Week 3 WP - Veritas501

RE

Security Window

没壳，先OD上跑跑熟悉一下，然后无情丢到IDA里：
定位关键函数（带注释）：

```
BYTE *sub_401200()
{
  BYTE *result; // eax@1
  signed int i; // eax@2
  int _ch; // ecx@3
  char v3; // dl@3
  int v4; // ecx@3
  char v5; // dl@3
  int v6; // ecx@3
  signed int j; // eax@4
  char v8; // cl@5
  int v9; // eax@6
  char *v10; // ecx@6
  signed int k; // esi@6
  char v12; // dl@7
  signed int l; // eax@8
  int v14; // edx@9
  char v15; // cl@9
  int v16; // edx@9
  char v17; // cl@9
  int v18; // edx@9
  unsigned int v19; // edi@10
  void *v20; // esi@10
```

```
  result = (BYTE *)strlen(pt_input_0);          // input length = 32 , pt = 0x403420
  if ( result == (BYTE *)32 )
  {
    i = 0;
    do
    {
      _ch = (unsigned __int8)pt_input_1[i];
      pt_input_0[i] = byte_4021D0[(unsigned __int8)pt_input_0[i]];
      v3 = byte_4021D0[_ch];
      v4 = (unsigned __int8)pt_input_2[i];
      pt_input_1[i] = v3;
      v5 = byte_4021D0[v4];
      v6 = (unsigned __int8)pt_input_3[i];
      pt_input_2[i] = v5;
      pt_input_3[i] = byte_4021D0[v6];
      i += 4;                                   // 4 bytes 一组
    }
    while ( i < 32 );                           // (变换1)从byte_4021D0[255]到pt_input 做字符映射
    j = 0;
    do
    {
      v8 = __ROL1__(pt_input_0[j], 4);          // (变换2)对每一位:ch = (ch << 4) ^ 34
      pt_input_0[j++] = v8 ^ 34;
    }
    while ( j < 32 );
    v9 = 0;
    v10 = (char *)&pt_input_31;                 // v10为第32位(下标31)
    k = 16;
    do
    {
      v12 = *v10;
      *v10 = pt_input_0[v9];
      pt_input_0[v9++] = v12;
      --v10;
      --k;
    }
    while ( k );                                // (变换3)倒序重排
    l = 0;
    do
    {
      v14 = (unsigned __int8)pt_input_1[l];
      pt_input_0[l] = byte_4021D0[(unsigned __int8)pt_input_0[l]];
      v15 = byte_4021D0[v14];
      v16 = (unsigned __int8)pt_input_2[l];
```

```
      pt_input_1[l] = v15;
      v17 = byte_4021D0[v16];
      v18 = (unsigned __int8)pt_input_3[l];
      pt_input_2[l] = v17;
      pt_input_3[l] = byte_4021D0[v18];
      l += 4;
    }
    while ( l < 32 );                                   // (变换4)从byte_402
1D0[255]到pt_input 做第二次字符映射
    sub_401000();                                       // (变换5)base64加密
    v19 = 43;                                           // 加密后43位
    v20 = &flag_enc;                                    // 0x402154
    result = sub_401140();                              // (变换6)CryptEncry
pt加密
    while ( *(_DWORD *)result == *(_DWORD *)v20 )
    {
      v19 -= 4;
      v20 = (char *)v20 + 4;
      result += 4;
      if ( v19 < 4 )
      {
        if ( *(_BYTE *)v20 == *result && *((_BYTE *)v20 + 1) == res
ult[1] && *((_BYTE *)v20 + 2) == result[2] )
          result = (BYTE *)MessageBoxA(0, "You won!", "congratulati
on", 0);
        return result;
      }
    }
  }
  return result;
}
```

```
BYTE *sub_401000()
{
  int v0; // esi@1
  signed int i; // ecx@1
  unsigned int j; // eax@1
  unsigned __int8 v3; // dl@3
  unsigned __int8 v4; // bl@3
  unsigned __int8 v5; // dl@4
  char v6; // si@5

  v0 = 1 - (_DWORD)pt_input_0;
  i = 0;
  j = 0;
  while ( 1 )                                   // 猜测 base64
  {
    v3 = 16 * (pt_input_0[i] & 3);              // v3 = 16 * (inpu
t[i] & 3)
    *(&des_0 + j) = asc_402188[(unsigned int)(unsigned __int8)pt_in
put_0[i] >> 2];// des[j+2] = asc[input[i] >> 2]
    v4 = v3;                                    // v4 = v3
    if ( (signed int)(&pt_input_0[v0] + i) >= 32 )// if(i+1 >= 32)
    {
      des_1[j] = asc_402188[v3];                // des[1+j] = asc[v
3]
      des_2_3[j / 2] = '==';                    // des[j/2+2] = de
s[j/2+3] = 0x3D
      des_4[j] = 0;                             // des[j+4] = 0
      return &des_0;                            // << break && retu
rn
    }
    v5 = 4 * (pt_input_1[i] & 0xF);             // v5 = 4 * (inpu
t[1+i] & 0xF)
    des_1[j] = asc_402188[v4 | ((unsigned int)(unsigned __int8)pt_i
nput_1[i] >> 4)];// des[j+1] = asc[v4 | input[1+i]>>4]
    if ( (signed int)(&pt_input_0[2 - (signed int)pt_input_0] + i)
>= 32 )// if(i + 2 >= 32)
      break;
    v6 = pt_input_2[i];                         // v6 = input[2+i]
    LOBYTE(des_2_3[j / 2]) = asc_402188[v5 | ((unsigned int)(unsign
ed __int8)pt_input_2[i] >> 6)];// des[j/2 + 2] = asc[v5 | inpu
t[2+i] >> 6]
    *((_BYTE *)&des_3 + j) = asc_402188[v6 & 0x3F];// des[j+3] = as
c[v6 & 0x3F]
    i += 3;                                     // i += 3
    j += 4;                                     // j += 4
    if ( i >= 32 )                              // if(i >= 32)
    {
      *(&des_0 + j) = 0;                        // des[j] = 0
```

```
      return &des_0;
    }
    v0 = 1 - (_DWORD)pt_input_0;
  }
  LOBYTE(des_2_3[j / 2]) = asc_402188[v5];      // des[j/2 + 2] = a
sc[v5]
  *(_WORD *)((char *)&des_3 + j) = '=';         // des[j+3] = 0x3D
  return &des_0;
}
```

```
BYTE *sub_401140()
{
  DWORD pdwDataLen; // [sp+0h] [bp-10h]@1
  HCRYPTKEY phKey; // [sp+4h] [bp-Ch]@1
  HCRYPTPROV phProv; // [sp+8h] [bp-8h]@1
  HCRYPTHASH phHash; // [sp+Ch] [bp-4h]@1

  CryptAcquireContextA(&phProv, 0, "Microsoft Base Cryptographic Pr
ovider v1.0", 1u, 0xF0000000);
  CryptCreateHash(phProv, 0x8003u, 0, 0, &phHash);
  CryptHashData(phHash, pbData, strlen((const char *)pbData), 0);//
pbData = 'vidar_aaa'
  CryptDeriveKey(phProv, 0x6801u, phHash, 0, &phKey);
  pdwDataLen = 44;
  CryptEncrypt(phKey, 0, 1, 0, &des_0, &pdwDataLen, 0x2Cu);
  CryptDestroyKey(phKey);
  CryptDestroyHash(phHash);
  CryptReleaseContext(phProv, 0);
  return &des_0;
}
```

