DAS12-刻板印象re的解题思路

题目信息

题目名	类型	难度
DAS12-刻板印象re	逆向	困难

FLAG

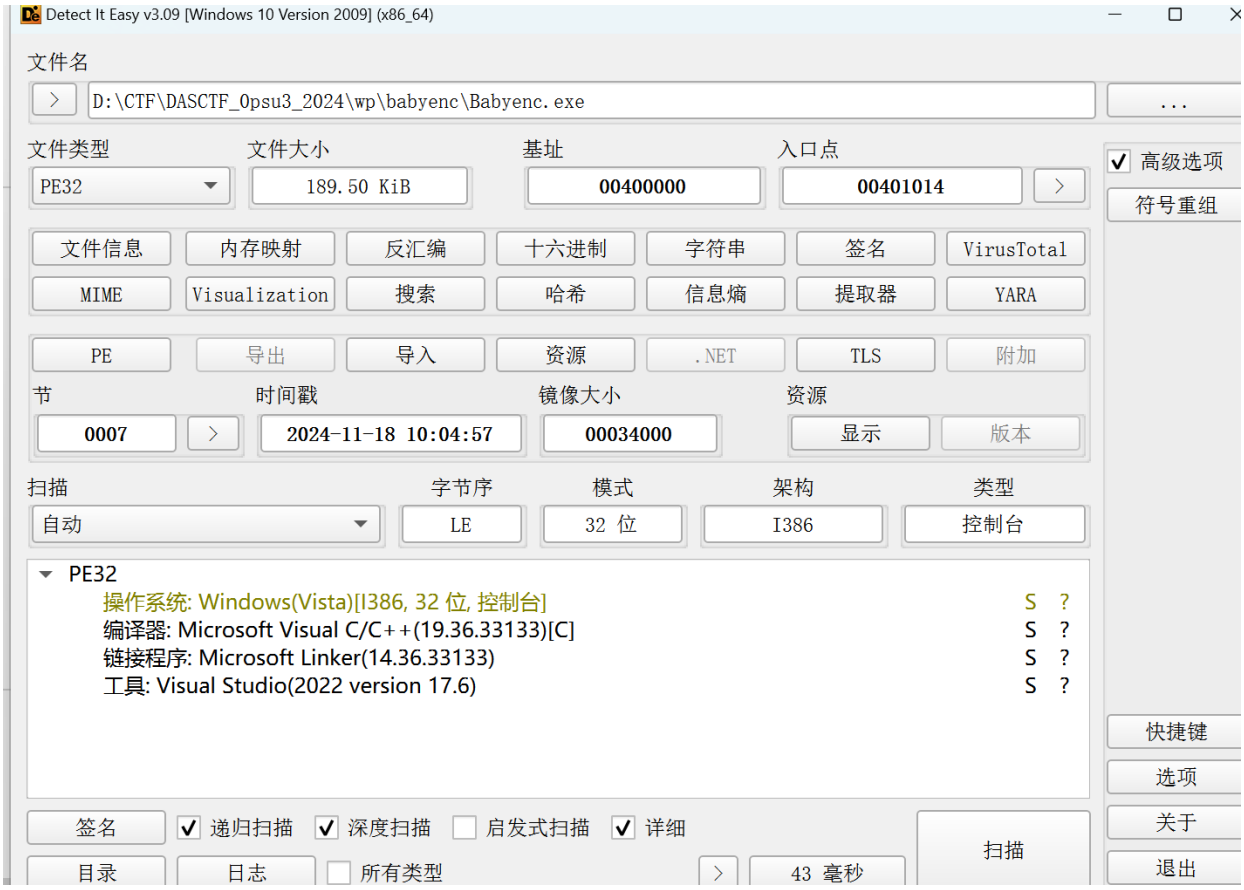
- DASCTF{You_come_to_me_better_than_all_the_good.}

知识点

1. 静态分析
2. 动态调试
3. 简单花指令
4. xtea，xxtea算法
5. inline hook
6. trace与指令处理

解题步骤

先用die查看，32位PE，无壳



主函数逻辑非常简单，就是检查flag长度，然后进入加密函数，最后检验

```
1  int main()
2  {
3    unsigned __int8 input[49]; // [esp+8h] [ebp-3Ch] BYREF
4
5    j__printf("Welcome to DASCTF! \n");
6    j__scanf("%48s", input);
7    if ( strlen((const char *)input) == 48 )
8    {
9      j__enc(input);
10     if ( !memcmp(input, cipher, 0x30u) )
11       j__printf("Right! \n");
12     else
13       j__printf("Wrong! \n");
14     return 0;
15   }
16   else
17   {
18     j__printf("Length wrong! \n");
19     return 0;
20   }
21 }
```

进入加密函数，只有一个xor

```
// positive sp value has been detected, the output may be wrong!
void __cdecl enc(unsigned __int8 *input)
{
  int i; // [esp+58h] [ebp-4h]

  for ( i = 0; i < 48; ++i )
    input[i] ^= key1[i % 26];
}
```

