1-easystr

```
int main()
{

  char Str1[40];
  char Str2[40] = "Welcome_to_the_reverse_world!";

  for (int i = 0; i < 20; ++i )
  {
    if ( Str2[i] == 'l' )
    {
      Str2[i] = '1';
    }
    else if ( Str2[i] == 'o' )
    {
      Str2[i] = '0';
    }
  }
  memset(Str1, 0, sizeof(Str1));
  puts("input the flag:");
  cin>>Str1;
  if ( !strcmp(Str1, Str2) )
    puts("right!");
  else
    puts("wrong!");
  system("pause");
  return 10086;
}
```

本题为新手福利。代码如上，把**Welcome_to_the_reverse_world!前20位**中的**字母l**换成**数字1**，**字母o**换成**数字0**即得到flag，flag为**We1c0me_t0_the_reverse_world!**

2-xorrr

~~这题有（bug）有趣的产品特性，当我们输入错误答案时返回This_is_flag。而输入正确答案则什么都不返回。表现出出题人对我们做错题后的鼓励。~~

```
int main()
{

  char v5[5][16] = {"X1j3y5a7u9t;`=|",";5w7n9 ;l=n?)A-","s9?;}=j?|AxChEj"};
  char str[30];
  cin>>str;
  for (int i = 0; i < 24; ++i )
  {
    if ( ((i + 8) ^ v5[i % 3][2 * (i / 3)]) - str[i] != 2 )
    {
```

```
      puts("flag{Th1s_1s_a_f1ag}");
      break;
    }
  }
  system("pause");
  return 0;
}
```

代码如上，这题就考了个字符串地址的寻找，能想到构造v5即能解出题目。

输入str，如果**str[i]+2 == (i+8)^v5[i%3] [2*(i/3)]对0<i<24均成立则正确**

通过穷举我们可以知道flag为**N0w_y0u_kn0w_what_x0r_1s**。

3-maze

```
int main()
{
    char str[20] = {0};
    int v9;
    int i;
    unsigned int v4;
    char byte_402180[] = "#11000011000100001010*1100";

    v4 = 0;
    cin >> str;
    if (strlen(str) == 10 )
    {
        i = 0;
        while ( i < 10 )
        {
            v9 = str[i];
            v9 -= 'a';
            switch ( v9 )
            {
                case 0:
                    --v4;
                    break;
                case 3:
                    ++v4;
                    break;
                case 18:
                    v4 += 5;
                    break;
                case 22:
                    v4 -= 5;
                    break;
                default:
                    return 0;
            }
```

```
            if ( v4 > 24 || byte_402180[v4] == '0' )
                return 0;

            ++i;
        }
        if ( byte_402180[v4] == '*' )
            puts("Right!");
    }
    return 0;
}
```

代码如上（错误路径用return 0简单表示）

本题接收了一个10位长的字符串str，并要求字符串必须由‘a’, ‘s’ , ‘d’ ,‘u’表示。然后**遍历str，根据不同字母对数字v4进行加减操作。**

数字**v4其实是一个指向数组byte_402180的指针**，本题要求我们在每次**对v4操作后，v4要保持在0到24之间并且byte_402180[v4] != ‘0’**。如果**遍历结束后，若byte_402180[v4] == ‘*’**则获得flag。

```
.rdata:00402180 23                      byte_402180 db 23h                      ; DATA XREF: _main+F5↑r
.rdata:00402180                                                                 ; _main+11B↑r
.rdata:00402181 31 31 30 30 30 30 31 31 30 30+a11000011000010 db '11000011000010000110*1100',0
.rdata:0040219A 00 00 00 00 00 00           align 10h
```

byte_402180的具体值可由上看出，为从00402180到0040219A的数据，即**"#11000011000010000110*1100"**这串字符串。因而**答案一眼鉴定为ddsdssasaa**。放入md翻译即可