dump出的flag_enc：

```
0xAF,0xA5,0x92,0x3C,0x0C,0xB1,0x1C,0x33,0x56,0x66,0x3F,
0x37,0x17,0x3E,0x2A,0xE0,0xFF,0xE9,0x97,0x29,0xEC,0x76,
0x85,0xF8,0xA7,0x5F,0x85,0xCB,0x7B,0x42,0xC9,0x04,0xCB,
0x9D,0x12,0x58,0x2D,0x25,0xA4,0xB0,0xC7,0x0F,0xB9,0xE0
```

dump出的byte_4021D0[255]：

```
0x07,0x0E,0x15,0x1C,0x23,0x2A,0x31,0x38,0x3F,0x46,0x4D,0x54,0x5B,0x
62,0x69,0x70,0x77,0x7E,0x85,0x8C,0x93,0x9A,0xA1,0xA8,0xAF,0xB6,0xB
D,0xC4,0xCB,0xD2,0xD9,0xE0,0xE7,0xEE,0xF5,0xFC,0x03,0x0A,0x11,0x1
8,0x1F,0x26,0x2D,0x34,0x3B,0x42,0x49,0x50,0x57,0x5E,0x65,0x6C,0x7
3,0x7A,0x81,0x88,0x8F,0x96,0x9D,0xA4,0xAB,0xB2,0xB9,0xC0,0xC7,0xC
E,0xD5,0xDC,0xE3,0xEA,0xF1,0xF8,0xFF,0x06,0x0D,0x14,0x1B,0x22,0x2
9,0x30,0x37,0x3E,0x45,0x4C,0x53,0x5A,0x61,0x68,0x6F,0x76,0x7D,0x8
4,0x8B,0x92,0x99,0xA0,0xA7,0xAE,0xB5,0xBC,0xC3,0xCA,0xD1,0xD8,0xD
F,0xE6,0xED,0xF4,0xFB,0x02,0x09,0x10,0x17,0x1E,0x25,0x2C,0x33,0x3
A,0x41,0x48,0x4F,0x56,0x5D,0x64,0x6B,0x72,0x79,0x80,0x87,0x8E,0x9
5,0x9C,0xA3,0xAA,0xB1,0xB8,0xBF,0xC6,0xCD,0xD4,0xDB,0xE2,0xE9,0xF
0,0xF7,0xFE,0x05,0x0C,0x13,0x1A,0x21,0x28,0x2F,0x36,0x3D,0x44,0x4
B,0x52,0x59,0x60,0x67,0x6E,0x75,0x7C,0x83,0x8A,0x91,0x98,0x9F,0xA
6,0xAD,0xB4,0xBB,0xC2,0xC9,0xD0,0xD7,0xDE,0xE5,0xEC,0xF3,0xFA,0x0
1,0x08,0x0F,0x16,0x1D,0x24,0x2B,0x32,0x39,0x40,0x47,0x4E,0x55,0x5
C,0x63,0x6A,0x71,0x78,0x7F,0x86,0x8D,0x94,0x9B,0xA2,0xA9,0xB0,0xB
7,0xBE,0xC5,0xCC,0xD3,0xDA,0xE1,0xE8,0xEF,0xF6,0xFD,0x04,0x0B,0x1
2,0x19,0x20,0x27,0x2E,0x35,0x3C,0x43,0x4A,0x51,0x58,0x5F,0x66,0x6
D,0x74,0x7B,0x82,0x89,0x90,0x97,0x9E,0xA5,0xAC,0xB3,0xBA,0xC1,0xC
8,0xCF,0xD6,0xDD,0xE4,0xEB,0xF2,0xF9
```