此时直接解密会得到一个fake flag

```
#include<iostream>
using namespace std;
unsigned char xor_key[] = "Laughter_is_poison_to_fear";
unsigned char cipher[] =
{
  0x18, 0x09, 0x1C, 0x14, 0x37, 0x1D, 0x16, 0x2D, 0x3C, 0x05,
  0x16, 0x3E, 0x02, 0x03, 0x10, 0x2C, 0x0E, 0x31, 0x39, 0x15,
  0x04, 0x3A, 0x39, 0x03, 0x0D, 0x13, 0x2B, 0x3E, 0x06, 0x08,
  0x37, 0x00, 0x17, 0x0B, 0x00, 0x1D, 0x1C, 0x00, 0x16, 0x06,
  0x07, 0x17, 0x30, 0x03, 0x30, 0x06, 0x0A, 0x71
};
int main()
{
    unsigned char a;
    for (int i = 0; i < 48; i++) {
        a = cipher[i] ^ xor_key[i % 26];
        printf("%c", a);
    }
}
```

```
Microsoft Visual Studio 调试控
This_is_clearly_a_fake_flag_so_try_to_find_more.
```

查看汇编层，发现是有一个花指令

```
.text:00401780                 mov     eax, [ebp+input]
.text:00401783                 add     eax, [ebp+i]
.text:00401786                 movzx   ecx, byte ptr [eax]
.text:00401789                 xor     ecx, edx
.text:0040178B                 mov     edx, [ebp+input]
.text:0040178E                 add     edx, [ebp+i]
.text:00401791                 mov     [edx], cl
.text:00401793                 jmp     short loc_40175F
.text:00401795 ; ---------------------------------------------------------------------------
.text:00401795
.text:00401795 loc_401795:                             ; CODE XREF: _enc+2C↑j
.text:00401795                 call    $+5
.text:0040179A
.text:0040179A label0:
.text:0040179A                 db      36h
.text:0040179A                 add     [esp+5Ch+var_5C], 6
.text:0040179F                 retn
.text:004017A0 ; ---------------------------------------------------------------------------
.text:004017A0                 mov     eax, [ebp+input]
.text:004017A3                 mov     [ebp+v], eax
.text:004017A6                 mov     ecx, dword_42F0AC
.text:004017AC                 mov     dword ptr [ebp+key2], ecx
.text:004017AF                 mov     edx, dword_42F0B0
.text:004017B5                 mov     dword ptr [ebp+key2+4], edx
.text:004017B8                 mov     eax, dword_42F0B4
.text:004017BD                 mov     dword ptr [ebp+key2+8], eax
```

去除

```
.text:00401772 B9 1A 00 00 00                   mov     ecx, 1Ah
.text:00401777 F7 F9                            idiv    ecx
.text:00401779 0F B6 92 90 F0 42 00             movzx   edx, _key1[edx]
.text:00401780 8B 45 08                         mov     eax, [ebp+input]
.text:00401783 03 45 FC                         add     eax, [ebp+i]
.text:00401786 0F B6 08                         movzx   ecx, byte ptr [eax]
.text:00401789 33 CA                            xor     ecx, edx
.text:0040178B 8B 55 08                         mov     edx, [ebp+input]
.text:0040178E 03 55 FC                         add     edx, [ebp+i]
.text:00401791 88 0A                            mov     [edx], cl
.text:00401793 EB CA                            jmp     short loc_40175F
.text:00401795                          ; ---------------------------------------------------------------------------
.text:00401795
.text:00401795                          loc_401795:                             ; CODE XREF: _enc+2C↑j
.text:00401795 90                               nop
.text:00401796 90                               nop
.text:00401797 90                               nop
.text:00401798 90                               nop
.text:00401799 90                               nop
.text:0040179A
.text:0040179A                          label0:
.text:0040179A 90                               nop
.text:0040179B 90                               nop
.text:0040179C 90                               nop
.text:0040179D 90                               nop
.text:0040179E 90                               nop
.text:0040179F 90                               nop
.text:004017A0 8B 45 08                         mov     eax, [ebp+input]
.text:004017A3 89 45 F8                         mov     [ebp+v], eax
.text:004017A6 8B 0D AC F0 42 00                mov     ecx, dword_42F0AC
.text:004017AC 89 4D E0                         mov     dword ptr [ebp+key2], ecx
.text:004017AF 8B 15 B0 F0 42 00                mov     edx, dword_42F0B0
.text:004017B5 89 55 E4                         mov     dword ptr [ebp+key2+4], edx
.text:004017B8 A1 B4 F0 42 00                   mov     eax, dword_42F0B4
.text:004017BD 89 45 E8                         mov     dword ptr [ebp+key2+8], eax
.text:004017C0 8B 0D B8 F0 42 00                mov     ecx, dword_42F0B8
```

