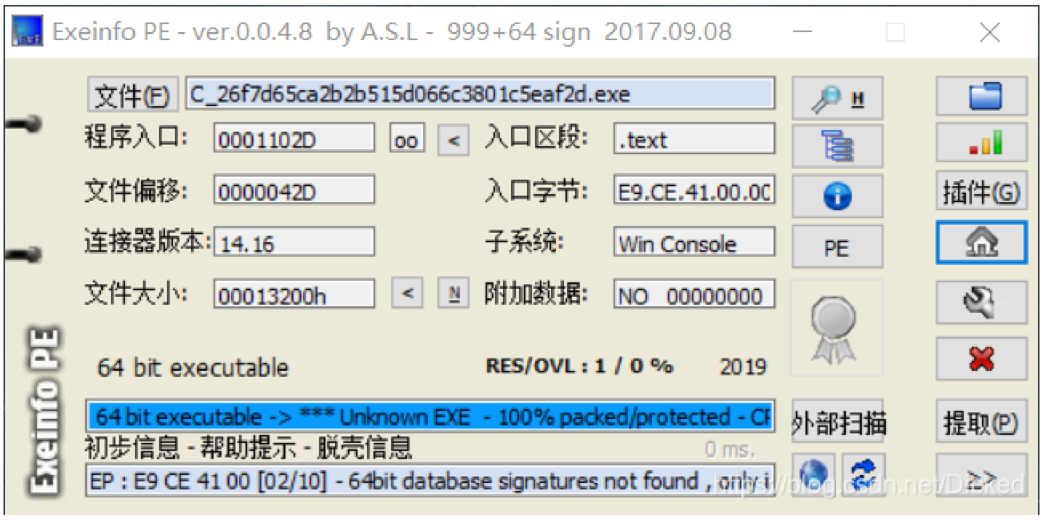
buuctf——[V&N2020 公开赛]strangeCpp && buuctf——[BJDCTF2020]easy && buuctf——[ACTF新生赛2020]usualCrypt

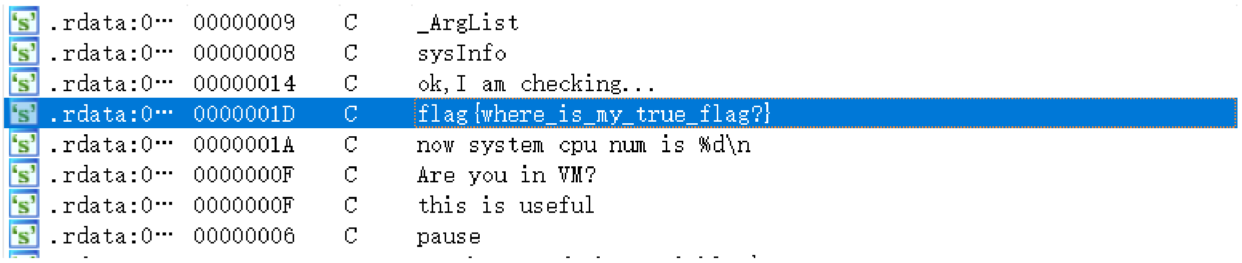
原创

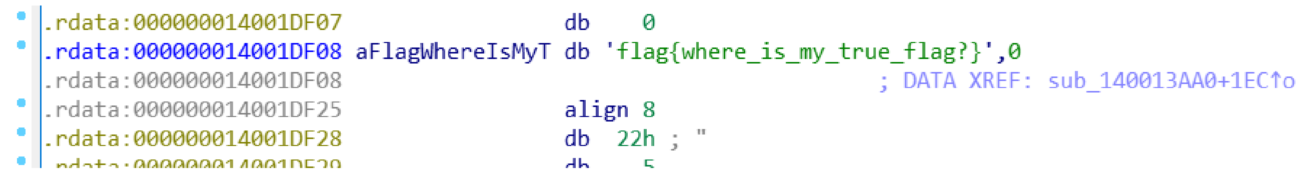
Dicked 于 2020-12-06 18:32:14 发布 84 收藏

[V&N2020 公开赛]strangeCpp

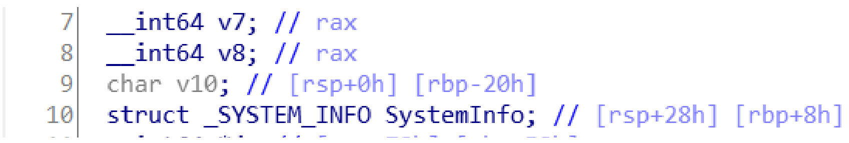

Exeinfo PE - ver.0.0.4.8 by A.S.L - 999+64 sign 2017.09.08

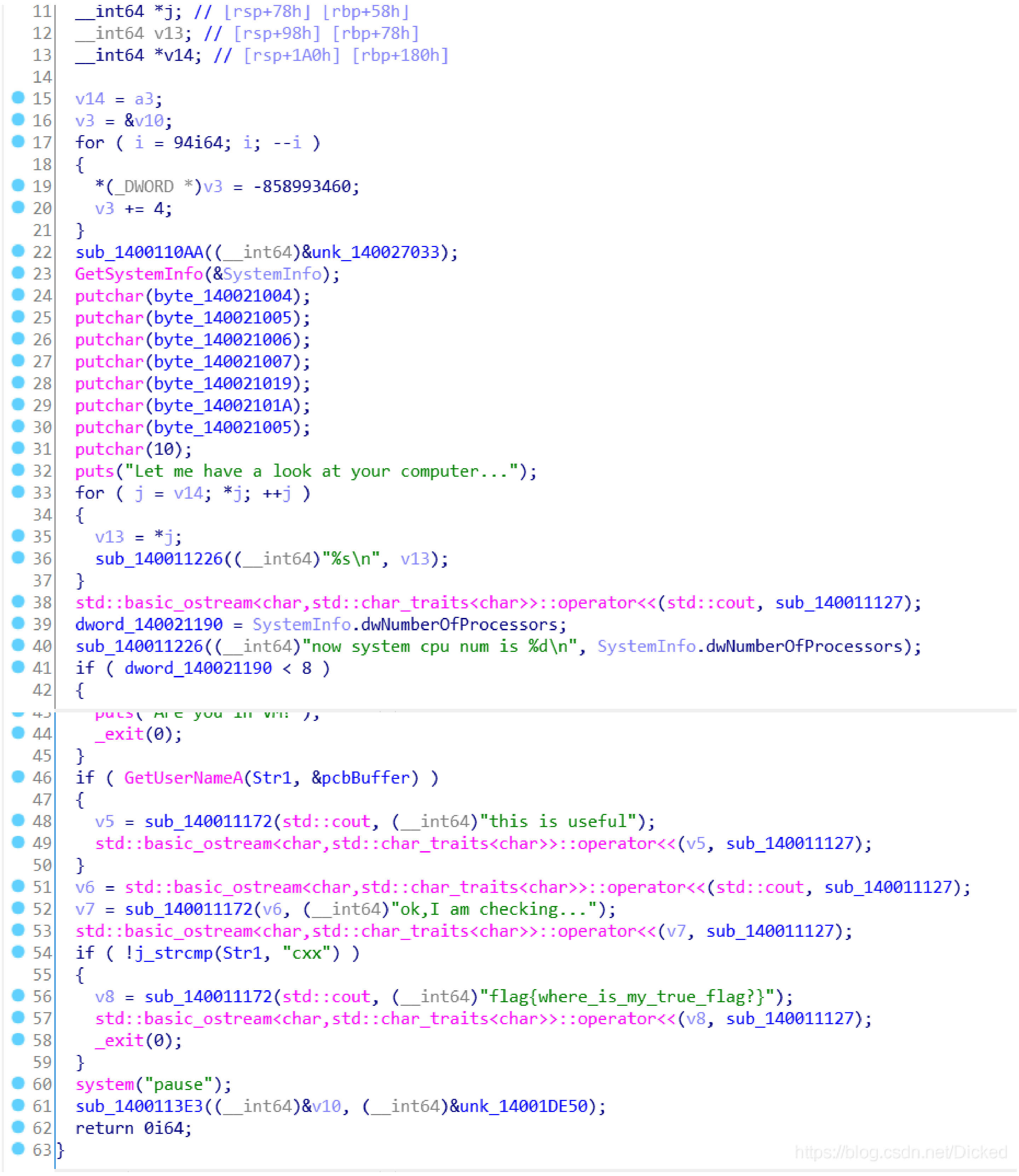
文件(F) C_26f7d65ca2b2b515d066c3801c5eaf2d.exe
程序入口: 0001102D 入口区段: .text
文件偏移: 0000042D 入口字节: E9.CE.41.00.00
连接器版本: 14.16 子系统: Win Console
文件大小: 00013200h 附加数据: NO 00000000
64 bit executable RES/OVL : 1 / 0 % 2019
64 bit executable -> *** Unknown EXE - 100% packed/protected - C
初步信息 - 帮助提示 - 脱壳信息 0 ms.
EP : E9 CE 41 00 [02/10] - 64bit database signatures not found , only i


```
.rdata:0…  00000009  C  _ArgList
.rdata:0…  00000008  C  sysInfo
.rdata:0…  00000014  C  ok,I am checking...
.rdata:0…  0000001D  C  flag{where_is_my_true_flag?}
.rdata:0…  0000001A  C  now system cpu num is %d\n
.rdata:0…  0000000F  C  Are you in VM?
.rdata:0…  0000000F  C  this is useful
.rdata:0…  00000006  C  pause
```


```
.rdata:000000014001DF07                 db    0
.rdata:000000014001DF08 aFlagWhereIsMyT db 'flag{where_is_my_true_flag?}',0
.rdata:000000014001DF08                                         ; DATA XREF: sub_140013AA0+1EC↑o
.rdata:000000014001DF25                 align 8
.rdata:000000014001DF28                 db  22h ; "
```