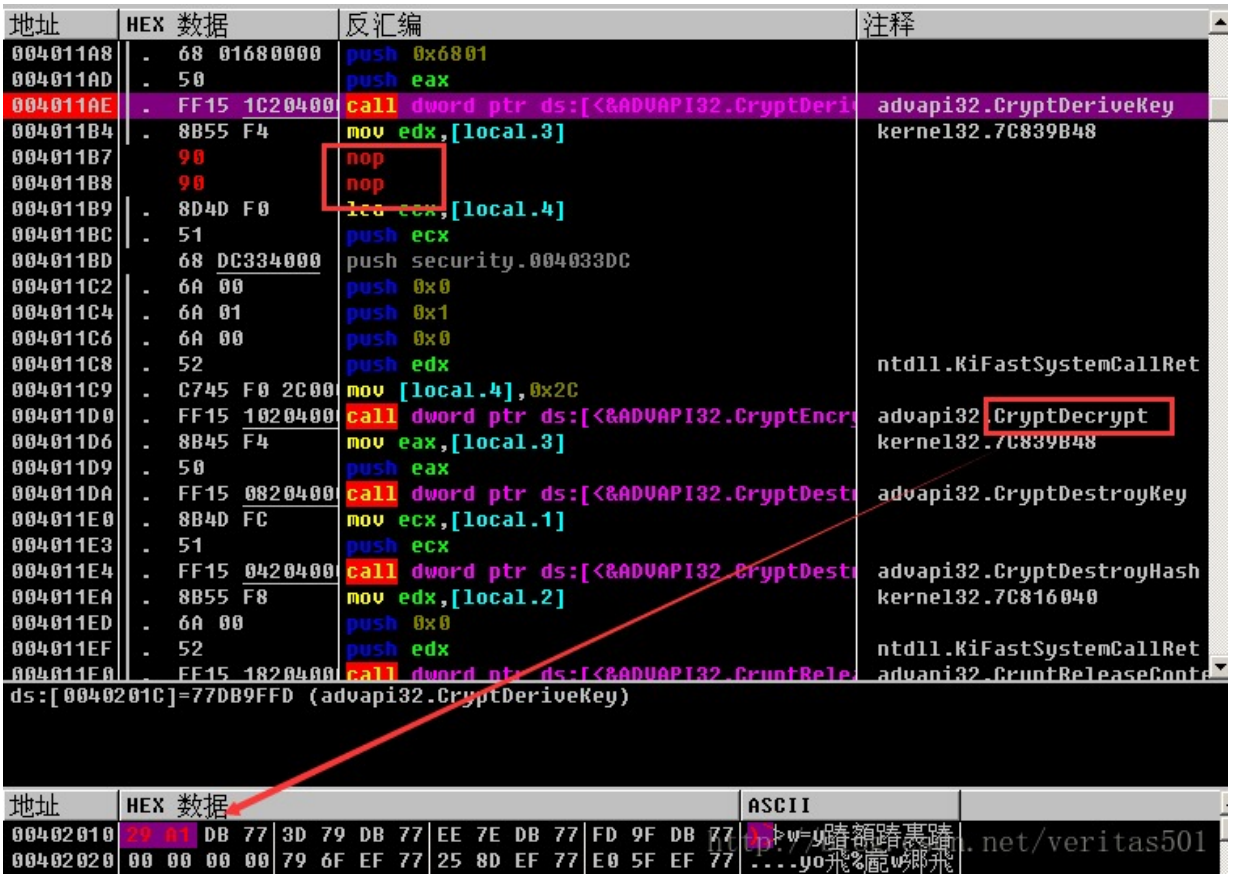
dump出的asc_402188字符
串："ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"

解密就是加密的反函数，加密的最后一步是调用CryptEncrypt，所以我们调用CryptDecrypt将flag_enc解密第一层，由于我此时并不会使用上面的一系列函数，所以我可以通过OD修改原程序，使他调用CryptDecrypt来解密flag_enc。
查询
https://msdn.microsoft.com/en-us/library/windows/desktop/aa379913(v=vs.85).aspx
和
https://msdn.microsoft.com/en-us/library/windows/desktop/aa379924(v=vs.85).aspx
得：

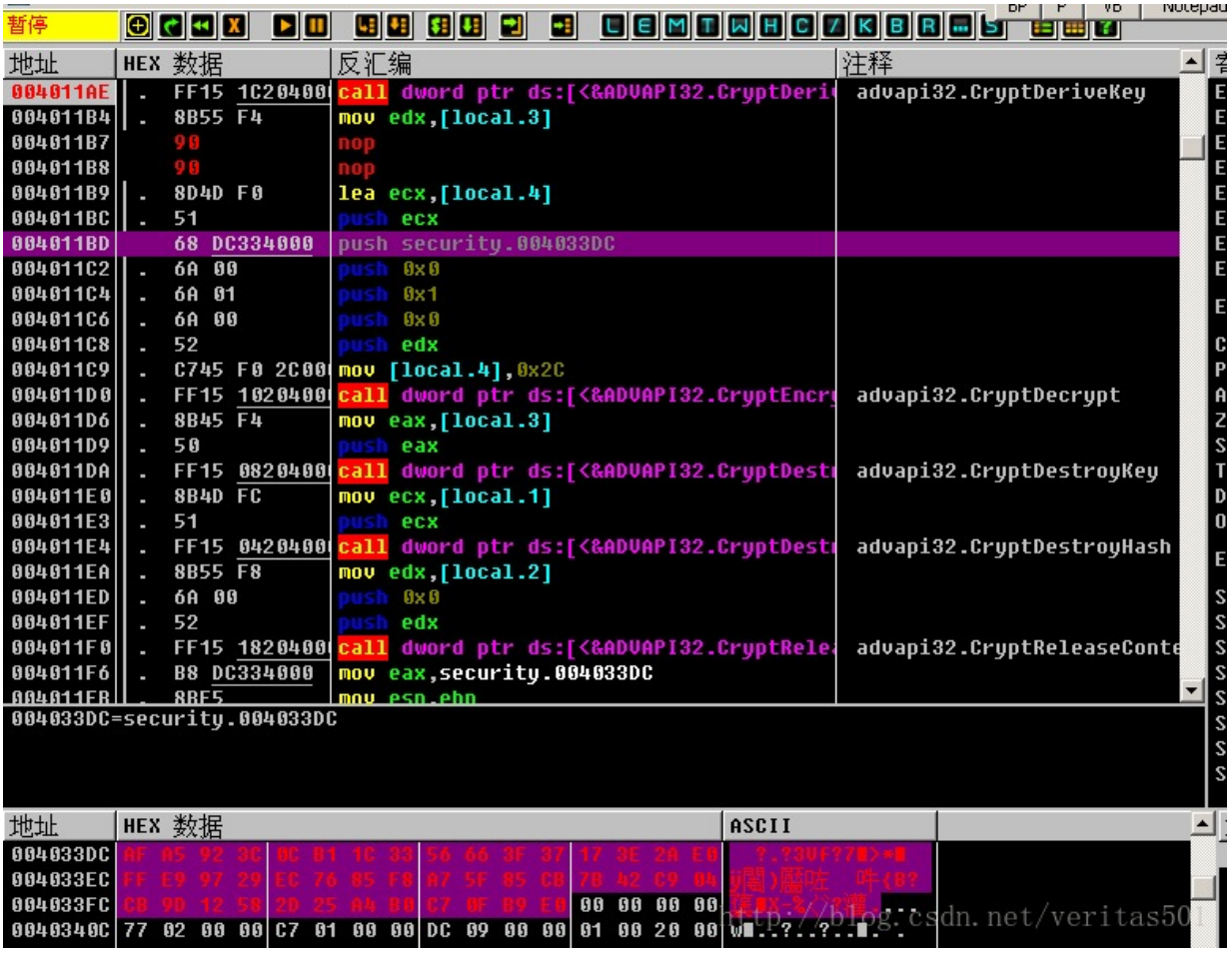
```
BOOL WINAPI CryptDecrypt(
  _In_    HCRYPTKEY  hKey,
  _In_    HCRYPTHASH hHash,
  _In_    BOOL       Final,
  _In_    DWORD      dwFlags,
  _Inout_ BYTE       *pbData,
  _Inout_ DWORD      *pdwDataLen
);
```

```
BOOL WINAPI CryptEncrypt(
  _In_    HCRYPTKEY  hKey,
  _In_    HCRYPTHASH hHash,
  _In_    BOOL       Final,
  _In_    DWORD      dwFlags,
  _Inout_ BYTE       *pbData,
  _Inout_ DWORD      *pdwDataLen,
  _In_    DWORD      dwBufLen
);
```