4-array

```
#include<bits/stdc++.h>

using namespace std;

char tar[20] = "2=7#+1;?5O:9&?2?9=+%!";
char byte_403000[4] = {255,255,255,255};

bool sub_401080(char * a){
    char v2[20]={0};
    for(int i=0;i<20;i++){
        v2[i] = byte_403000[a[i]];
        //v2[i] = 158 - a[i];
    }
    for(int j=0;j<20;j++){
        if(v2[j] != tar[j])
            return false;
    }
    return true;
}

int main()
{
    char str[36]={0};
```

```
    puts("Please input flag:");
    cin>>str;
    if (strlen(str) == 20 && sub_401080(str))
        puts("Right!");
    else
        puts("Wrong!");
    system("pause");
    return 10086;
}
```

代码如上，输入一个20位的str，若**byte_403000[str[i]] == tar[i]对0<i<20都成立**则得到flag。

本题难点在于**byte_403000是一个4位的char数组，而且每一位都是0xFF，无法被用来匹配tar**

```
.data:00403020 7E 7D 7C 7B 7A 79 78 77 76 75+aZyxwvutsrqponm db '~}|{zyxwvutsrqponmlkjihgfedcba`_^]\[ZYXWVUTSRQPONMLKJIHGFEDCBA@?>=<;:9876543210/.-,+*)(',27h,'&'
.data:00403020 74 73 72 71 70 6F 6E 6D 6C 6B+db '%$#"! ',0
```

但是如上图，我们发现从data:403020到data:403080有一个隐藏的数组，内容是反过来的ASLL码表。由此我们可以**让byte_403000[str[i]]越界到403020以后。**具体而言，对数组byte_403000：

byte_403000[str[i]] = 126 - (str[i] - 0x20)

因此**现在只需tar[i] == 126 - (str[i] - 0x20)即可。**解出**flag{smc_index_easy}**

5-rome

```
int main(){
    char destination[100], va0[100], str[52];
    memset(str,0,50);
    memset(destination,0,100);
    memset(va0,0,100);
    cin >> str;
    strcpy(destination, "Zxb3xo_qe4_Dob@q");
    if ( strlen(str) == 16 )
    {
        for ( int i = 0; i < strlen(str); ++i )
        {
            if ( str[i] < 'A' || str[i] > 'Z' ) //不是A~Z
            {
                if ( str[i] < 'a' || str[i] > 'z' ) //不是a~z
                {
                    va0[i] = str[i];
                }
                else
                {
                    int j = str[i] - 'd';
                    for ( ; j < 0; j += 26){}
                    va0[i] = j + 'a';
                }
            }
            else //A到Z
            {
                int j = str[i] - 'D';
```

```
                for ( ; j < 0; j += 26 ){}
                va0[i] = j + 'A';
            }

            if ( va0[i] != destination[i] )
            {
                return 0;
            }
        }
        puts("right!");
    }
    return 0;
}
```

（由本题的名字可知本题是凯撒加密（bushi））

程序逻辑如上，即输入一个16位的字符串str，然后对str进行凯撒加密使每个字母前移3个字母，若str匹配"Zxb3xo_qe4_Dob@q"则通过。

所以我们把"Zxb3xo_qe4_Dob@q"的每一个英文字母后移3个字母即可得到flag，即
“Cae3ar_th4_Gre@t”