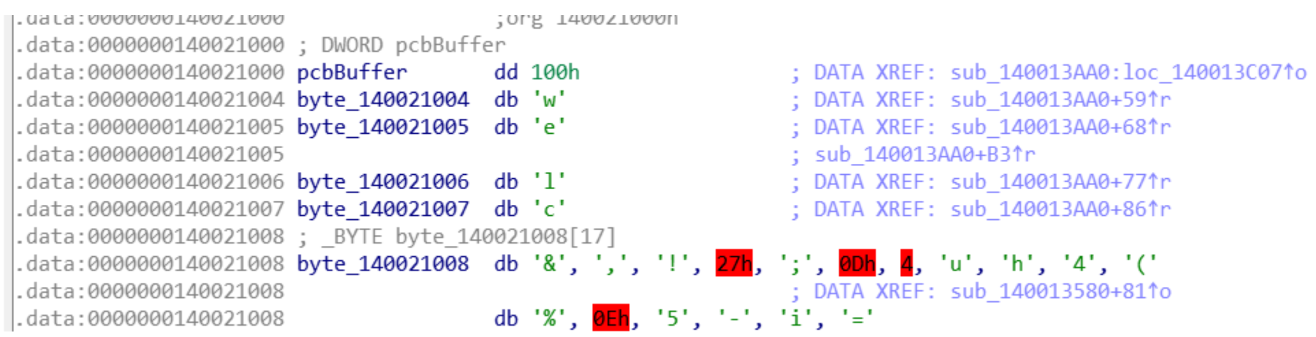
交叉引用，看伪c，应该是mian函数


```
  __int64 v7; // rax
  __int64 v8; // rax
  char v10; // [rsp+0h] [rbp-20h]
  struct _SYSTEM_INFO SystemInfo; // [rsp+28h] [rbp+8h]
```

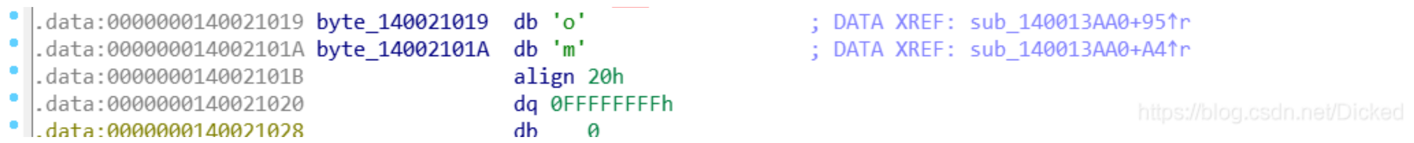


```
11  __int64 *j; // [rsp+78h] [rbp+58h]
12  __int64 v13; // [rsp+98h] [rbp+78h]
13  __int64 *v14; // [rsp+1A0h] [rbp+180h]
14
15  v14 = a3;
16  v3 = &v10;
17  for ( i = 94i64; i; --i )
18  {
19    *(_DWORD *)v3 = -858993460;
20    v3 += 4;
21  }
22  sub_1400110AA((__int64)&unk_140027033);
23  GetSystemInfo(&SystemInfo);
24  putchar(byte_140021004);
25  putchar(byte_140021005);
26  putchar(byte_140021006);
27  putchar(byte_140021007);
28  putchar(byte_140021019);
29  putchar(byte_14002101A);
30  putchar(byte_140021005);
31  putchar(10);
32  puts("Let me have a look at your computer...");
33  for ( j = v14; *j; ++j )
34  {
35    v13 = *j;
36    sub_140011226((__int64)"%s\n", v13);
37  }
38  std::basic_ostream<char,std::char_traits<char>>::operator<<(std::cout, sub_140011127);
39  dword_140021190 = SystemInfo.dwNumberOfProcessors;
40  sub_140011226((__int64)"now system cpu num is %d\n", SystemInfo.dwNumberOfProcessors);
41  if ( dword_140021190 < 8 )
42  {
43    puts("Are you in vm?");
44    _exit(0);
45  }
46  if ( GetUserNameA(Str1, &pcbBuffer) )
47  {
48    v5 = sub_140011172(std::cout, (__int64)"this is useful");
49    std::basic_ostream<char,std::char_traits<char>>::operator<<(v5, sub_140011127);
50  }
51  v6 = std::basic_ostream<char,std::char_traits<char>>::operator<<(std::cout, sub_140011127);
52  v7 = sub_140011172(v6, (__int64)"ok,I am checking...");
53  std::basic_ostream<char,std::char_traits<char>>::operator<<(v7, sub_140011127);
54  if ( !j_strcmp(Str1, "cxx") )
55  {
56    v8 = sub_140011172(std::cout, (__int64)"flag{where_is_my_true_flag?}");
57    std::basic_ostream<char,std::char_traits<char>>::operator<<(v8, sub_140011127);
58    _exit(0);
59  }
60  system("pause");
61  sub_1400113E3((__int64)&v10, (__int64)&unk_14001DE50);
62  return 0i64;
63}
```

24-31输出了字符串

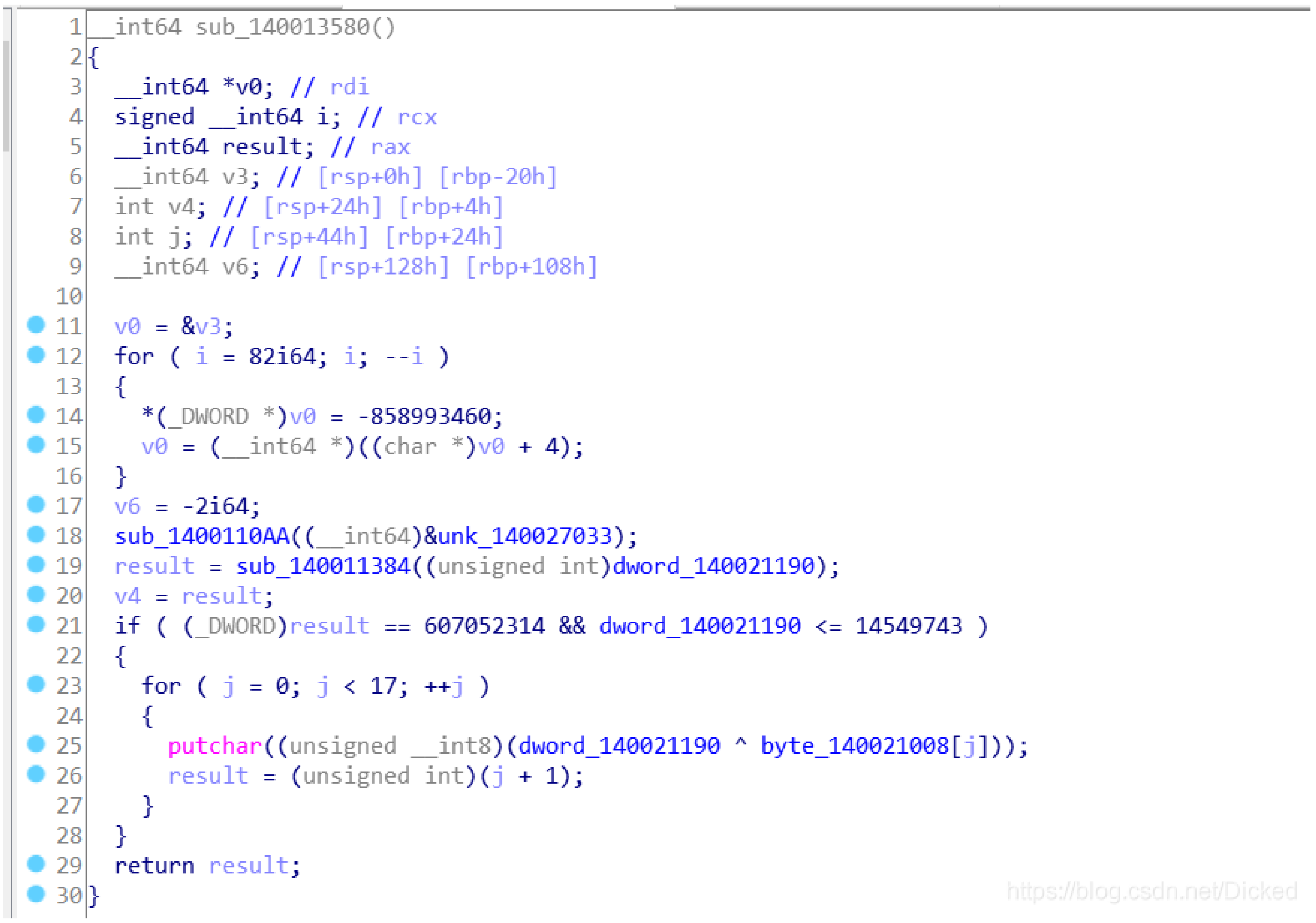


```
.data:0000000140021000                 ;org 140021000h
.data:0000000140021000 ; DWORD pcbBuffer
.data:0000000140021000 pcbBuffer       dd 100h                 ; DATA XREF: sub_140013AA0:loc_140013C07↑o
.data:0000000140021004 byte_140021004  db 'w'                  ; DATA XREF: sub_140013AA0+59↑r
.data:0000000140021005 byte_140021005  db 'e'                  ; DATA XREF: sub_140013AA0+68↑r
.data:0000000140021005                                         ; sub_140013AA0+B3↑r
.data:0000000140021006 byte_140021006  db 'l'                  ; DATA XREF: sub_140013AA0+77↑r
.data:0000000140021007 byte_140021007  db 'c'                  ; DATA XREF: sub_140013AA0+86↑r
.data:0000000140021008 ; _BYTE byte_140021008[17]
.data:0000000140021008 byte_140021008  db '&', ',', '!', 27h, ';', 0Dh, 4, 'u', 'h', '4', '('
.data:0000000140021008                                         ; DATA XREF: sub_140013580+81↑o
.data:0000000140021008                 db '%', 0Eh, '5', '-', 'i', '='
```