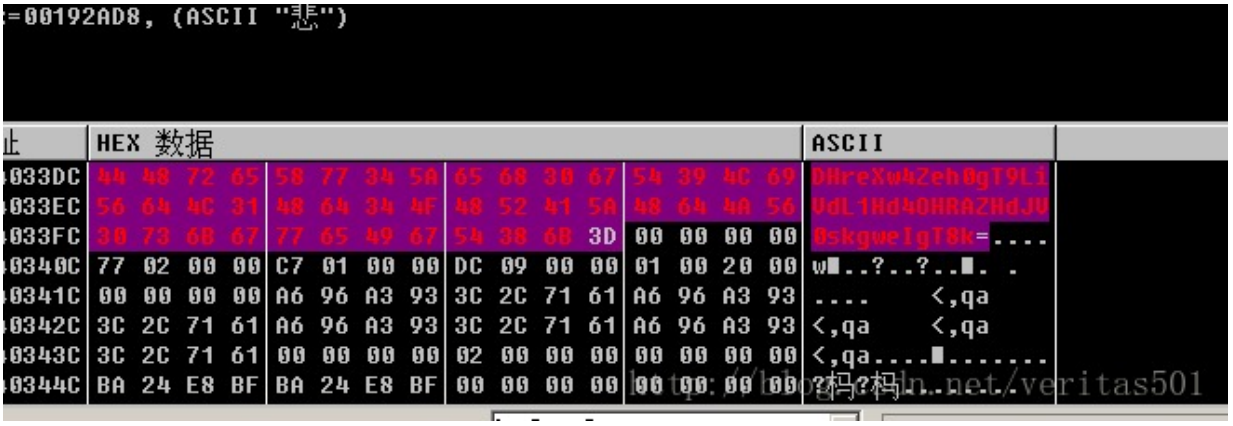
我们这样修改程序：



并在合适时机把flag_enc写入0x004033DC：



接着跑完这段代码，flag_enc成功解密(第6层)：



flag_enc_5:DHreXw4Zeh0gT9LiVdL1Hd4OHRAZHdJV0skgweIgT8k=

由于之前做过base64相关的re，看的还是比较准的，直接base64解密（第5层）：

flag_enc_4：

```
0x0c,0x7a,0xde,0x5f,0x0e,0x19,0x7a,0x1d,
0x20,0x4f,0xd2,0xe2,0x55,0xd2,0xf5,0x1d,
0xde,0x0e,0x1d,0x10,0x19,0x1d,0xd2,0x55,
0xd2,0xc9,0x20,0xc1,0xe2,0x20,0x4f,0xc9
```

第4层直接用py搞定（第4层）：

```
a=[0x07,0x0E,0x15,0x1C,0x23,0x2A,0x31,0x38,0x3F,0x46,0x4D,0x54,0x5
B,0x62,0x69,0x70,0x77,0x7E,0x85,0x8C,0x93,0x9A,0xA1,0xA8,0xAF,0xB
6,0xBD,0xC4,0xCB,0xD2,0xD9,0xE0,0xE7,0xEE,0xF5,0xFC,0x03,0x0A,0x1
1,0x18,0x1F,0x26,0x2D,0x34,0x3B,0x42,0x49,0x50,0x57,0x5E,0x65,0x6
C,0x73,0x7A,0x81,0x88,0x8F,0x96,0x9D,0xA4,0xAB,0xB2,0xB9,0xC0,0xC
7,0xCE,0xD5,0xDC,0xE3,0xEA,0xF1,0xF8,0xFF,0x06,0x0D,0x14,0x1B,0x2
2,0x29,0x30,0x37,0x3E,0x45,0x4C,0x53,0x5A,0x61,0x68,0x6F,0x76,0x7
D,0x84,0x8B,0x92,0x99,0xA0,0xA7,0xAE,0xB5,0xBC,0xC3,0xCA,0xD1,0xD
8,0xDF,0xE6,0xED,0xF4,0xFB,0x02,0x09,0x10,0x17,0x1E,0x25,0x2C,0x3
3,0x3A,0x41,0x48,0x4F,0x56,0x5D,0x64,0x6B,0x72,0x79,0x80,0x87,0x8
E,0x95,0x9C,0xA3,0xAA,0xB1,0xB8,0xBF,0xC6,0xCD,0xD4,0xDB,0xE2,0xE
9,0xF0,0xF7,0xFE,0x05,0x0C,0x13,0x1A,0x21,0x28,0x2F,0x36,0x3D,0x4
4,0x4B,0x52,0x59,0x60,0x67,0x6E,0x75,0x7C,0x83,0x8A,0x91,0x98,0x9
F,0xA6,0xAD,0xB4,0xBB,0xC2,0xC9,0xD0,0xD7,0xDE,0xE5,0xEC,0xF3,0xF
A,0x01,0x08,0x0F,0x16,0x1D,0x24,0x2B,0x32,0x39,0x40,0x47,0x4E,0x5
5,0x5C,0x63,0x6A,0x71,0x78,0x7F,0x86,0x8D,0x94,0x9B,0xA2,0xA9,0xB
0,0xB7,0xBE,0xC5,0xCC,0xD3,0xDA,0xE1,0xE8,0xEF,0xF6,0xFD,0x04,0x0
B,0x12,0x19,0x20,0x27,0x2E,0x35,0x3C,0x43,0x4A,0x51,0x58,0x5F,0x6
6,0x6D,0x74,0x7B,0x82,0x89,0x90,0x97,0x9E,0xA5,0xAC,0xB3,0xBA,0xC
1,0xC8,0xCF,0xD6,0xDD,0xE4,0xEB,0xF2,0xF9]
b = [0x0c,0x7a,0xde,0x5f,0x0e,0x19,0x7a,0x1d,0x20,0x4f,0xd2,0xe2,0x
55,0xd2,0xf5,0x1d,0xde,0x0e,0x1d,0x10,0x19,0x1d,0xd2,0x55,0xd2,0xc
9,0x20,0xc1,0xe2,0x20,0x4f,0xc9]

for chb in b:
    for i in range(len(a)):
        if a[i] == chb :
            print(i+1,end = ',')
```