然后就可以正常识别了

```
1  void __cdecl enc(unsigned __int8 *input)
2  {
3    int k; // [esp+8h] [ebp-54h]
4    unsigned int j; // [esp+20h] [ebp-3Ch]
5    unsigned int sum; // [esp+24h] [ebp-38h]
6    unsigned int v1; // [esp+28h] [ebp-34h]
7    unsigned int v0; // [esp+2Ch] [ebp-30h]
8    int l; // [esp+30h] [ebp-2Ch]
9    unsigned __int8 key2[17]; // [esp+3Ch] [ebp-20h] BYREF
10   unsigned int *v; // [esp+54h] [ebp-8h]
11   int i; // [esp+58h] [ebp-4h]
12
13   for ( i = 0; i < 48; ++i )
14     input[i] ^= key1[i % 26];
15   v = input;
16   strcpy(key2, "{you_find_it_!?}");
17   for ( l = 0; l < 12; l += 2 )
18   {
19     v0 = v[l];
20     v1 = v[l + 1];
21     sum = 0;
22     for ( j = 0; j < 0x20; ++j )
23     {
24       v0 += (sum + *&key2[4 * (sum & 3)]) ^ (v1 + ((16 * v1) ^ (v1 >> 5)));
25       sum -= 1640531527;
26       v1 += (sum + *&key2[4 * ((sum >> 11) & 3)]) ^ (v0 + ((16 * v0) ^ (v0 >> 5)));
27     }
28     v[l] = v0;
29     v[l + 1] = v1;
30   }
31   for ( k = 0; k < 48; ++k )
32     input[k] ^= fake_xor_key[k];
33 }
```

可以看到就是一个tea和xor，但解出来仍然是一个fake flag

```
28      unsigned int* v = (unsigned int*)cipher;
29      unsigned char key2[] = "{you_find_it_!?}";
30      unsigned int* k = (unsigned int*)key2;
31      for (int l = 0; l < 12; l += 2) {
32
33
34
35
36          unsigned int  v0 = v[l], v1 = v[l + 1], sum = 0, i;     //v0,v1分别为字符串的低字节高字节
37          //printf("%x %x \n", v0, v1);
38          unsigned int delta = 0x61C88647;
39          for (int i = 0; i < 32; i++) {
40              sum -= delta;
41          }
42
43
44          int k0 = k[0], k1 = k[1], k2 = k[2], k3 = k[3];
45          for (i = 0; i < 32; i++) {                  //加密32轮
46              v1 -= (((v0 >> 5) ^ (16 * v0)) + v0) ^ (k[(sum >> 11) & 3] + sum);
47              sum += delta;
48              v0 -= (((v1 >> 5) ^ (16 * v1)) + v1) ^ (k[sum & 3] + sum);
49          }
50          v[l] = v0; v[l + 1] = v1;//加密后再重新赋值
51      }
52
53      for (int i = 0; i < 48; i++) {
54          cipher[i] ^= xor_key[i%26];
55      }
56
57      for (int i = 0; i < 48; i++) {
58          printf("%c", cipher[i]);
```

Microsoft Visual Studio 调试控

```
fakeflag_plz_Try_more_hard_to_find_the_true_flag
```

接下来开始动调查看

```
 8  int i; // [esp+30h] [ebp-2Ch]
 9  unsigned __int8 key2[17]; // [esp+3Ch] [ebp-20h] BYREF
10  unsigned int *v; // [esp+54h] [ebp-8h]
11  int i; // [esp+58h] [ebp-4h]
12
13  for ( i = 0; i < 48; ++i )
14    input[i] ^= key1[i % 26];
15  v = input;
16  strcpy(key2, "{you_find_it_!?}");
17  for ( l = 0; l < 12; l += 2 )
18  {
19    v0 = v[l];
20    v1 = v[l + 1];
21    sum = 0;
22    for ( j = 0; j < 0x20; ++j )
23    {
24      v0 += (sum + *&key2[4 * (sum & 3)]) ^ (v1 + ((16 * v1) ^ (v1 >> 5)));
25      sum -= 1640531527;
26      v1 += (sum + *&key2[4 * ((sum >> 11) & 3)]) ^ (v0 + ((16 * v0) ^ (v0 >> 5)));
27    }
28    v[l] = v0;
29    v[l + 1] = v1;
30  }
31  for ( k = 0; k < 48; ++k )
32    input[k] ^= fake_xor_key[k];
33 }
```

发现在enc函数结束后，还进入了一个函数

IDA View-EIP

```
1  // local variable allocation has failed, the output may be wrong!
2  void __cdecl hhh(unsigned __int8 *input)
3  {
4    _BYTE BTEA[10]; // [esp+4h] [ebp-14h] OVERLAPPED BYREF
5    unsigned __int8 *btea2; // [esp+14h] [ebp-4h]
6
7    memset(BTEA, 204, sizeof(BTEA));
8    btea2 = _malloc(0x4359u);
9    j__init(btea2);
10   j__unHook();
11   j__enc(input);
12   VirtualProtect(btea2, 0x4358u, 0x40u, &BTEA[8]);
13   *BTEA = btea2 + 1;
14   ((btea2 + 1))(input);
15 }
```