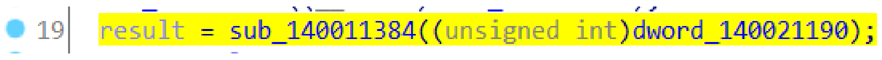
```
.data:0000000140021019 byte_140021019  db 'o'              ; DATA XREF: sub_140013AA0+95↑r
.data:000000014002101A byte_14002101A  db 'm'              ; DATA XREF: sub_140013AA0+A4↑r
.data:000000014002101B                 align 20h
.data:0000000140021020                 dq 0FFFFFFFFh
.data:0000000140021028                 db    0
```

welcome，应该就是这里了。。多了byte_140021008



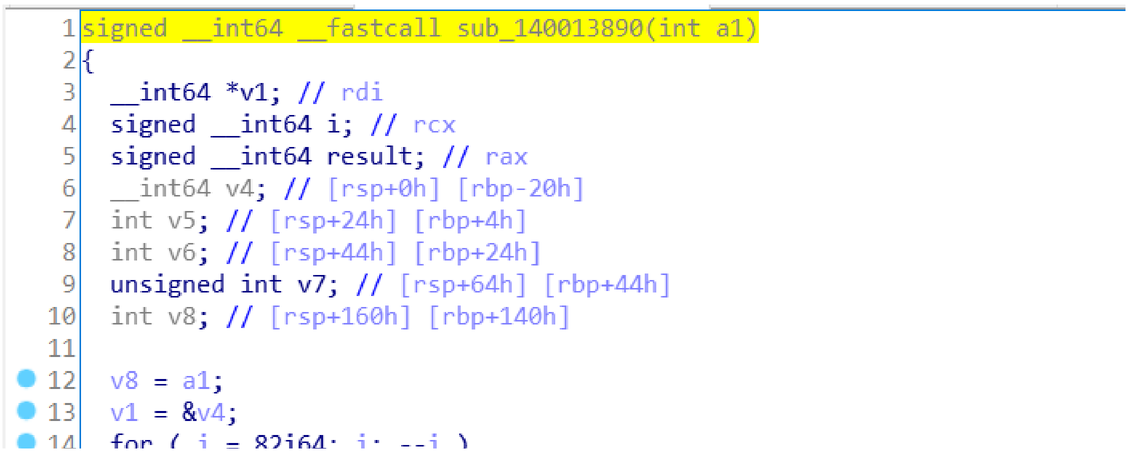
```
__int64 sub_140013580()
{
  __int64 *v0; // rdi
  signed __int64 i; // rcx
  __int64 result; // rax
  __int64 v3; // [rsp+0h] [rbp-20h]
  int v4; // [rsp+24h] [rbp+4h]
  int j; // [rsp+44h] [rbp+24h]
  __int64 v6; // [rsp+128h] [rbp+108h]

  v0 = &v3;
  for ( i = 82i64; i; --i )
  {
    *(_DWORD *)v0 = -858993460;
    v0 = (__int64 *)((char *)v0 + 4);
  }
  v6 = -2i64;
  sub_1400110AA((__int64)&unk_140027033);
  result = sub_140011384((unsigned int)dword_140021190);
  v4 = result;
  if ( (_DWORD)result == 607052314 && dword_140021190 <= 14549743 )
  {
    for ( j = 0; j < 17; ++j )
    {
      putchar((unsigned __int8)(dword_140021190 ^ byte_140021008[j]));
      result = (unsigned int)(j + 1);
    }
  }
  return result;
}
```

就这个函数，看到putchar函数里面有刚刚看见的字符串，有flag那味儿了。
从尾开始搞，打印的肯定是flag了，就是dword_140021190这一段和刚刚找到的字符串逐个异或，然后就是找dword_140021190这一段是啥了

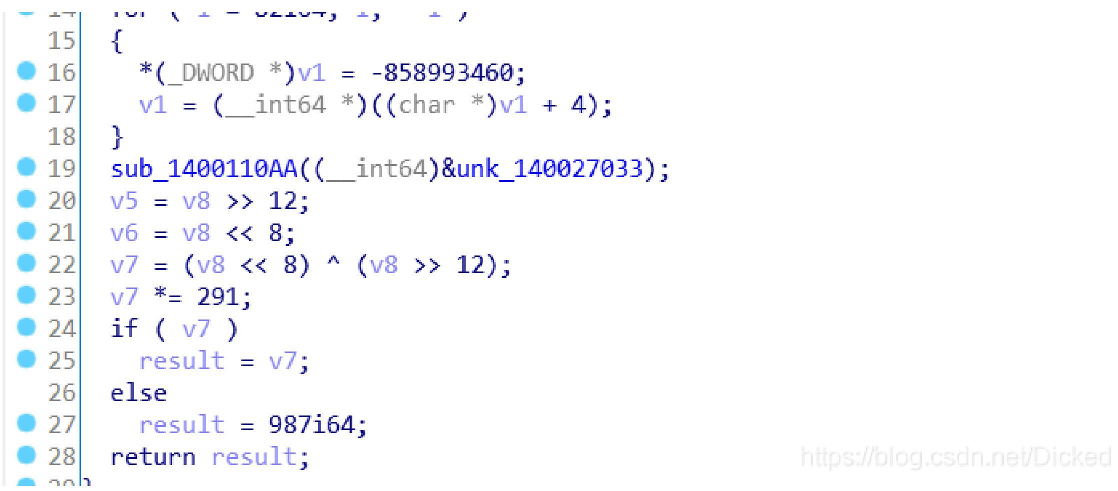


```
  result = sub_140011384((unsigned int)dword_140021190);
```

这里应该是对dword_140021190进行了操作的，进去看看sub_140011384



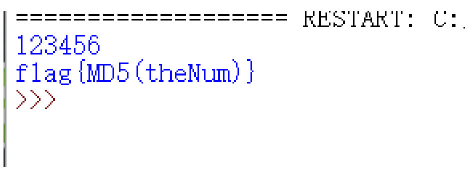
```
signed __int64 __fastcall sub_140013890(int a1)
{
  __int64 *v1; // rdi
  signed __int64 i; // rcx
  signed __int64 result; // rax
  __int64 v4; // [rsp+0h] [rbp-20h]
  int v5; // [rsp+24h] [rbp+4h]
  int v6; // [rsp+44h] [rbp+24h]
  unsigned int v7; // [rsp+64h] [rbp+44h]
  int v8; // [rsp+160h] [rbp+140h]

  v8 = a1;
  v1 = &v4;
  for ( i = 82i64; i; --i )
```



```
  {
    *(_DWORD *)v1 = -858993460;
    v1 = (__int64 *)((char *)v1 + 4);
  }
  sub_1400110AA((__int64)&unk_140027033);
  v5 = v8 >> 12;
  v6 = v8 << 8;
  v7 = (v8 << 8) ^ (v8 >> 12);
  v7 *= 291;
  if ( v7 )
    result = v7;
  else
    result = 987i64;
  return result;
}
```