得到flag_enc_3:

```
148,54,178,233,2,223,54,187,224,121,30,142,195,30,35,187,178,2,18
7,112,223,187,30,195,30,175,224,247,142,224,121,175
```

数组倒序（第3层），继续py：

```
a=[148,54,178,233,2,223,54,187,224,121,30,142,195,30,35,187,178,2,1
87,112,223,187,30,195,30,175,224,247,142,224,121,175]

for i in reversed(range(len(a))):
    print(a[i],end=',')
```

得到flag_enc_2：

```
175,121,224,142,247,224,175,30,195,30,187,223,112,187,2,178,187,3
5,30,195,142,30,121,224,187,54,223,2,233,178,54,148
```

第2层和第1层由于计算量不大，我直接一位一位用的爆破，py:

```
a=[0x07,0x0E,0x15,0x1C,0x23,0x2A,0x31,0x38,0x3F,0x46,0x4D,0x54,0x5
B,0x62,0x69,0x70,0x77,0x7E,0x85,0x8C,0x93,0x9A,0xA1,0xA8,0xAF,0xB
6,0xBD,0xC4,0xCB,0xD2,0xD9,0xE0,0xE7,0xEE,0xF5,0xFC,0x03,0x0A,0x1
1,0x18,0x1F,0x26,0x2D,0x34,0x3B,0x42,0x49,0x50,0x57,0x5E,0x65,0x6
C,0x73,0x7A,0x81,0x88,0x8F,0x96,0x9D,0xA4,0xAB,0xB2,0xB9,0xC0,0xC
7,0xCE,0xD5,0xDC,0xE3,0xEA,0xF1,0xF8,0xFF,0x06,0x0D,0x14,0x1B,0x2
2,0x29,0x30,0x37,0x3E,0x45,0x4C,0x53,0x5A,0x61,0x68,0x6F,0x76,0x7
D,0x84,0x8B,0x92,0x99,0xA0,0xA7,0xAE,0xB5,0xBC,0xC3,0xCA,0xD1,0xD
8,0xDF,0xE6,0xED,0xF4,0xFB,0x02,0x09,0x10,0x17,0x1E,0x25,0x2C,0x3
3,0x3A,0x41,0x48,0x4F,0x56,0x5D,0x64,0x6B,0x72,0x79,0x80,0x87,0x8
E,0x95,0x9C,0xA3,0xAA,0xB1,0xB8,0xBF,0xC6,0xCD,0xD4,0xDB,0xE2,0xE
9,0xF0,0xF7,0xFE,0x05,0x0C,0x13,0x1A,0x21,0x28,0x2F,0x36,0x3D,0x4
4,0x4B,0x52,0x59,0x60,0x67,0x6E,0x75,0x7C,0x83,0x8A,0x91,0x98,0x9
F,0xA6,0xAD,0xB4,0xBB,0xC2,0xC9,0xD0,0xD7,0xDE,0xE5,0xEC,0xF3,0xF
A,0x01,0x08,0x0F,0x16,0x1D,0x24,0x2B,0x32,0x39,0x40,0x47,0x4E,0x5
5,0x5C,0x63,0x6A,0x71,0x78,0x7F,0x86,0x8D,0x94,0x9B,0xA2,0xA9,0xB
0,0xB7,0xBE,0xC5,0xCC,0xD3,0xDA,0xE1,0xE8,0xEF,0xF6,0xFD,0x04,0x0
B,0x12,0x19,0x20,0x27,0x2E,0x35,0x3C,0x43,0x4A,0x51,0x58,0x5F,0x6
6,0x6D,0x74,0x7B,0x82,0x89,0x90,0x97,0x9E,0xA5,0xAC,0xB3,0xBA,0xC
1,0xC8,0xCF,0xD6,0xDD,0xE4,0xEB,0xF2,0xF9]


b=[175,121,224,142,247,224,175,30,195,30,187,223,112,187,2,178,18
7,35,30,195,142,30,121,224,187,54,223,2,233,178,54,148]
for j in range(len(b)):
    for i in range(30,128):
        if ((a[i]<<4)%255)^34 == b[j]:
            print(chr(i+1),end='')
```