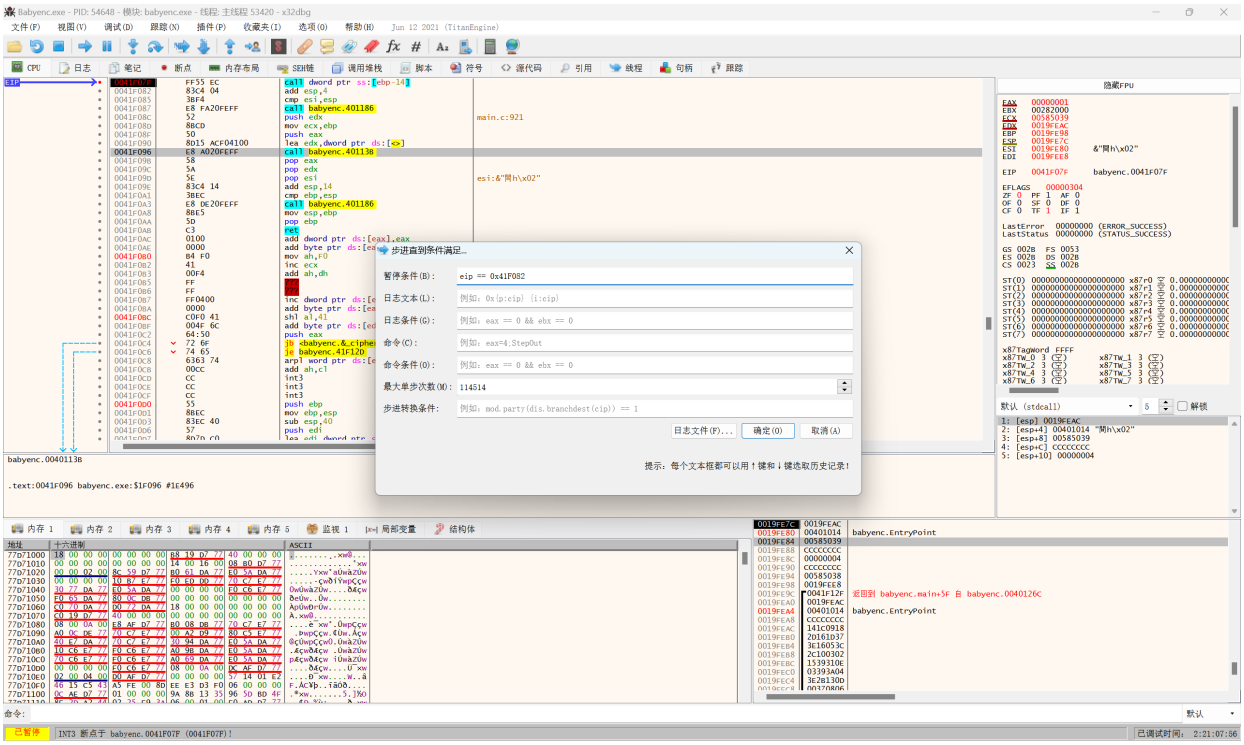
进入那段地址，可以看到一长串汇编代码

```
debug042:0075ECB7 db 0FDh
debug042:0075ECB8 db    0
debug042:0075ECB9 ; ---------------------------------------------------------
debug042:0075ECB9 mov     ecx, 77h ; 'w'
debug042:0075ECBE call    $+5
EIP
debug042:0075ECC3 pop     ebx
debug042:0075ECC4 mov     [ebx-0Bh], cl
debug042:0075ECC7 push    ebp
debug042:0075ECC8 pusha
debug042:0075ECC9 pushf
debug042:0075ECCA call    $+5
debug042:0075ECCF pop     eax
debug042:0075ECD0 xor     edx, edx
debug042:0075ECD2 mov     dl, [eax-17h]
debug042:0075ECD8 xor     edx, 1D82h
debug042:0075ECDE lea     ebx, [eax+edx]
debug042:0075ECE1 mov     ecx, 35h ; '5'
debug042:0075ECE6 mov     byte ptr [eax-17h], 0
debug042:0075ECED jmp     ebx
debug042:0075ECED ; ---------------------------------------------------------
debug042:0075ECEF db    0
debug042:0075ECF0 db 0E8h
```