6-equation

```
int main()
{
    char str[40];
    char tar[6] = {0};
    cin >> str;
    if ( strlen(str) == 32 )
    {
        tar[4] += 16;
        tar[3] += str[3] * tar[4];
        tar[4] += 3;
        tar[3] += str[2] * tar[4];
        tar[4] -= 10;
        tar[0] += str[3] * tar[4];
        tar[4] -= 2;
        tar[1] += str[2] * tar[4];
        tar[4] += 13;
        tar[0] += str[1] * tar[4];
        tar[4] -= 8;
        tar[2] += str[1] * tar[4];
        tar[4] -= 7;
        tar[2] -= 3481;
        tar[4] += 3;
        tar[1] -= 2422;
        tar[4] += 9;
        tar[0] += str[2] * tar[4];
        tar[4] -= 2;
        tar[2] += str[2] * tar[4];
```

```
        tar[4] -= 6;
        tar[0] -= 4518;
        tar[4] += 7;
        tar[3] -= 5006;
        tar[4] += 9;
        tar[1] += str[0] * tar[4];
        tar[4] += 5;
        tar[0] += str[0] * tar[4];
        tar[4]++;
        tar[2] += str[0] * tar[4];
        tar[4] -= 8;
        tar[2] += str[3] * tar[4];
        tar[4] += 14;
        tar[3] += str[0] * tar[4];
        tar[4] -= 11;
        tar[1] += str[3] * tar[4];
        tar[4] += 3;
        tar[3] += tar[1] * tar[4];
        tar[4] -= 2;
        tar[1] += tar[1] * tar[4];
        for(int i=0;i<=4;i++)
            if(tar[i] != 0)
                return 0;

        char qword_140004040[65] = {0x64,0,0x10,0,0x7C,0,0x7C,0,
                                    0x22,0,0x17,0,0x2F,0,0x34,0,
                                    0x13,0,0x13,0,0xB,0,0x3B,0,
                                    0x1F,0,0x1B,0,0xF,0,0,0
                                    0x1E,0,0x5,0,0x72,0,0x3C,0,
                                    0x62,0,0x1F,0,0x36,0,0x4,0,
                                    0x1F,0,0x2E,0,0x32,0,0x3F,0,
                                    0,0,0,0,0,0,0,0};

        for (int j = 0; j < 28; ++j)
        {
            if ( (str[j % 4] ^ str[j + 4]) != qword_140004040[2*j] )
            return 0;
        }
    }
    puts("right!");
    return 0;
}
```

本题代码如上。输入一个32位的str，前4位用来解决一道**4元一次方程组**，穷举可得**前四位为"QTEM"**

后28位要求要通过与前四位异或的方式匹配上qword_140004040。

私以为本题的难点就是求出qword_140004040，该数组本来是longlong[8]类型，被转换成char[64]。

以第一个数字举例，**longlong[0] = 7C007C00100064，将该数字无符号扩展到64位，然后再按小端方式保存。可以得到char[0]到char[8]分别就是0x64,0x00,0x10,0x00,x7C,0x00,0x7C,0x00。**后面同理。

然后再穷举str的后28位即可得到答案。flag为**QTEM5D91sCjyBGNvNOJMOQ7q3KsINzwr**

7-click

本题重在逻辑推理。由于调用了一堆Qt库，我们很难看到全貌，但是在查阅反汇编代码时。我发现了两个可疑点。

第一个可疑点如下，在某个地方存储了5个意义不明的Qstring，但很遗憾，经过尝试后均不是flag

```
.text:00000001400010DC 48 8D 15 F5 44 00 00          lea     rdx, aAhfrcwstmje4mj            ; "aHFrcWstMjE4MjMtamNoZGts"
.text:00000001400010E3 FF 15 2F 30 00 00             call    cs:??0QString@@QEAA@PEBD@Z      ; QString::QString(char const *)
.text:00000001400010E3
.text:00000001400010E9 90                            nop
.text:00000001400010EA 48 8D 4B 50                   lea     rcx, [rbx+50h]
.text:00000001400010EE 48 8D 15 03 45 00 00          lea     rdx, aYwjjzgutmzg0od            ; "YWJjZGUtMzg0ODMta2RraHly"
.text:00000001400010F5 FF 15 1D 30 00 00             call    cs:??0QString@@QEAA@PEBD@Z      ; QString::QString(char const *)
.text:00000001400010F5
.text:00000001400010FB 90                            nop
.text:00000001400010FC                               ;   try {
.text:00000001400010FC 48 8D 4B 68                   lea     rcx, [rbx+68h]
.text:0000000140001100 48 8D 15 11 45 00 00          lea     rdx, aYwjjzgutmtiznd             ; "YWJjZGUtMTIzNDUtZ2hpamts"
.text:0000000140001107 FF 15 0B 30 00 00             call    cs:??0QString@@QEAA@PEBD@Z      ; QString::QString(char const *)
.text:0000000140001107
.text:000000014000110D 90                            nop
.text:000000014000110D                               ;   } // starts at 1400010FC
.text:000000014000110E                               ;   try {
.text:000000014000110E 48 8D 8B 80 00 00 00          lea     rcx, [rbx+80h]
.text:0000000140001115 48 8D 15 1C 45 00 00          lea     rdx, aEhh4exktmtiznd             ; "eHh4eXktMTIzNDUtamtvcG1w"
.text:000000014000111C FF 15 F6 2F 00 00             call    cs:??0QString@@QEAA@PEBD@Z      ; QString::QString(char const *)
.text:000000014000111C
.text:0000000140001122 90                            nop
.text:0000000140001122                               ;   } // starts at 14000110E
.text:0000000140001123                               ;   try {
.text:0000000140001123 48 8D 8B 98 00 00 00          lea     rcx, [rbx+98h]
.text:000000014000112A 48 8D 15 27 45 00 00          lea     rdx, aUxrmdw4tmtawod             ; "UXRmdW4tMTAwODYtR1VJdG9v"
.text:0000000140001131 FF 15 E1 2F 00 00             call    cs:??0QString@@QEAA@PEBD@Z      ; QString::QString(char const *)
```