得到flag_dec：`hctf{there_is_no_perfect_window}`

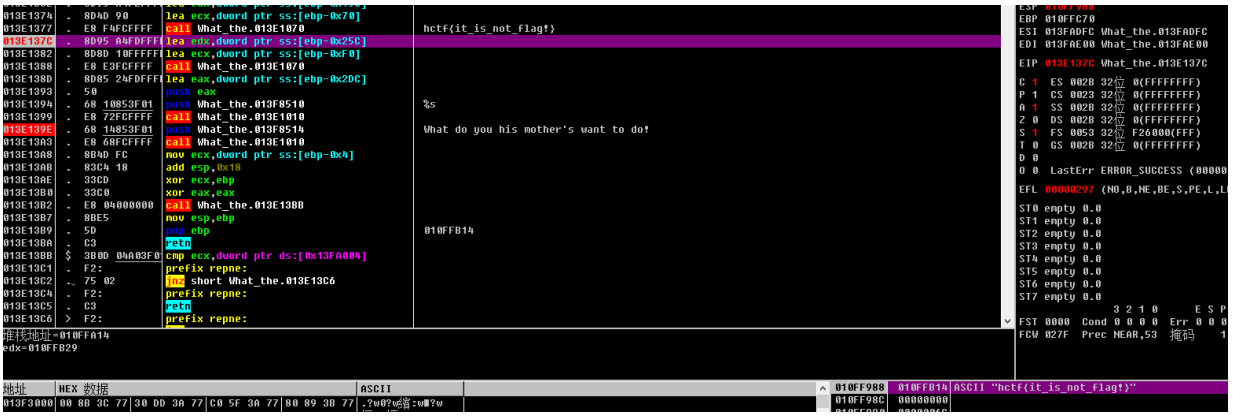
思维混乱的出题人

说句实话，我做完后感觉这题的确有些混乱，我尽量简单的说。

先随便跑跑，发现随便输入最后会显示`What do you his mother's want to do!`，OD载入，查找字符串，发现3个比较可疑的字符串`tutushigecaiji`，`aGN0ZntpdF9pc19ub3RfZmxhZyF9`和`What do you his mother's want to do!`，第一个字符串让我感觉到了出题者和土土~~某种微妙的关系~~。动态调试后可知0x013E2B70是一个gets之类获取用户输入的函数，不是重点，略过。

```
013E130F  .  8D8D 24FEFFF lea ecx,dword ptr ss:[ebp-0x1DC]
013E1315  .  E8 66FEFFFF  call What_the.013E1180
013E131A  .  8D85 10FFFFF lea eax,dword ptr ss:[ebp-0xF0]
013E1320  .  50           push eax
013E1321  .  E8 4A180000  call What_the.013E2B70                gets
013E1326  .  A1 D0843F01  mov eax,dword ptr ds:[0x13F84D0]      ZyF9
013E132B  .  0F1005 B8843 movups xmm0,dqword ptr ds:[0x13F84B8] aGN0ZntpdF9pc19ub3RfZmxhZyF9
013E1332  .  8945 A8      mov dword ptr ss:[ebp-0x58],eax
013E1335  .  A0 D4843F01  mov al,byte ptr ds:[0x13F84D4]
013E133A  .  6A 4F        push 0x4F
```

比较微妙的是，当我们f8小心单步后可以发现：

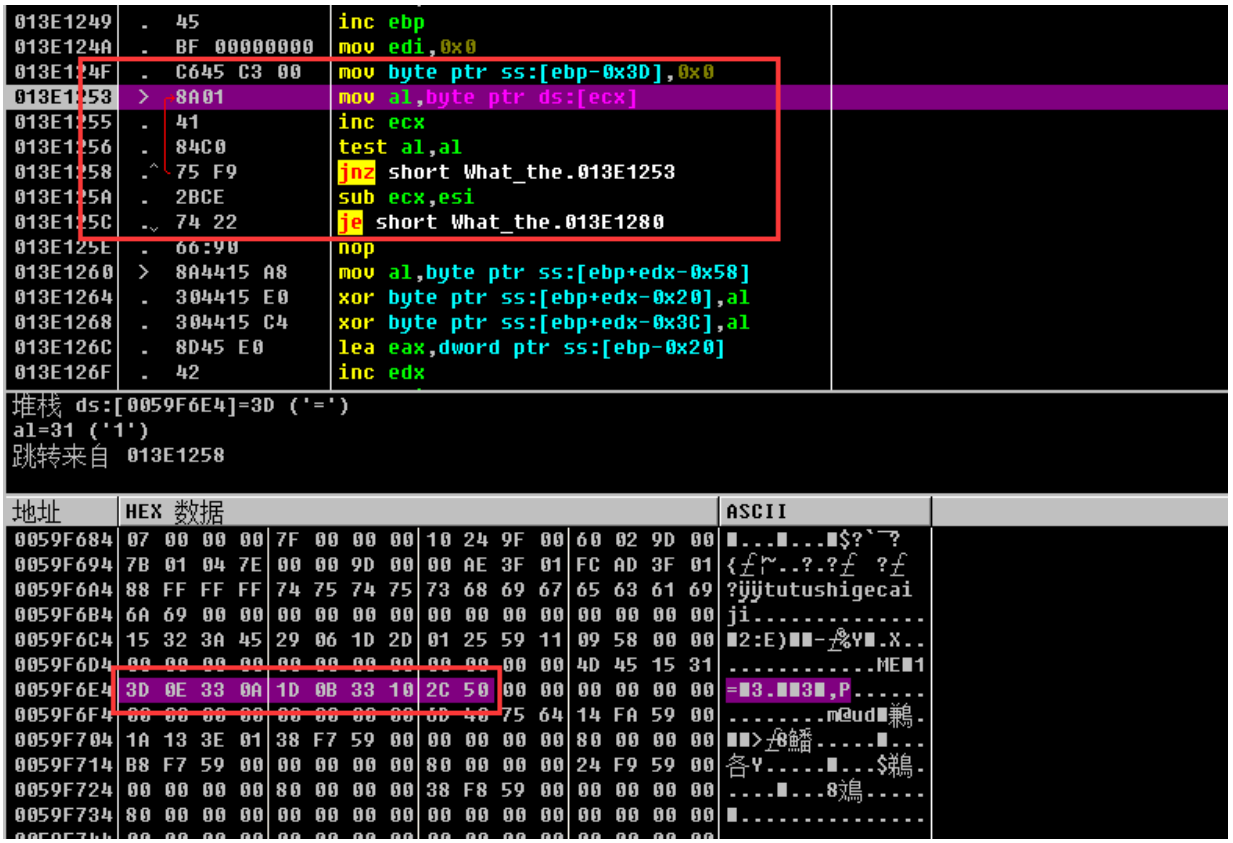


当我们执行完CALL 0x13E1070后，字符串`hctf{it_is_not_flag!}`出现在了我们的堆栈区，所以我用IDA看了一下这个CALL，发现0x61（‘=’）这个字节：