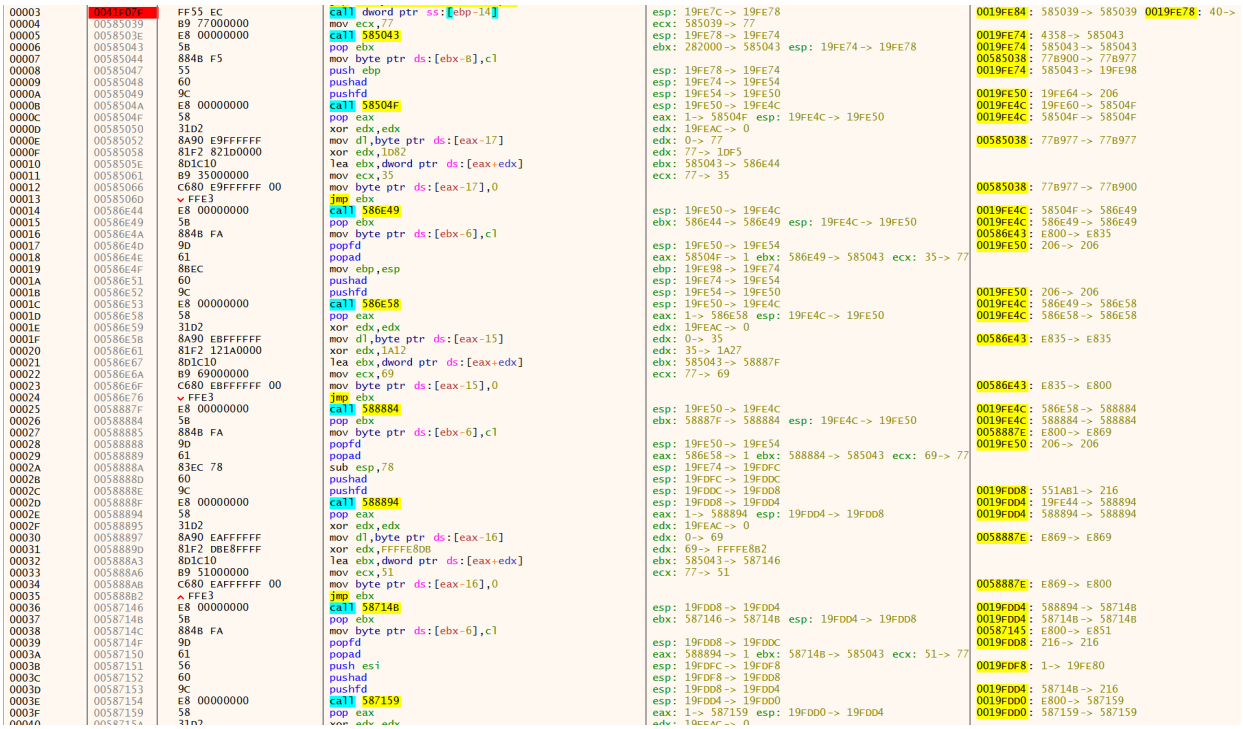
而核心逻辑就藏在每个popf popa和pusha，pushf之间

```
debug042:00760AC3 db    0
debug042:00760AC4 ; ---------------------------------------------------------
EBX
debug042:00760AC4 call    $+5
EIP
debug042:00760AC9 pop     ebx
debug042:00760ACA mov     [ebx-6], cl
debug042:00760ACD popf
debug042:00760ACE popa
debug042:00760ACF mov     ebp, esp
debug042:00760AD1 pusha
debug042:00760AD2 pushf
debug042:00760AD3 call    $+5
debug042:00760AD8 pop     eax
debug042:00760AD9 xor     edx, edx
debug042:00760ADB mov     dl, [eax-15h]
debug042:00760AE1 xor     edx, 1A12h
debug042:00760AE7 lea     ebx, [eax+edx]
debug042:00760AEA mov     ecx, 69h ; 'i'
debug042:00760AEF mov     byte ptr [eax-15h], 0
debug042:00760AF6 jmp     ebx
debug042:00760AF6 ; ---------------------------------------------------------
debug042:00760AF8 db    0
```

