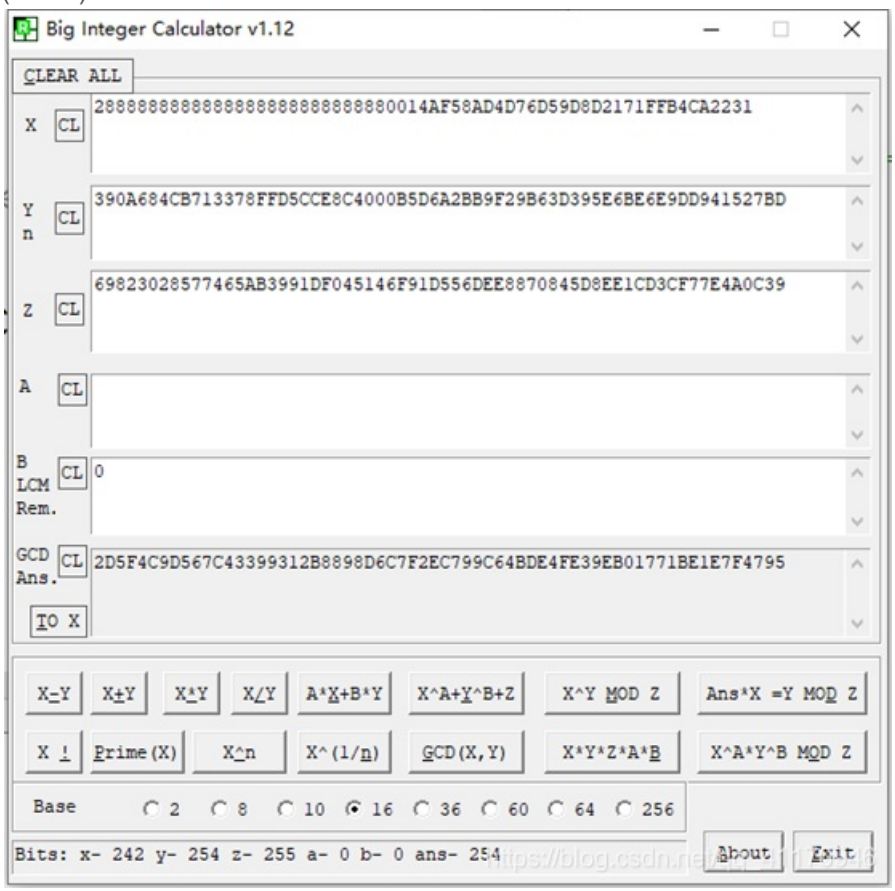
已知d =

0x390A684CB713378FFD5CCE8C4000B5D6A2BB9F29B63D395E6BE6E9DD941527BD

e = 0x10001

n = 0x69823028577465AB3991DF045146F91D556DEE8870845D8EE1CD3CF77E4A0C39

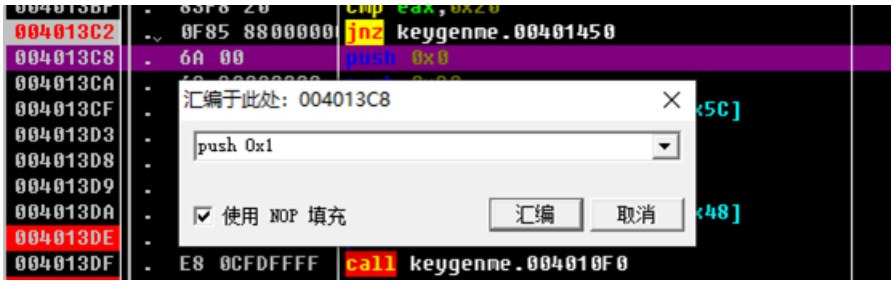
rsa解密算法： cd ≡ m (mod n)

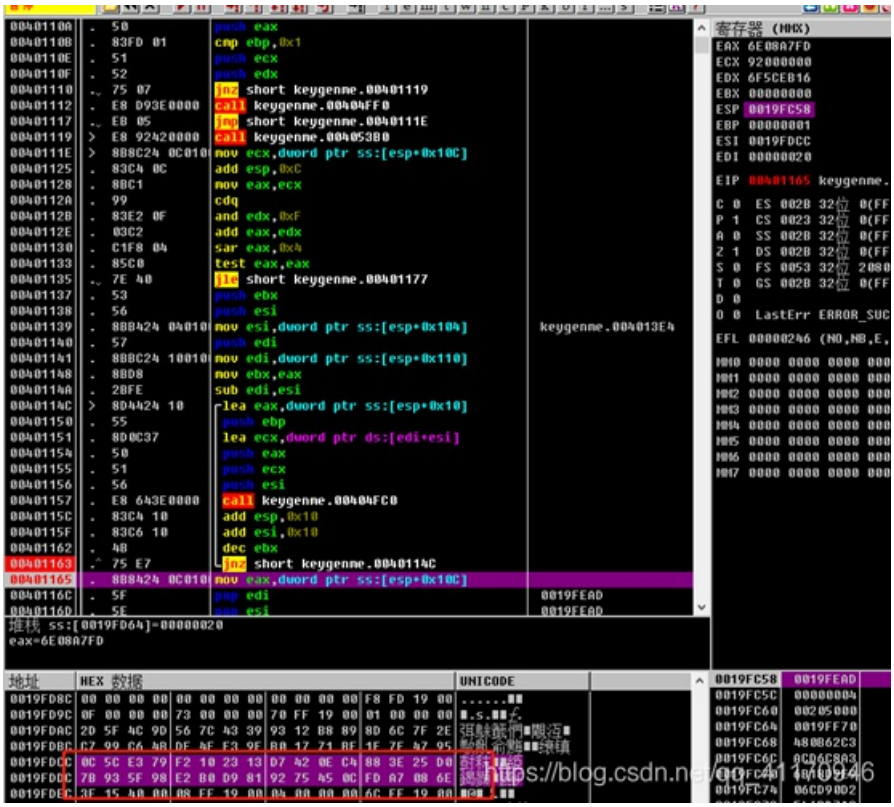


得到RSA解密后的明文为

M = 2D5F4C9D567C43399312B8898D6C7F2EC799C64BDE4FE39EB01771BE1E7F4795

把M当做sn传入OD，在aes之前设断点，把传入的0设置为1，即把加密修改成解密





跟踪到解密后的堆栈数据为

0C5CE379F2102313D7420EC4883E25D07B935F98E2B0D9819275450CFDA7086E，传到程序里发现成功了！