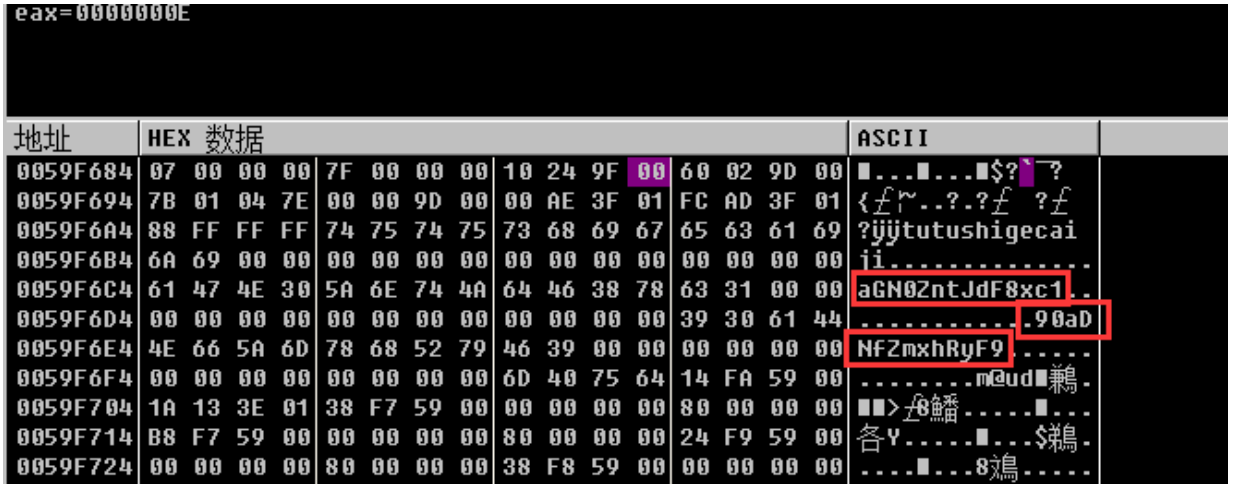
```
  if ( v2 < result )
  {
    v7 = v2 + 2;
    v8 = ((result - v2 - 1) >> 2) + 1;
    do
    {
      v9 = *(_BYTE *)(v7 - 1);
      v7 += 4;
      *(_BYTE *)a2 = 4 * dword_418540[4 * *(_BYTE *)(v7 - 6)] | (*(_DWORD *)&dv
      *(_BYTE *)(a2 + 1) = 16 * dword_418540[4 * *(_BYTE *)(v7 - 5)] | (*(_DWOR
      *(_BYTE *)(a2 + 2) = dword_418540[4 * *(_BYTE *)(v7 - 3)] | (dword_41854(
      a2 += 3;
      --v8;
    }
    while ( v8 );
    result = v10;
  }
  if ( *(_BYTE *)(result - 2) == '=' )
  {
    *(_BYTE *)(a2 - 2) = 0;
  }
  else if ( *(_BYTE *)(result - 1) == '=' )
  {
    *(_BYTE *)(a2 - 1) = 0;
  }
```

理所当然的想到了base64，拿`aGN0ZntpdF9pc19ub3RfZmxhZyF9`进行base64解密得到了我们看到的字符串`hctf{it_is_not_flag!}`，合理推测flag应该是由某个base64字串解密得到。

回头看一下，我们现在还有一个字符串没有用到：`tutushigecaiji`，我们总不能把出题人想的这么坏，写个和题目无关的字符串只为了~~吐槽~~一下土土吧？我们到字符串的附近下断，动态跟踪一下，发现此处：



程序在循环利用写在0x0059F6E4出的字节码进行某种解密，我们就在跳转的前面轻轻摁一下f4，发现此时数据窗口出现了两个可疑字符串：



我们把它拼起来，得到`aGN0ZntJdF8xc190aDNfZmxhRyF9`,base64解密一下得到flag：`hctf{It_1s_th3_flaG!}`

Crypto

explorer的奇怪番外3

该Google的先Google，知道是Feistel结构，直接用py写出反函数解密：

```
# -*- coding: cp936 -*-
from hashlib import sha256

def xor(a,b):
    return ''.join([chr(ord(i)^ord(j)) for i,j in zip(a,b)])#简单的异或

def HASH(data):
    return sha256(data).digest()[:8]#取前8位

def bes_encrypt(subkeys, data):
    i = 0
    d1 = data[:8]
    d2 = data[8:]
    for i in subkeys:
       d1 = xor(xor(HASH(d2),i),d1)
       d1,d2 = d2,d1

    return d2 + d1

def key_schedule(key):
    subKeys = []
    subKey = key
    for i in xrange(16):
        subKey = HASH(subKey)
        subKeys.append(subKey)#len(subKeys)=16
    return subKeys

def bes(key,data):
    subKeys = key_schedule(key)
    return bes_encrypt(subKeys, data).encode('hex')

def bes_de(key,data):
    subKeys = reversed(key_schedule(key))
    return bes_decrypt(subKeys, data.decode('hex'))
```

```
def bes_decrypt(subkeys, data):
    i = 0
    d1 = data[8:]
    d2 = data[:8]
    for i in subkeys:
        d1,d2 = d2,d1
        d1 = xor(xor(HASH(d2),i),d1)
    return d1+d2

#the result  is "rEvers3_tHe_kEy!"
if __name__ == "__main__":
    print bes_de('explorer','1fde6a7b2ff15d0abad691215ca5d470')
```