然而每个指令块之间的跳转经过了xor解密，而key来自于上个块，不好静态解出，故使用x32dbg trace查看



trace完毕，将记录保存下来并用python脚本处理



```
# 读取文件并提取每列的数据

class CODE:
    def __init__(self, index, eip, asm, dis):
        self.index = index
        self.eip = eip
        self.asm = asm
        self.dis = dis
def extract_columns_from_file(file_path):
    # 打开文件并读取所有行
    with open(file_path, 'r', encoding='utf-8') as file:
        lines = file.readlines()

    codes = []
    # 处理每一行，提取每一列的数据
    for i in range(len(lines)):
        line = lines[i]
        # 去除行首尾的空白符，并按'|'分割
        parts = line.strip().split('|')

        # 确保行包含至少7个部分，防止异常
        if len(parts) >= 6:
            #column1.append(parts[0].strip())
            eip = int(parts[1].strip(), 16)
            asm = parts[2].strip().replace(" ", "")
            dis = parts[3].strip()
            code = CODE(i, eip, asm, dis)
            codes.append(code)

    # 返回每列的数据作为结果
```

```
    return codes


# 使用示例
file_path = '1.txt'  # 替换为你的txt文件路径
codes = extract_columns_from_file(file_path)



final_codes = []
final_codes.append(codes[0x5])
for i in range(0x16, len(codes) - 1):

    insert1 = 0
    insert2 = 0
    code = codes[i]

    #普通的指令
    if(codes[i - 1].dis == "popad" and codes[i + 1].dis == "pushad"):
        insert1 = 1
        #print(f"{code.index} {(code.eip):x} {code.asm} {code.dis}")
        for k in range(len(final_codes)):
            if(final_codes[k].eip == code.eip):
                insert1 = 0
                break

    #跳转指令
    if (codes[i - 2].dis == "pushfd" and codes[i + 1].dis == "popfd"):
        insert2 = 1
        # print(f"{code.index} {(code.eip):x} {code.asm} {code.dis}")
        for k in range(len(final_codes)):
            if (final_codes[k].eip == code.eip):
                insert2 = 0
                break

    if(insert1 == 1 or insert2 == 1):
        final_codes.append(code)

final_codes.append(codes[len(codes)-  1])

for i in range(len(final_codes)):
    code = final_codes[i]
    print(f"{(code.eip):X} {code.asm} {code.dis}")
```