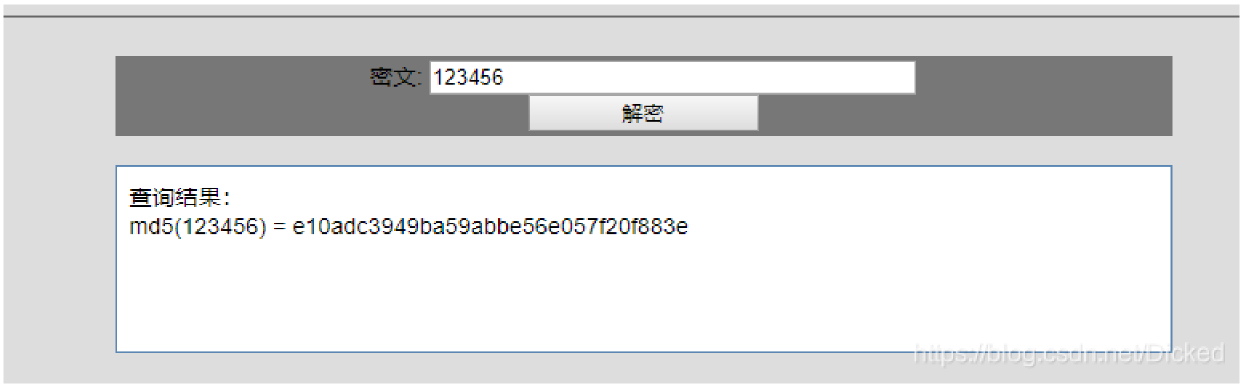
由最初的值经过位移和乘法得到了dword_140021190的最终值，但是我们并不知道最初的值。
也就是说经过一个函数处理dword_140021190返回值等于607052314，同时这个数又小于14549743（也就是之前那个函数中的第21行if语句对dword_140021190的限制）
不知道最初的数据，就爆破吧

```python
v8=0
for i in range(14549743):
    tmp=(((i<<8)^(i>>12))*291)&0xffffffff
    if tmp==607052314:
        v8=i
        break

a=[0x26, 0x2C, 0x21, 0x27, 0x3B, 0x0D, 0x04, 0x75, 0x68, 0x34,
  0x28, 0x25, 0x0E, 0x35, 0x2D, 0x69, 0x3D, 0x6F, 0x6D, 0x00]
print(v8)
for j in range(17):
    print(chr((v8^a[j])&0xff),end="")
```



```
================== RESTART: C:.
123456
flag{MD5(theNum)}
>>>
```

MD5解密，加上题目中说的是flag中没有大写字母，所以知道是小写。

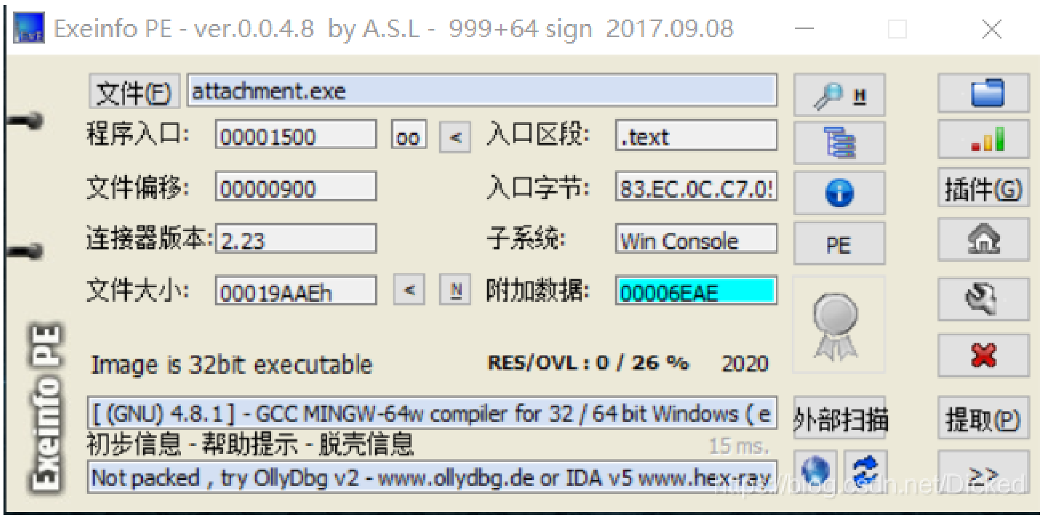


密文: 123456
解密

查询结果:
md5(123456) = e10adc3949ba59abbe56e057f20f883e

flag{e10adc3949ba59abbe56e057f20f883e}
（&0xffffffff和&0xff学到了，开始没有加这个东西怎么跑都跑不出来，后来才知道是v8和flag溢出了，必须对变量进行限制，0xFFFFFFF是最大32个字节，0xFF是最大8个字节）

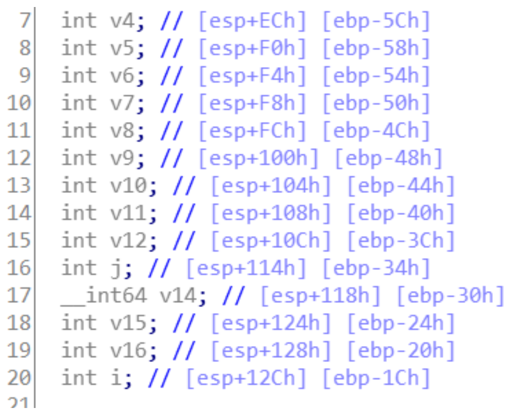
[BJDCTF2020]easy



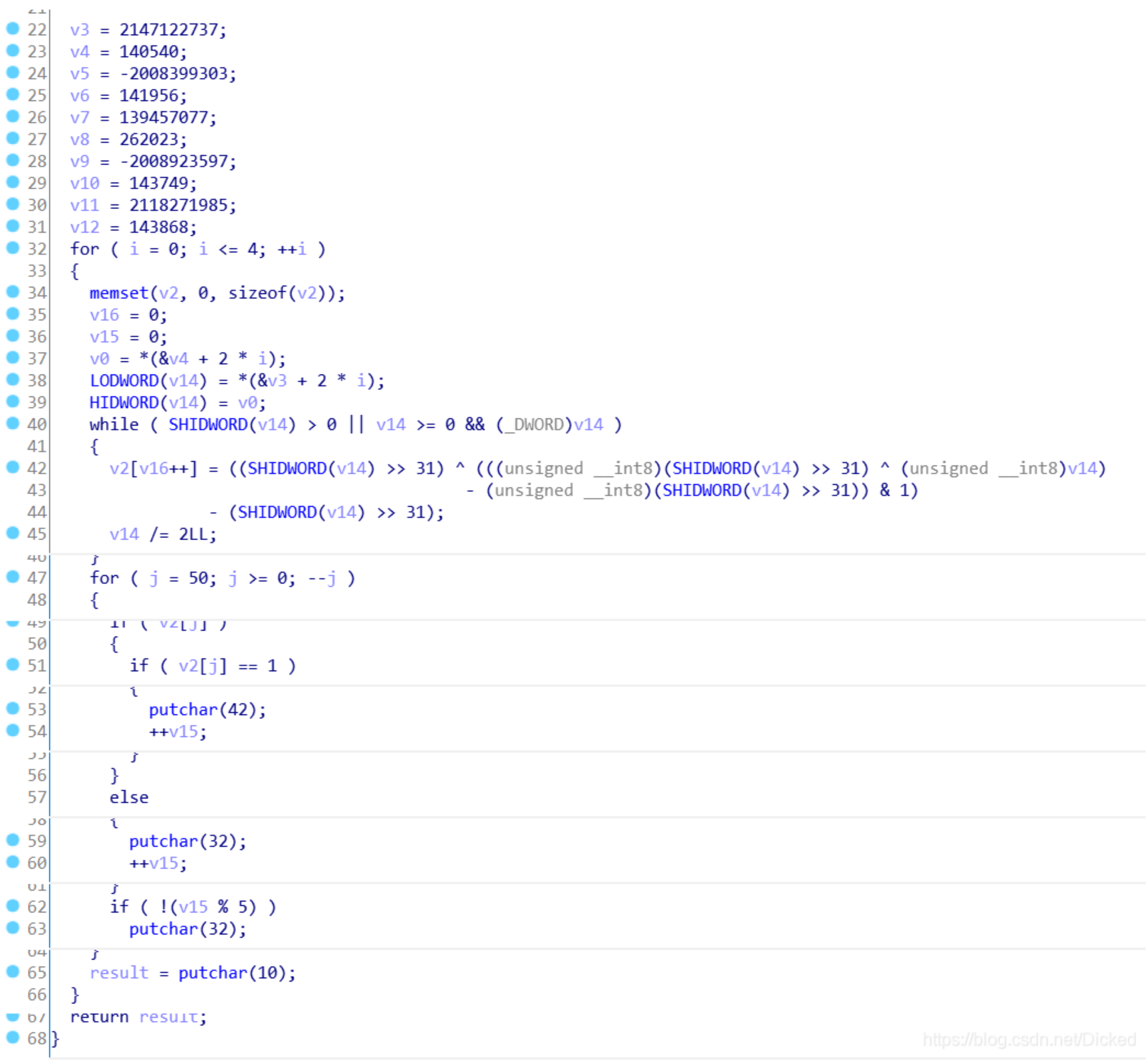
Exeinfo PE - ver.0.0.4.8 by A.S.L - 999+64 sign 2017.09.08