加上`flag{}`得到最终的flag：`flag{rEvers3_tHe_kEy!}`

---

explorer的奇怪番外5

题目提到了CBC，自然就想到了CBC翻转字节攻击，话不多说。

首先，我们sign up，比如我提交的是bdmin，密码和原来一样，是alvndasjncakslbdvlaksdn。得到了token:

```
dbc290e4acefd383cd4bb93a0ee020e6169513e29fd7e5452f955b945cab77cf334
8a76d1ea5cfc9278c8e2ad3539498
```

观察程序源码可知，前面的16位（decode之后）是iv：

```
def signin():
    token = raw_input("give me you token:")
    iv = token[:32].decode('hex')
    cipherText = token[32:].decode('hex')
    aes = AES.new(key,AES.MODE_CBC,iv)
    plainText = aes.decrypt(cipherText)
    name,passwd = checkPad(plainText).split(':')
    if name == 'admin' and passwd == 'alvndasjncakslbdvlaksdn':
        print flag
    else:
        print "you are not admin"
```

CBC是16位一组进行加密的，因此我们编写代码在破坏iv的情况下把bdmin改成admin：

```
a='dbc290e4acefd383cd4bb93a0ee020e6169513e29fd7e5452f955b945cab77cf
3348a76d1ea5cfc9278c8e2ad3539498'
b=a.decode('hex')
b=list(b)
b[0] = chr(ord(b[0]) ^ ord('b') ^ ord('a'))
b = ''.join(b)
c=b.encode('hex')
print c
```

得到构造的token：

```
d8c290e4acefd383cd4bb93a0ee020e6169513e29fd7e5452f955b945cab77cf334
8a76d1ea5cfc9278c8e2ad3539498
```

提交得到flag：`hctf{cRypT0_ls_1nteRestlng!}`

进击的 Crypto [1]

这题怎么说呢，~~可能是题目有点坑~~，也可能是我有点傻，最后那个在网页中的flag_enc我一直复制不好，最后用了python才解决，迟了一点，分数也没拿到。不过那都是后话了。

首先我们随便输入，发现每次加密的结果都不一样，我猜测加密的key不可能无限长，所以我输入了非常长的一串'1'，试出来key的长度为128bytes。然后观察加密后网页的源码，可以发现下面有flag_enc：

```
                    <input type="submit" value="encode">
                </form>
        </div>
</body>
</html>

<br><br><br><br><br><br>
<div align="center"><h2>Cipher:
\x65\x66\x0e\x7f\x44\x60\x09\x5f\x10\x78\x7f\x5d\x1e\x10\x15\x15\x63\x50\x7f\x7d\x53\x4d\x51\x61\x5e\x54\x02\x4d\x6
x59\x00\x45\x1d\x71\x78\x1b\x02\x73\x49\x63\x48\x40\x0f\x10\x69\x71\x6d\x53\x57\x73\x12\x7a\x79\x74\x03\x4d\x15\x41

<!-- Flag:}YA+dWW$5XR{=c5Zz'VLv}kmqsXPx++MT@u    ^`djnNMQEci45{1}-
lBlD`kORo?J#Yg  cl%cXP!?/7l-c4kU9wP^sU*-->
```

加密的方法是K xor M_input = C_input ，K xor Flag = Flag_enc。
而我们现在知道了M_input，C_input，Flag_enc。

所以Flag = K xor Flag_enc = (M_input xor C_input) xor Flag_enc。

先写个python脚本获取Flag_enc(此处吐槽一下为什么Flag_enc不base64或hex显示)：

```
import urllib2
req = urllib2.Request( url = 'http://119.29.138.57:23333/crypto/enc
ode_system/index.php', data = 'message=1111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111')
response = urllib2.urlopen(req).read()
print response
print response[-131:-3].encode('hex')
```

从而得到了C_input和Flag_enc，python解密之:

```
c=[0x59,0x4f,0x72,0x5d,0x5c,0x07,0x72,0x6a,0x7e,0x50,0x07,0x67,0x4
9,0x44,0x15,0x6a,0x6f,0x7b,0x73,0x5e,0x77,0x18,0x50,0x53,0x59,0x0
8,0x6c,0x4f,0x40,0x4b,0x74,0x10,0x5d,0x5b,0x59,0x6f,0x15,0x05,0x0
1,0x62,0x1d,0x6d,0x7d,0x40,0x4c,0x62,0x7d,0x19,0x10,0x04,0x09,0x1
b,0x68,0x5f,0x79,0x4d,0x19,0x7e,0x1c,0x4a,0x06,0x09,0x6e,0x55,0x5
1,0x08,0x12,0x1e,0x70,0x72,0x75,0x6d,0x72,0x70,0x40,0x6c,0x7d,0x0
4,0x42,0x5d,0x48,0x7f,0x07,0x1e,0x76,0x01,0x42,0x61,0x7a,0x67,0x4
3,0x00,0x74,0x7d,0x79,0x45,0x41,0x0d,0x55,0x58,0x7a,0x79,0x00,0x4
a,0x07,0x52,0x76,0x45,0x04,0x15,0x45,0x44,0x15,0x4d,0x42,0x4b,0x4
2,0x42,0x5f,0x5d,0x69,0x63,0x5f,0x12,0x74,0x64,0x61,0x7b]

c_flag=list('3d11342d4c7230012d036e0c0523071a6a7e1e0b2f4d233d016624
2f20286f1e475339030804597949656555290039091075f77154622146432744d6
6430e2e56715f5e043565682d004e6f2a760e3b4a1600476e4b18200f035d7a693d
6417286f2d560720520f5018151d434141017b3f420a1b161c336c0d5f501a302a3
7'.decode('hex'))

b=[]
for i in range(128):
    c[i]=c[i]^ord('1')
    b.append(chr(c[i] ^ ord(c_flag[i])))
b=''.join(b)
print b
#result: UowA!DsZbbXZ}V#A44\didB_i_yQQR*?+9Q],0i*e9)$TSu!1Ng]L(jhL}Y
6Q{QJ6H|qEv!4nA?2fC}W3X6h){kpDU/K,q,cXSI?Lhctf{Rive5t_C1pher_4_1s_e
z}
```

得到flag: `hctf{Rive5t_C1pher_4_1s_ez}`