最终得到以下代码

```
754047 55 push ebp
755E4F 8BEC mov ebp,esp
75788A 83EC78 sub esp,78
```

```
756151 56 push esi
754OB2 57 push edi
7543B4 C745EC0C000000 mov dword ptr ss:[ebp-14],C
755FFE 8B4508 mov eax,dword ptr ss:[ebp+8]
7548A2 8945F0 mov dword ptr ss:[ebp-10],eax
7580E0 C645B87B mov byte ptr ss:[ebp-48],7B
757CCF C645B957 mov byte ptr ss:[ebp-47],57
755658 C645BA68 mov byte ptr ss:[ebp-46],68
75781E C645BB61 mov byte ptr ss:[ebp-45],61
75662F C645BC74 mov byte ptr ss:[ebp-44],74
755A77 C645BD5F mov byte ptr ss:[ebp-43],5F
756C21 C645BE69 mov byte ptr ss:[ebp-42],69
75426B C645BF73 mov byte ptr ss:[ebp-41],73
7543EE C645C05F mov byte ptr ss:[ebp-40],5F
755C2A C645C174 mov byte ptr ss:[ebp-3F],74
754152 C645C268 mov byte ptr ss:[ebp-3E],68
756ADE C645C369 mov byte ptr ss:[ebp-3D],69
757300 C645C473 mov byte ptr ss:[ebp-3C],73
7564EB C645C55F mov byte ptr ss:[ebp-3B],5F
754FE8 C645C63F mov byte ptr ss:[ebp-3A],3F
755B51 C645C77D mov byte ptr ss:[ebp-39],7D
754B97 8D4DB8 lea ecx,dword ptr ss:[ebp-48]
7563DC 894DD8 mov dword ptr ss:[ebp-28],ecx
7568F7 837DEC01 cmp dword ptr ss:[ebp-14],1
75505D 0F8EC7F4FFFF jle 75452A
756D6B B834000000 mov eax,34
75699B 99 cdq
756855 F77DEC idiv dword ptr ss:[ebp-14]
758074 83C006 add eax,6
7542A2 8945E4 mov dword ptr ss:[ebp-1C],eax
7579D2 C745E800000000 mov dword ptr ss:[ebp-18],0
7577E8 8B55EC mov edx,dword ptr ss:[ebp-14]
755D41 8B45F0 mov eax,dword ptr ss:[ebp-10]
757C2B 8B4C90FC mov ecx,dword ptr ds:[eax+edx*4-4]
7567E9 894DF8 mov dword ptr ss:[ebp-8],ecx
7570AD 8B55E8 mov edx,dword ptr ss:[ebp-18]
756C8E 81C219144511 add edx,11451419
756BEB 8955E8 mov dword ptr ss:[ebp-18],edx
757595 8B45E8 mov eax,dword ptr ss:[ebp-18]
755BF4 C1E802 shr eax,2
7568C1 83E003 and eax,3
755104 8945DC mov dword ptr ss:[ebp-24],eax
757E56 C745FC00000000 mov dword ptr ss:[ebp-4],0
755212 8B55EC mov edx,dword ptr ss:[ebp-14]
7576D9 83EA01 sub edx,1
7557A9 3955FC cmp dword ptr ss:[ebp-4],edx
757DE5 0F8332E4FFFF jae 75621D
758331 8B45FC mov eax,dword ptr ss:[ebp-4]
754425 8B4DF0 mov ecx,dword ptr ss:[ebp-10]
755D77 8B548104 mov edx,dword ptr ds:[ecx+eax*4+4]
754E61 8955F4 mov dword ptr ss:[ebp-C],edx
75752A 8B45F8 mov eax,dword ptr ss:[ebp-8]
```

```
7571BC C1E805 shr eax,5
756DA3 8B4DF4 mov ecx,dword ptr ss:[ebp-C]
758007 C1E102 shl ecx,2
757855 33C1 xor eax,ecx
7561F2 8B55F4 mov edx,dword ptr ss:[ebp-C]
7556C5 C1EA03 shr edx,3
755F5C 8B4DF8 mov ecx,dword ptr ss:[ebp-8]
755926 C1E104 shl ecx,4
755248 33D1 xor edx,ecx
754200 03C2 add eax,edx
7550CE 8B55E8 mov edx,dword ptr ss:[ebp-18]
758181 3355F4 xor edx,dword ptr ss:[ebp-C]
754AC0 8B4DFC mov ecx,dword ptr ss:[ebp-4]
7551DC 83E103 and ecx,3
75584D 334DDC xor ecx,dword ptr ss:[ebp-24]
756965 8B75D8 mov esi,dword ptr ss:[ebp-28]
7572CA 8B0C8E mov ecx,dword ptr ds:[esi+ecx*4]
757C99 334DF8 xor ecx,dword ptr ss:[ebp-8]
758117 03D1 add edx,ecx
755883 33C2 xor eax,edx
75711A 8B55FC mov edx,dword ptr ss:[ebp-4]
7540E6 8B4DF0 mov ecx,dword ptr ss:[ebp-10]
754D54 8B1491 mov edx,dword ptr ds:[ecx+edx*4]
7582C7 03D0 add edx,eax
75647F 8955D4 mov dword ptr ss:[ebp-2C],edx
7555EE 8B45FC mov eax,dword ptr ss:[ebp-4]
75677E 8B4DF0 mov ecx,dword ptr ss:[ebp-10]
75681F 8B55D4 mov edx,dword ptr ss:[ebp-2C]
755EB9 891481 mov dword ptr ds:[ecx+eax*4],edx
756302 8B45D4 mov eax,dword ptr ss:[ebp-2C]
756B80 8945F8 mov dword ptr ss:[ebp-8],eax
7561BC 8B4DFC mov ecx,dword ptr ss:[ebp-4]
756E45 83C101 add ecx,1
757637 894DFC mov dword ptr ss:[ebp-4],ecx
756228 B9040000000 mov ecx,4
75538C 6BD100 imul edx,ecx,0
754A8A 8B45F0 mov eax,dword ptr ss:[ebp-10]
755B1B 8B0C10 mov ecx,dword ptr ds:[eax+edx]
7545D9 894DF4 mov dword ptr ss:[ebp-C],ecx
75636F 8B55F8 mov edx,dword ptr ss:[ebp-8]
7577B2 C1EA05 shr edx,5
75658D 8B45F4 mov eax,dword ptr ss:[ebp-C]
757964 C1E002 shl eax,2
756558 33D0 xor edx,eax
755515 8B4DF4 mov ecx,dword ptr ss:[ebp-C]
755A0A C1E903 shr ecx,3
7564B5 8B45F8 mov eax,dword ptr ss:[ebp-8]
757228 C1E004 shl eax,4
755171 33C8 xor ecx,eax