文件(F) attachment.exe
程序入口: 00001500 入口区段: .text
文件偏移: 00000900 入口字节: 83.EC.0C.C7.0!
连接器版本: 2.23 子系统: Win Console
文件大小: 00019AAEh 附加数据: 00006EAE
Image is 32bit executable RES/OVL : 0 / 26 % 2020
[(GNU) 4.8.1] - GCC MINGW-64w compiler for 32 / 64 bit Windows (e
初步信息 - 帮助提示 - 脱壳信息 15 ms.
Not packed , try OllyDbg v2 - www.ollydbg.de or IDA v5 www.hex-ray

main函数中什么都没有
找到ques()函数就是输出我们的flag

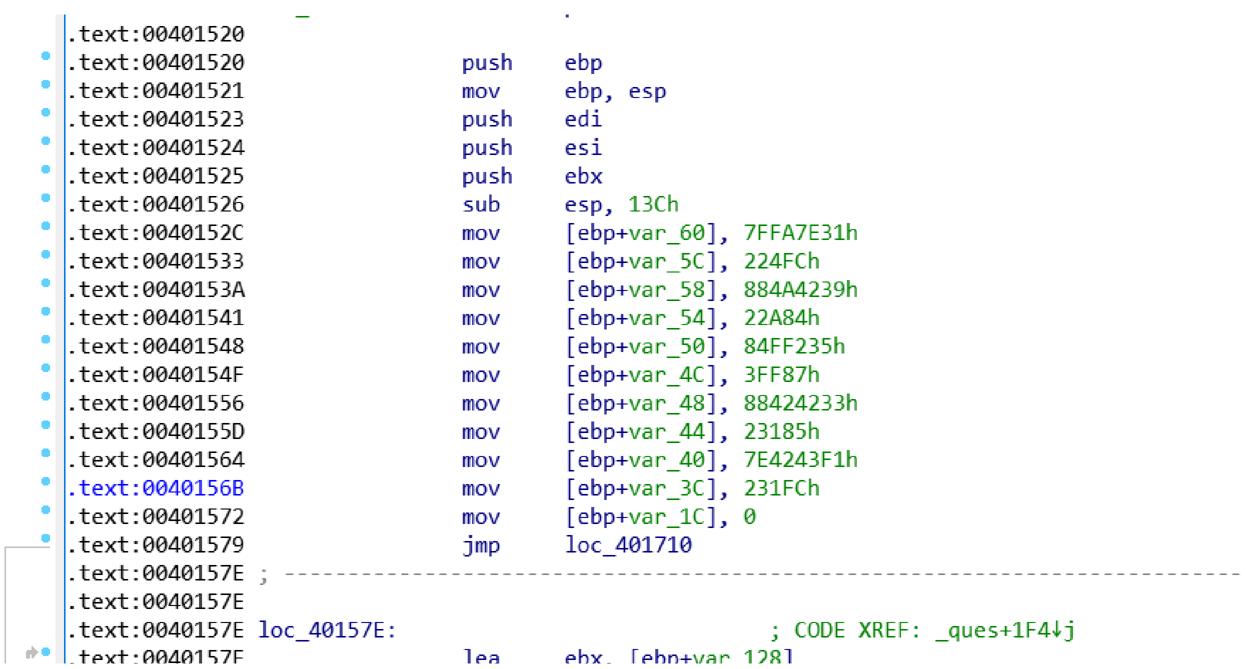


```
int v4; // [esp+ECh] [ebp-5Ch]
int v5; // [esp+F0h] [ebp-58h]
int v6; // [esp+F4h] [ebp-54h]
int v7; // [esp+F8h] [ebp-50h]
int v8; // [esp+FCh] [ebp-4Ch]
int v9; // [esp+100h] [ebp-48h]
int v10; // [esp+104h] [ebp-44h]
int v11; // [esp+108h] [ebp-40h]
int v12; // [esp+10Ch] [ebp-3Ch]
int j; // [esp+114h] [ebp-34h]
__int64 v14; // [esp+118h] [ebp-30h]
int v15; // [esp+124h] [ebp-24h]
int v16; // [esp+128h] [ebp-20h]
int i; // [esp+12Ch] [ebp-1Ch]
```

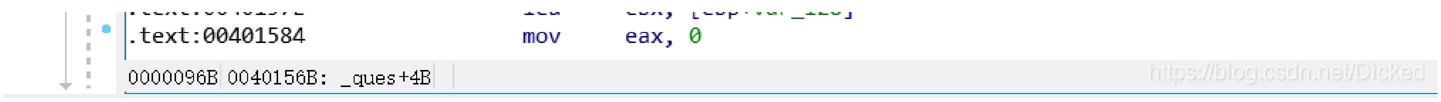


```
  v3 = 2147122737;
  v4 = 140540;
  v5 = -2008399303;
  v6 = 141956;
  v7 = 139457077;
  v8 = 262023;
  v9 = -2008923597;
  v10 = 143749;
  v11 = 2118271985;
  v12 = 143868;
  for ( i = 0; i <= 4; ++i )
  {
    memset(v2, 0, sizeof(v2));
    v16 = 0;
    v15 = 0;
    v0 = *(&v4 + 2 * i);
    LODWORD(v14) = *(&v3 + 2 * i);
    HIDWORD(v14) = v0;
    while ( SHIDWORD(v14) > 0 || v14 >= 0 && (_DWORD)v14 )
    {
      v2[v16++] = ((SHIDWORD(v14) >> 31) ^ (((unsigned __int8)(SHIDWORD(v14) >> 31) ^ (unsigned __int8)v14)
                                           - (unsigned __int8)(SHIDWORD(v14) >> 31)) & 1)
                - (SHIDWORD(v14) >> 31);
      v14 /= 2LL;
    }
    for ( j = 50; j >= 0; --j )
    {
      if ( v2[j] )
      {
        if ( v2[j] == 1 )
        {
          putchar(42);
          ++v15;
        }
      }
      else
      {
        putchar(32);
        ++v15;
      }
      if ( !(v15 % 5) )
        putchar(32);
    }
    result = putchar(10);
  }
  return result;
}
```