第二个可疑点如下，在主程序的调用函数中，我发现了该程序调用了Qstring::toLatin1把某个Qstring转化为QByteArray，然后又调用QByteArray::fromBase64进行了Base64解码，之后又将其转回Qstring并输出。考虑到该程序输出的文字类信息除了**点击1k次出现的伪flag**外，其余均能在反汇编代码中找到。我们猜测**伪flag是进行了Base64解码后的字符串。**

```
call    cs:?toLatin1@QString@@QEGBA?AVQByteArray@@XZ ; QString::toLatin1(void)
nop
mov     rax, [rdi+28h]
mov     rbx, [rax+18h]
xor     r8d, r8d
lea     rdx, [rsp+88h+var_28]
lea     rcx, [rsp+88h+var_40]
call    cs:?fromBase64@QByteArray@@SA?AV1@AEBV1@V?$QFlags@W4Base64Option@QByteArray@@@@@Z ; QByteArray::fromBase64(QByteArray const &,QFlags<QByteArray::Base64Option>)
```

于是使用py脚本点击1k次，将得到的伪flag{xxxyy-12345-jkopmp}进行Base64编码（伪flag可能不同，由于发现了Qtime，推测是和点击1k次所用的时间有关）。得到eHh4eXktMTIzNDUtamtvcG1w，正好对应了上面第四个神秘Qstring。我们将剩下四个神秘Qstring依次进行Base64解码再尝试输入即可获得flag

真正的flag为**Qtfun-10086-GUItoo**

~~（Qt一点也不fun）~~

8-junkcode

本题使用了大量垃圾代码和错误指令来迷惑ida，使之无法判断出main函数的范围，因而无法使用图形化界面和F5生成C语言代码

我们可以使用ida自带的**undefine和code**两个操作对汇编代码进行合理地编辑，使之逻辑通顺。

```
.text:004013C3 74 03                         jz      short near ptr loc_4013C7+1
.text:004013C3
.text:004013C5 75 01                         jnz     short near ptr loc_4013C7+1
.text:004013C5
.text:004013C7
.text:004013C7                               loc_4013C7:                             ; CODE XREF: .text:004013C3↑j
.text:004013C7                                                                       ; .text:004013C5↑j
.text:004013C7 E9 8D 4D B0 51                jmp     near ptr 51F06159h
.text:004013C7
.text:004013C7                               ; ---------------------------------------------------------------------------
.text:004013CC E8 C1 0C 00 00 83 C4 04 50 8D+dd 0CC1E8h, 4C48300h, 4C958D50h, 52FFFFFFh, 50B0458Dh, 0FFFD7BE8h, 0CC483FFh, 0C085C033h, 750B74h, 50C3235Eh
.text:004013CC 95 4C FF FF FF 52 8D 45 B0 50+dd 374C333h, 0FC45C7C2h, 0
.text:00401400 EB 09 8B 4D FC 83 C1 01 89 4D+dd 4D8B09EBh, 1C183FCh, 83FC4D89h, 7D04FC7Dh, 89D2336Bh, 5589F055h, 0F85589F4h, 4589C033h, 0E84589E4h
.text:00401400 FC 83 7D FC 04 7D 6B 33 D2 89+dd 6AEC4589h, 0FC4D6B08h, 0D948D08h, 0FFFFFF4Ch, 0F0458D52h
.text:00401438 50 FF 15                      db 50h, 0FFh, 15h
```