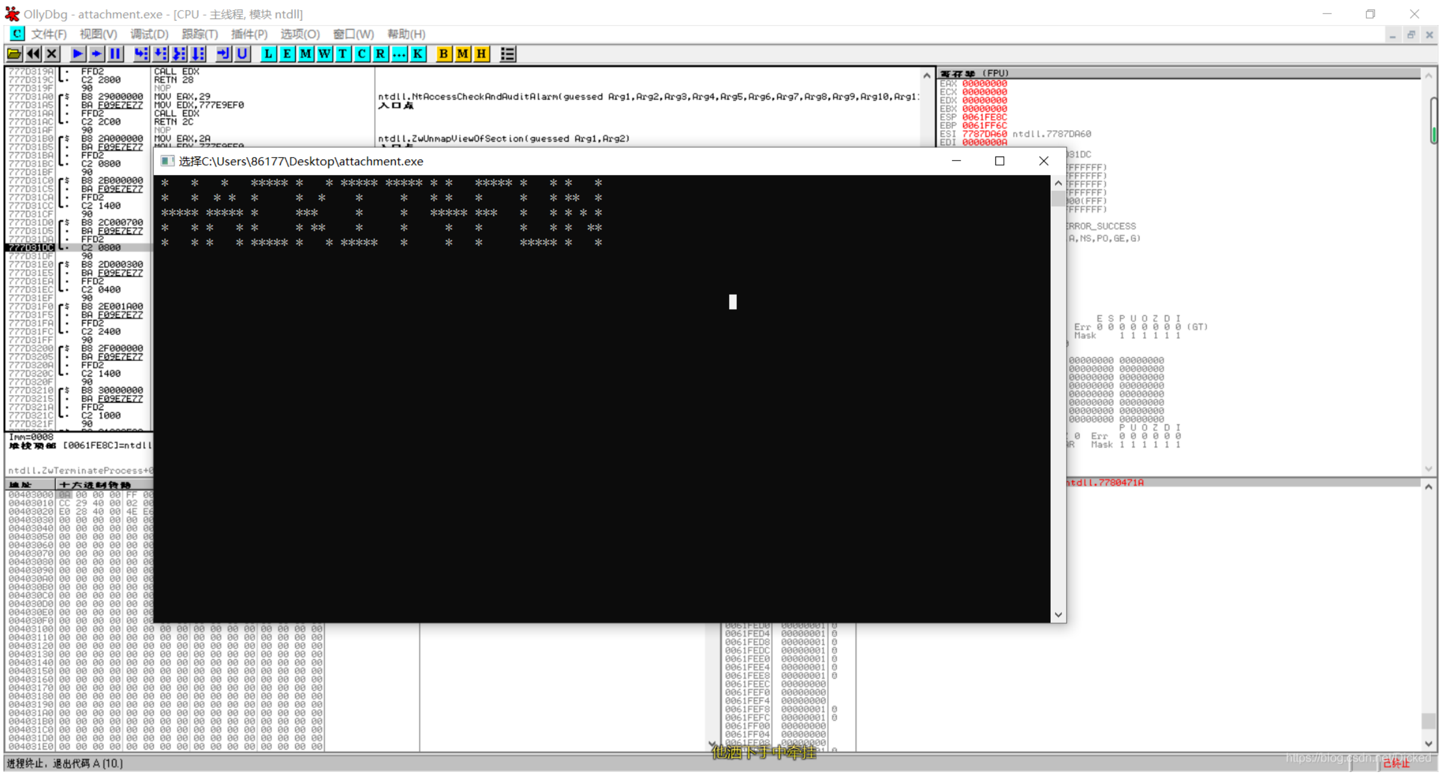
我们可以通过调试修改EIP的地址到ques函数(0x00401520)从而输出flag。



```
.text:00401520
.text:00401520                 push    ebp
.text:00401521                 mov     ebp, esp
.text:00401523                 push    edi
.text:00401524                 push    esi
.text:00401525                 push    ebx
.text:00401526                 sub     esp, 13Ch
.text:0040152C                 mov     [ebp+var_60], 7FFA7E31h
.text:00401533                 mov     [ebp+var_5C], 224FCh
.text:0040153A                 mov     [ebp+var_58], 884A4239h
.text:00401541                 mov     [ebp+var_54], 22A84h
.text:00401548                 mov     [ebp+var_50], 84FF235h
.text:0040154F                 mov     [ebp+var_4C], 3FF87h
.text:00401556                 mov     [ebp+var_48], 88424233h
.text:0040155D                 mov     [ebp+var_44], 23185h
.text:00401564                 mov     [ebp+var_40], 7E4243F1h
.text:0040156B                 mov     [ebp+var_3C], 231FCh
.text:00401572                 mov     [ebp+var_1C], 0
.text:00401579                 jmp     loc_401710
.text:0040157E ; ---------------------------------------------------------------------------
.text:0040157E
.text:0040157E loc_40157E:                             ; CODE XREF: _ques+1F4↓j
.text:0040157E                 lea     ebx, [ebp+var_128]
```



```
.text:00401584                 mov     eax, 0
0000096B 0040156B: _ques+4B
```



OllyDbg - attachment.exe - [CPU - 主线程, 模块 ntdll]
文件(F) 视图(V) 调试(D) 跟踪(T) 插件(P) 选项(O) 窗口(W) 帮助(H)
L E M W T C R ... K B M H

```
CALL EDX
RETN 28
NOP
MOV EAX,29                ntdll.NtAccessCheckAndAuditAlarm(guessed Arg1,Arg2,Arg3,Arg4,Arg5,Arg6,Arg7,Arg8,Arg9,Arg10,Arg1
MOV EDX,777E9EF0          入口点
CALL EDX
RETN 2C
NOP
MOV EAX,2A                ntdll.ZwUnmapViewOfSection(guessed Arg1,Arg2)
```

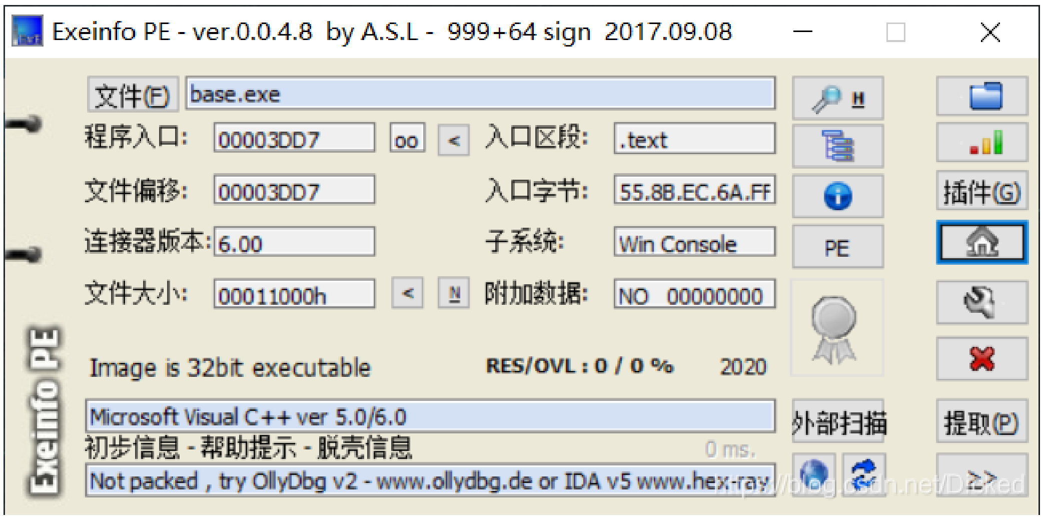
选择C:\Users\86177\Desktop\attachment.exe

```
*   *   *   ***** *   * ***** ***** * *   ***** *   * *   *
*   *  * *  *     *  *    *     *   * *   *     *   * **  *
***** ***** *     ***     *     *   ***** ***   *   * * * *
*   * *   * *     * **    *     *     *   *     *   * *  **
*   * *   * ***** *   * *****   *     *   *     ***** *   *
```

进程终止，退出代码 A (10.)

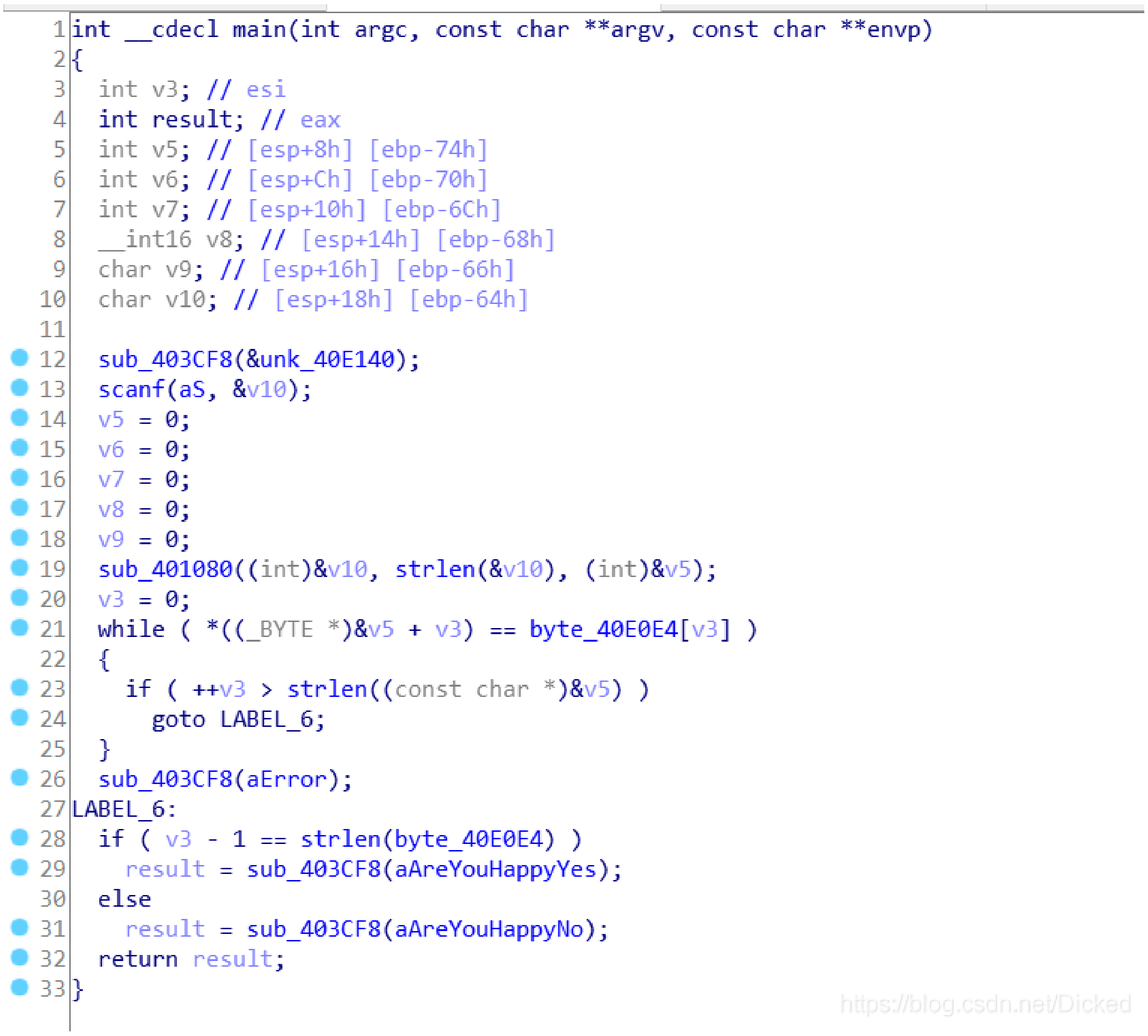
flag{HACKIT4FUN}

[ACTF新生赛2020]usualCrypt



```
Exeinfo PE - ver.0.0.4.8  by A.S.L -  999+64 sign  2017.09.08

文件(F)   base.exe
程序入口:  00003DD7        入口区段:  .text
文件偏移:  00003DD7        入口字节:  55.8B.EC.6A.FF
连接器版本: 6.00            子系统:    Win Console
文件大小:  00011000h       附加数据:  NO  00000000
Image is 32bit executable       RES/OVL : 0 / 0 %    2020
Microsoft Visual C++ ver 5.0/6.0
初步信息 - 帮助提示 - 脱壳信息                          0 ms.
Not packed , try OllyDbg v2 - www.ollydbg.de or IDA v5 www.hex-rays
```