例如上述代码，**通过4013C3和4013C5两行我们知道代码一定跳转到4013C8**，因此**将4014C7处地代码undefine掉，然后从4014C8处开始code**，结果如下图，变成了通畅的代码。其他地方同理。

```
.text:004013C3 74 03                    jz      short loc_4013C8
.text:004013C3
.text:004013C5 75 01                    jnz     short loc_4013C8
.text:004013C5
.text:004013C5                          ; ---------------------------------------------------------------------------
.text:004013C7 E9                       db 0E9h
.text:004013C8                          ; ---------------------------------------------------------------------------
.text:004013C8
.text:004013C8                          loc_4013C8:                             ; CODE XREF: .text:004013C3↑j
.text:004013C8                                                                  ; .text:004013C5↑j
.text:004013C8 8D 4D B0                 lea     ecx, [ebp-50h]
.text:004013CB 51                       push    ecx
.text:004013CC E8 C1 0C 00 00           call    strlen
```

由此我们可以得到正确的C语言代码：

```
void Base64(char* a1, char* a2, unsigned int len);
int Caeser(char *Str, char* a2, int a3);
int main()
{
  char tar[45] = "P1Ekb1UxW9ErWC6ZVUKiKgMaLSEgS5gpyZOrSQG3tP8g";
  char v118[100]={0};
  char vb4[100] ={0};
  char str[52]  ={0};
  cin>>str;
  while(true){

    Base64(str,vb4,strlen(str));
    for(int i=0;i<4;i++){
      char v10[12];
      char v1c[12];
      strncpy(v10,vb4+11*i,11);
      Caeser(v10,v1c,2*i+1);
      strncpy(v118+11*i,v1c,11);
    }
    for(int i=0;i<45;i++)
      if(v118[i] != tar[i])
        return 0;

  }
  cout<<"Right";
}
```

本题输入了一个长度为44的str字符串，**先对其进行Base64加密，再将其分成4份，分别对其进行后移2*i+1（0<i<4）的凯撒加密。若其等于P1Ekb1UxW9ErWC6ZVUKiKgMaLSEgS5gpyZOrSQG3tP8g则flag正确**

所以我们先反过来凯撒解密得到**Q2Flc2VyX0FuZF9CYXNlNjRfQXJlX0ludGVyZXN0aW5n**，然后再对其进行Base64解码得到**Caeser_And_Base64_Are_Interesting**，即flag。

9-multithreading

```
unsigned char myArray[45] = {
    0xDD, 0x5B, 0x9E, 0x1D, 0x20, 0x9E, 0x90, 0x91, 0x90, 0x90,
    0x91, 0x92, 0xDE, 0x8B, 0x11, 0xD1, 0x1E, 0x9E, 0x8B, 0x51,
    0x11, 0x50, 0x51, 0x8B, 0x9E, 0x5D, 0x5D, 0x11, 0x8B, 0x90,
```

```
    0x12, 0x91, 0x50, 0x12, 0xD2, 0x91, 0x92, 0x1E, 0x9E, 0x90,
    0xD2, 0x9F, 0x00, 0x00
};

int main()
{
  char v5; // [esp+0h] [ebp-14h]
  HANDLE Handles[2]; // [esp+8h] [ebp-Ch] BYREF
  unsigned char Str[45]={0};
  cout<<"plz input your flag:";
  cin>>Str;
  Handles[0] = CreateThread(0, 0, StartAddress, 0, 0, 0);
  Handles[1] = CreateThread(0, 0, loc_401200, 0, 0, 0);
  CreateThread(0, 0, sub_401240, 0, 0, 0);
  WaitForMultipleObjects(2u, Handles, 1, 0xFFFFFFFF);
  for (int i = 0; i < 42; ++i )
  {
    if ( Str[i] != myArray[i] )
    {
      sub_401020("error", (char)Handles[0]);
      exit(0);
    }
  }
  sub_401020("win", (char)Handles[0]);
  getchar();
  return 0;
}
```