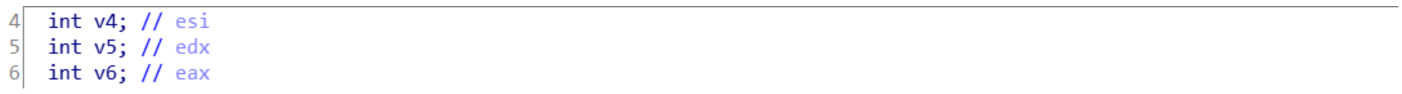
main函数



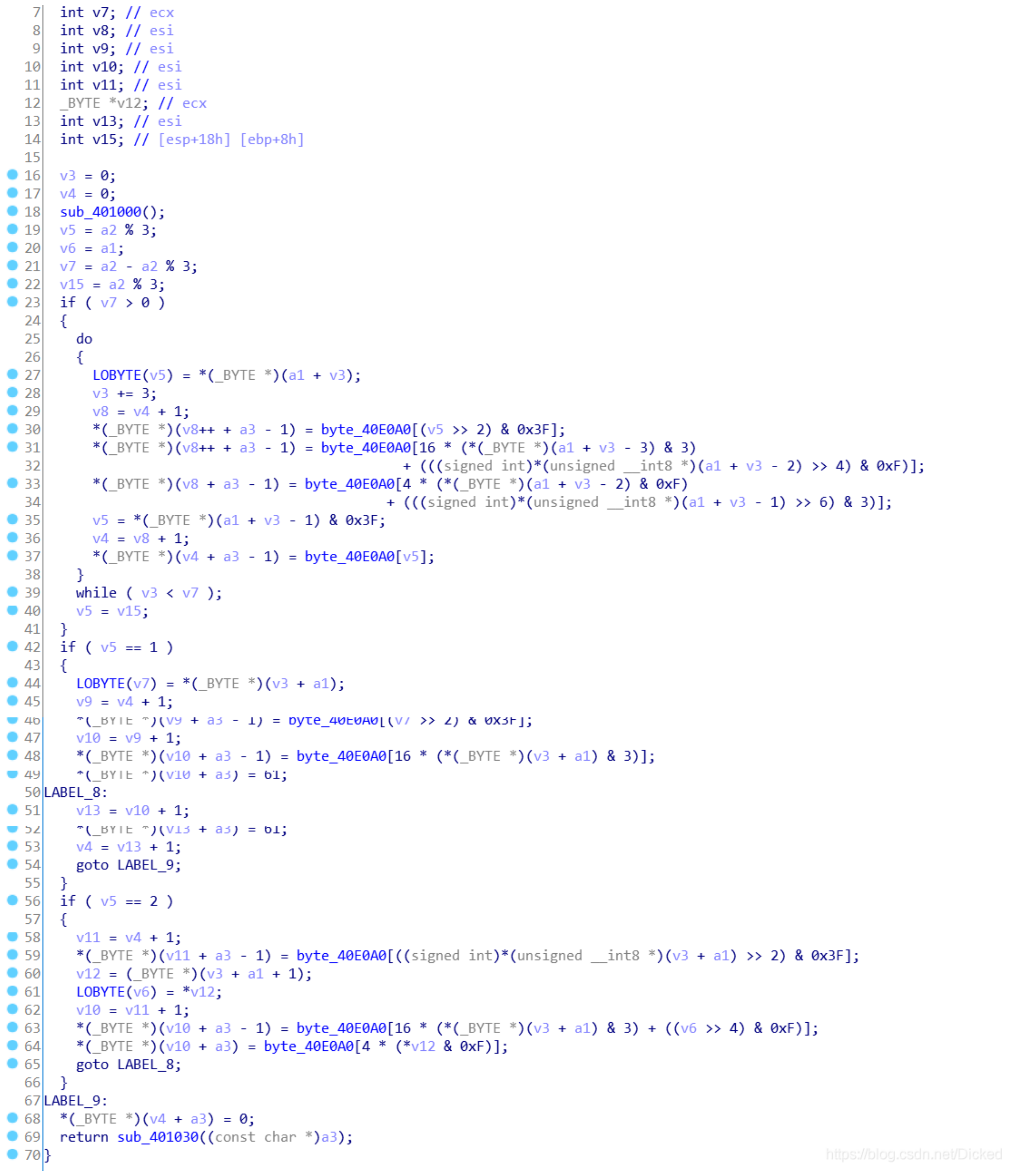
```
 1 int __cdecl main(int argc, const char **argv, const char **envp)
 2 {
 3   int v3; // esi
 4   int result; // eax
 5   int v5; // [esp+8h] [ebp-74h]
 6   int v6; // [esp+Ch] [ebp-70h]
 7   int v7; // [esp+10h] [ebp-6Ch]
 8   __int16 v8; // [esp+14h] [ebp-68h]
 9   char v9; // [esp+16h] [ebp-66h]
10   char v10; // [esp+18h] [ebp-64h]
11
12   sub_403CF8(&unk_40E140);
13   scanf(aS, &v10);
14   v5 = 0;
15   v6 = 0;
16   v7 = 0;
17   v8 = 0;
18   v9 = 0;
19   sub_401080((int)&v10, strlen(&v10), (int)&v5);
20   v3 = 0;
21   while ( *((_BYTE *)&v5 + v3) == byte_40E0E4[v3] )
22   {
23     if ( ++v3 > strlen((const char *)&v5) )
24       goto LABEL_6;
25   }
26   sub_403CF8(aError);
27 LABEL_6:
28   if ( v3 - 1 == strlen(byte_40E0E4) )
29     result = sub_403CF8(aAreYouHappyYes);
30   else
31     result = sub_403CF8(aAreYouHappyNo);
32   return result;
33 }
```

12 和 26 行的sub_403CF8显然是输出函数，21 - 25 行的while()是字符串比较的过程，看来还是一个输入 flag 加密验证正确性的模型。

发现19行的sub_401080是加密函数，进入分析

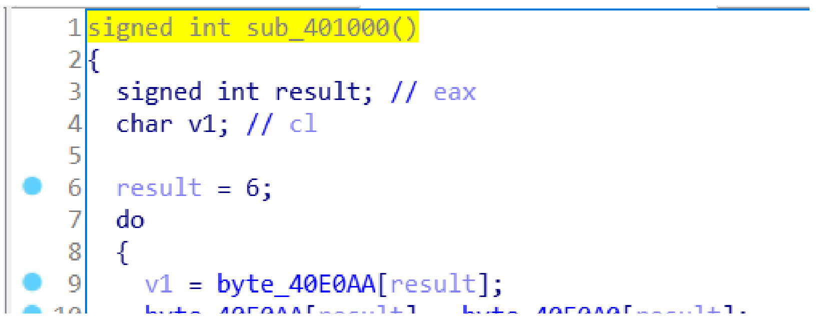


```
4   int v4; // esi
5   int v5; // edx
6   int v6; // eax
```

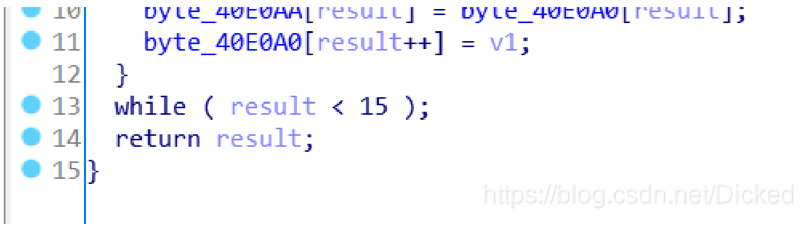


```
  int v7; // ecx
  int v8; // esi
  int v9; // esi
  int v10; // esi
  int v11; // esi
  _BYTE *v12; // ecx
  int v13; // esi
  int v15; // [esp+18h] [ebp+8h]

  v3 = 0;
  v4 = 0;
  sub_401000();
  v5 = a2 % 3;
  v6 = a1;
  v7 = a2 - a2 % 3;
  v15 = a2 % 3;
  if ( v7 > 0 )
  {
    do
    {
      LOBYTE(v5) = *(_BYTE *)(a1 + v3);
      v3 += 3;
      v8 = v4 + 1;
      *(_BYTE *)(v8++ + a3 - 1) = byte_40E0A0[(v5 >> 2) & 0x3F];
      *(_BYTE *)(v8++ + a3 - 1) = byte_40E0A0[16 * (*(_BYTE *)(a1 + v3 - 3) & 3)
                                            + (((signed int)*(unsigned __int8 *)(a1 + v3 - 2) >> 4) & 0xF)];
      *(_BYTE *)(v8 + a3 - 1) = byte_40E0A0[4 * (*(_BYTE *)(a1 + v3 - 2) & 0xF)
                                          + (((signed int)*(unsigned __int8 *)(a1 + v3 - 1) >> 6) & 3)];
      v5 = *(_BYTE *)(a1 + v3 - 1) & 0x3F;
      v4 = v8 + 1;
      *(_BYTE *)(v4 + a3 - 1) = byte_40E0A0[v5];
    }
    while ( v3 < v7 );
    v5 = v15;
  }
  if ( v5 == 1 )
  {
    LOBYTE(v7) = *(_BYTE *)(v3 + a1);
    v9 = v4 + 1;
    *(_BYTE *)(v9 + a3 - 1) = byte_40E0A0[(v7 >> 2) & 0x3F];
    v10 = v9 + 1;
    *(_BYTE *)(v10 + a3 - 1) = byte_40E0A0[16 * (*(_BYTE *)(v3 + a1) & 3)];
    *(_BYTE *)(v10 + a3) = 61;
LABEL_8:
    v13 = v10 + 1;
    *(_BYTE *)(v13 + a3) = 61;
    v4 = v13 + 1;
    goto LABEL_9;
  }
  if ( v5 == 2 )
  {
    v11 = v4 + 1;
    *(_BYTE *)(v11 + a3 - 1) = byte_40E0A0[((signed int)*(unsigned __int8 *)(v3 + a1) >> 2) & 0x3F];
    v12 = (_BYTE *)(v3 + a1 + 1);
    LOBYTE(v6) = *v12;
    v10 = v11 + 1;
    *(_BYTE *)(v10 + a3 - 1) = byte_40E0A0[16 * (*(_BYTE *)(v3 + a1) & 3) + ((v6 >> 4) & 0xF)];
    *(_BYTE *)(v10 + a3) = byte_40E0A0[4 * (*v12 & 0xF)];
    goto LABEL_8;
  }
LABEL_9:
  *(_BYTE *)(v4 + a3) = 0;
  return sub_401030((const char *)a3);
}
```

主体的代码是base64加密算法，但在加密前执行了sub_401000()，这是一个改变密钥对应表的函数



```
signed int sub_401000()
{
  signed int result; // eax
  char v1; // cl

  result = 6;
  do
  {
    v1 = byte_40E0AA[result];
    byte_40E0AA[result] = byte_40E0A0[result];
```



```
    byte_40E0AA[result] = byte_40E0A0[result];
    byte_40E0A0[result++] = v1;
  }
  while ( result < 15 );
  return result;
}
```

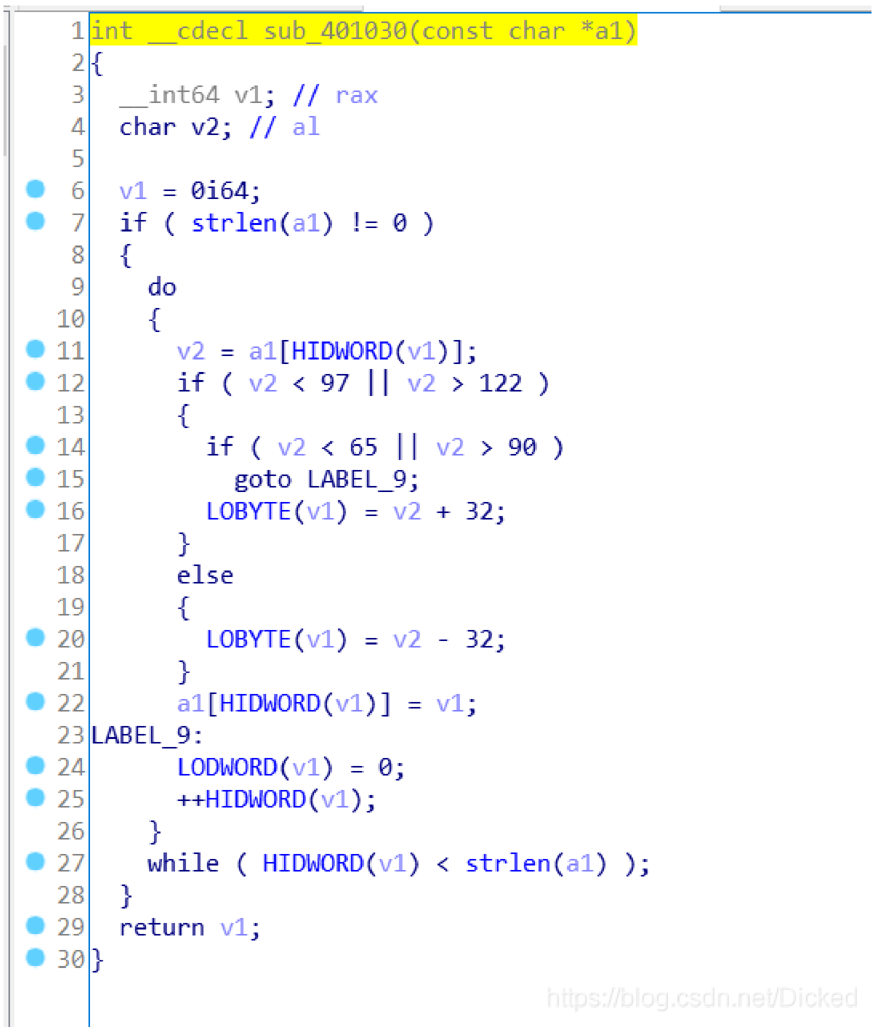
byte_40E0A0和byte_40E0AA看起来是两个数组，其实都是一个字符串上（即密钥表串）的地址，

所以这个子函数的功能就是把一个字符串上的范围内的字符按偏移索引（result）两两交换

在操作后，base64 的加密规则未变，但是部分字符的意义发生了变化

在写逆向脚本代码的时候可以用 python 的字典构造原密钥与改过的密钥的对应关系

在加密后，该函数又调用了sub_401030



```
int __cdecl sub_401030(const char *a1)
{
  __int64 v1; // rax
  char v2; // al

  v1 = 0i64;
  if ( strlen(a1) != 0 )
  {
    do
    {
      v2 = a1[HIDWORD(v1)];
      if ( v2 < 97 || v2 > 122 )
      {
        if ( v2 < 65 || v2 > 90 )
          goto LABEL_9;
        LOBYTE(v1) = v2 + 32;
      }
      else
      {
        LOBYTE(v1) = v2 - 32;
      }
      a1[HIDWORD(v1)] = v1;
LABEL_9:
      LODWORD(v1) = 0;
      ++HIDWORD(v1);
    }
    while ( HIDWORD(v1) < strlen(a1) );
  }
  return v1;
}
```

这个子函数把字符串每个英文字符进行了大小写转换，最后形成数据段保存的密码byte_40E0E4。

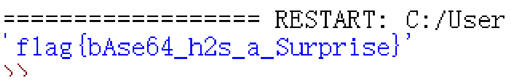
该程序进行了多重加密，在取出密码之后首先要进行大小写转换，令其出于 base64（更改密钥表后）加密后的形态

第二步是还原经 base64（更改密钥表后）加密字符的原含义，还原规则即sub_401000()的交换

最后得到了真实的nbase64n加密字符串，解密即可得到flag

脚本如下

```python
import base64

flag = ''; dict = {}; offset = 10
orgin = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/'
for i in range(len(orgin)):
    dict[orgin[i]] = orgin[i]
for i in range(6, 15): #sub_401000()
    dict[orgin[i]] , dict[orgin[i+offset]] = dict[orgin[i+offset]] , dict[orgin[i]] # 恢复base64密钥表
secret = 'zMXHz3TIgnxLxJhFAdtZn2fFk3lYCrtPC2l9'.swapcase() #sub_401030()
for i in range(len(secret)):
    flag += dict[secret[i]]
flag = base64.b64decode(flag)
print(flag)
```



```
=================== RESTART: C:/User
'flag{bAse64_h2s_a_Surprise}'
>>
```