这图主程序大致如上，先输入str，再召唤三个线程，等这三个线程完毕后，比对str和myArra。若两者相等则为flag。所以我们主要看三个线程的开始函数。第三个线程的函数调用了debug的东西所以和主程序无关，第二个线程的函数完全破坏了str的结构，所以我推测可能也是调试或特殊情况触发的线程。所以**先主要关注第一个线程。**

第一个线程使用了一些junkcode，除去这些junkcode后我们发现了如下代码。

```
mov     ecx, [ebp-4]
movzx   edx, byte_40336C[ecx]
sar     edx, 2
mov     eax, [ebp-4]
movzx   ecx, byte_40336C[eax]
shl     ecx, 6
xor     edx, ecx
mov     eax, [ebp-4]
mov     byte_40336C[eax], dl
mov     ecx, [ebp-4]
movzx   edx, byte_40336C[ecx]
xor     edx, 23h
mov     eax, [ebp-4]
mov     byte_40336C[eax], dl
...

movzx   edx, byte_40336C[ecx]
add     edx, 23h ; '#'
mov     eax, [ebp-4]
mov     byte_40336C[eax], dl
```

可以看出，第一个线程对str[i]进行了如下操作：

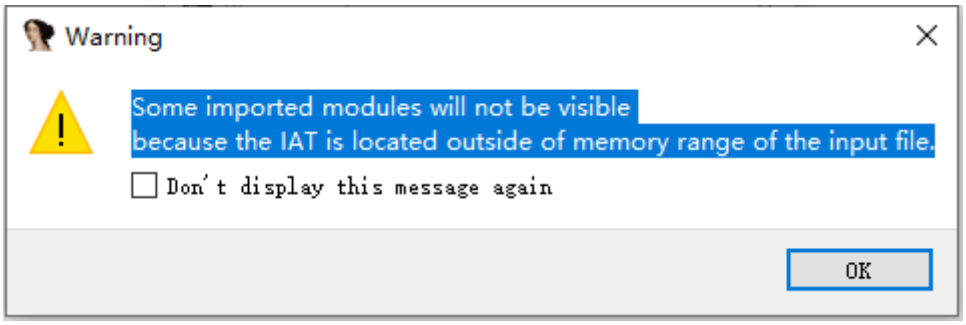
str[i] = (((str[i]>>2) ^ (str[i]<<6)) ^ 0x23) + 0x23

根据上述等式反推即可获得**flag{a959951b-76ca-4784-add7-93583251ca92}**，经验证flag正确

10-babyvm

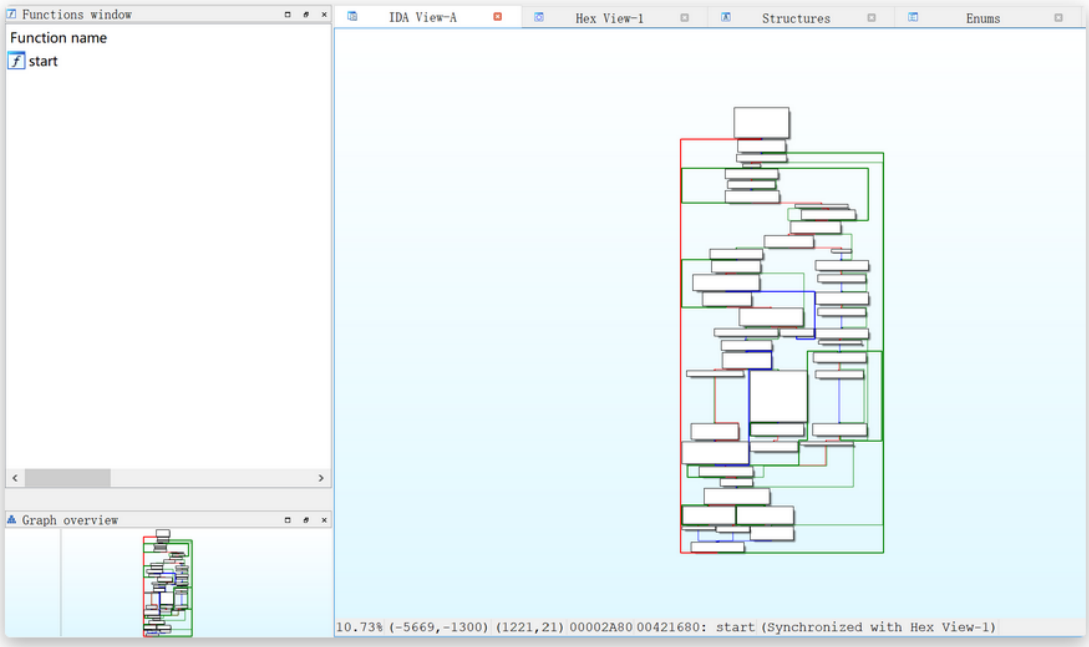
~~这题感觉跟click一样主要靠观察~~

这题一放进IDA就弹出了如下warning，并且反汇编后的代码甚至没有main函数。疑点重重。



于是我直接把warning的内容搜了一搜，找到了一篇博客：

使用 IDA 打开时会提示 `Some imported modules will not be visible because the IAT is located outside of memory range of the input file.` ，忽略后的结果如图：



加壳后的IDA页面

这样的代码完全无法阅读，字符串、函数信息等都已经没了。

于是我们得知**这题的程序可能是被UPX加壳了**，这篇博客下面介绍了如何使用x64dbg脱壳，但是过于复杂。于是我又搜了搜发现可以直接用UPX脱壳，于是在开心地脱壳后我们得到了正确的反汇编代码并将其转为C。

```
char dword_403000[24] = {'~','x','u',0x7F,'k','R','u','r','m','w','N','y',
                         'y','y','w','D','b','$','`','q','s','`','5','i'};
int sub_401000(char * a1)
{
  int result; // eax
  int v2; // [esp+8h] [ebp-4h]
  unsigned char dword_403060[10]={0xCC,0xA,0xB,0xC,0xD,0xE,0xF,0xAA,0xBB,0xFF};
  int dword_403104[5];
  dword_403104[0] = 0;

  if ( dword_403060[dword_403104[0]] == 204 )
    dword_403104[1] = 0;
  ++dword_403104[0];
  do
  {
    result = dword_403060[dword_403104[0]];
    v2 = result;
    if ( result == 221 )
    {
      ++dword_403104[2];
    }
    else if ( result <= 221 )
    {
      if ( result == 187 )
      {
        if ( !dword_403104[4] )
```

```
          dword_403104[0] = 0;
      }
      else if ( result <= 187 )
      {
        if ( result <= 15 )
        {
          if ( result >= 10 )
          {
            result = dword_403060[dword_403104[0]];
            switch ( v2 )
            {
              case 10:
                dword_403104[2] = a1[dword_403104[1]];
                break;
              case 11:
                dword_403104[3] = dword_403104[2];
                break;
              case 12:
                result = (dword_403104[3] + 2) ^ 0x16;
                dword_403104[3] = result;
                break;
              case 13:
                dword_403104[2] = dword_403104[3];
                break;
              case 14:
                result = dword_403104[2];
                a1[dword_403104[1]] = dword_403104[2];
                break;
              case 15:
                ++dword_403104[1];
                break;
              default:
                break;
            }
          }
        }
        else if ( result == 170 )
        {
          dword_403104[4] = dword_403104[1] == 24;
        }
      }
    }
    ++dword_403104[1];
  }
  while ( dword_403104[1] <= 8 );
  return result;
}

int main()
{
  char Str[30]={0};
  puts("please input your flag:");
  cin>>Str;
  if ( strlen(Str) == 24 )
  {
```

```
    sub_401000(Str);
    for (int i = 0; i < 24; ++i )
    {
      if ( dword_403000[i] != Str[i] )
      {
        return 0;
      }
    }
    puts("success!");
  }
  return 0;
}
```

还是老一套，输入，加密，匹配。看加密函数sub_401000()，一眼看上去很复杂，但是跟着函数跑一趟后发现发现整个函数就是**对每个str[i], 让str[i] = (str[i] + 2) ^ 0x16**,鉴于本题叫babyvm，加密函数sub_401000()可能是通过模拟了虚拟机的寄存器来实现这样一个简单的操作。不过这和我们解题没什么关系。我们只需要对着这个加密来解密即可。得到答案**flag{Baby_Vmmm_Pr0tect!}**