7582FC 03D1 add edx,ecx
756F89 8B4DE8 mov ecx,dword ptr ss:[ebp-18]
757AAF 334DF4 xor ecx,dword ptr ss:[ebp-C]
```

```
754ECD 8B45FC mov eax,dword ptr ss:[ebp-4]
755356 83E003 and eax,3
7573DA 3345DC xor eax,dword ptr ss:[ebp-24]
755F26 8B75D8 mov esi,dword ptr ss:[ebp-28]
756412 8B0486 mov eax,dword ptr ds:[esi+eax*4]
7562CC 3345F8 xor eax,dword ptr ss:[ebp-8]
755E1A 03C8 add ecx,eax
7559D5 33D1 xor edx,ecx
757A0C 8B4DEC mov ecx,dword ptr ss:[ebp-14]
755F92 8B45F0 mov eax,dword ptr ss:[ebp-10]
7570E3 8B4C88FC mov ecx,dword ptr ds:[eax+ecx*4-4]
758367 03CA add ecx,edx
75527D 894DD0 mov dword ptr ss:[ebp-30],ecx
757601 8B55EC mov edx,dword ptr ss:[ebp-14]
754A1E 8B45F0 mov eax,dword ptr ss:[ebp-10]
757F9B 8B4DD0 mov ecx,dword ptr ss:[ebp-30]
75430F 894C90FC mov dword ptr ds:[eax+edx*4-4],ecx
755FC8 8B55D0 mov edx,dword ptr ss:[ebp-30]
757D3C 8955F8 mov dword ptr ss:[ebp-8],edx
755BBE 8B45E4 mov eax,dword ptr ss:[ebp-1C]
75411C 83E801 sub eax,1
756034 8945E4 mov dword ptr ss:[ebp-1C],eax
7554DA 0F85C21B0000 jne 7570A2
7563A5 C645888F mov byte ptr ss:[ebp-78],8F
75736D C645896C mov byte ptr ss:[ebp-77],6C
755EEF C6458AA6 mov byte ptr ss:[ebp-76],A6
75513A C6458B3F mov byte ptr ss:[ebp-75],3F
757A42 C6458C94 mov byte ptr ss:[ebp-74],94
75542E C6458D3D mov byte ptr ss:[ebp-73],3D
75467B C6458EF5 mov byte ptr ss:[ebp-72],F5
7558B8 C6458FD9 mov byte ptr ss:[ebp-71],D9
75490F C6459036 mov byte ptr ss:[ebp-70],36
754D1D C6459166 mov byte ptr ss:[ebp-6F],66
754F3A C6459251 mov byte ptr ss:[ebp-6E],51
754346 C64593D7 mov byte ptr ss:[ebp-6D],D7
755AE4 C6459466 mov byte ptr ss:[ebp-6C],66
755581 C645952F mov byte ptr ss:[ebp-6B],2F
755A40 C64596B3 mov byte ptr ss:[ebp-6A],B3
754C38 C645978F mov byte ptr ss:[ebp-69],8F
757446 C64598C0 mov byte ptr ss:[ebp-68],C0
75692E C6459961 mov byte ptr ss:[ebp-67],61
75611A C6459A9E mov byte ptr ss:[ebp-66],9E
7581B7 C6459BCE mov byte ptr ss:[ebp-65],CE
755772 C6459CE9 mov byte ptr ss:[ebp-64],E9
75437D C6459DD7 mov byte ptr ss:[ebp-63],D7
75825A C6459EE1 mov byte ptr ss:[ebp-62],E1
7552E9 C6459FBF mov byte ptr ss:[ebp-61],BF
756CC7 C645A013 mov byte ptr ss:[ebp-60],13
754F03 C645A114 mov byte ptr ss:[ebp-5F],14
75445B C645A216 mov byte ptr ss:[ebp-5E],16
7580D3 C645A314 mov byte ptr ss:[ebp-5D],14
75456C C645A4C2 mov byte ptr ss:[ebp-5C],C2
```

```
7578C0 C645A5E7 mov byte ptr ss:[ebp-5B],E7
7542D8 C645A6C3 mov byte ptr ss:[ebp-5A],C3
7548D8 C645A73A mov byte ptr ss:[ebp-59],3A
75770F C645A87F mov byte ptr ss:[ebp-58],7F
75606A C645A994 mov byte ptr ss:[ebp-57],94
754492 C645AAA1 mov byte ptr ss:[ebp-56],A1
75549C C645ABE7 mov byte ptr ss:[ebp-55],E7
757B50 C645AC24 mov byte ptr ss:[ebp-54],24
756185 C645AD0E mov byte ptr ss:[ebp-53],E
757C62 C645AEA7 mov byte ptr ss:[ebp-52],A7
75792D C645AF5C mov byte ptr ss:[ebp-51],5C
75501F C645B0D3 mov byte ptr ss:[ebp-50],D3
756EB1 C645B177 mov byte ptr ss:[ebp-4F],77
756338 C645B2FE mov byte ptr ss:[ebp-4E],FE
7558EF C645B34F mov byte ptr ss:[ebp-4D],4F
75407B C645B411 mov byte ptr ss:[ebp-4C],11
755465 C645B5DC mov byte ptr ss:[ebp-4B],DC
756448 C645B669 mov byte ptr ss:[ebp-4A],69
755816 C645B723 mov byte ptr ss:[ebp-49],23
755D07 C745E0000000 mov dword ptr ss:[ebp-20],0
7560A1 837DE030 cmp dword ptr ss:[ebp-20],30
754867 0F8D4A0B0000 jge 7553B7
756522 8B45E0 mov eax,dword ptr ss:[ebp-20]
7553F6 0FB64C0588 movzx ecx,byte ptr ss:[ebp+eax-78]
756748 8B5508 mov edx,dword ptr ss:[ebp+8]
756DD9 0355E0 add edx,dword ptr ss:[ebp-20]
756296 0FB602 movzx eax,byte ptr ds:[edx]
756BB6 33C1 xor eax,ecx
755320 8B4D08 mov ecx,dword ptr ss:[ebp+8]
757FD1 034DE0 add ecx,dword ptr ss:[ebp-20]
757560 8801 mov byte ptr ds:[ecx],al
758291 8B55E0 mov edx,dword ptr ss:[ebp-20]
757186 83C201 add edx,1
756C58 8955E0 mov dword ptr ss:[ebp-20],edx
7553C2 5F pop edi
757EC6 5E pop esi
754BCD 8BE5 mov esp,ebp
7546E8 5D pop ebp
754752 C3 ret
```

可以看出就是一个xxtea，并且最后有一个xor

因此加密流程就是xor key1 -> tea -> xor key2 -> xxtea -> xor key3

exp:

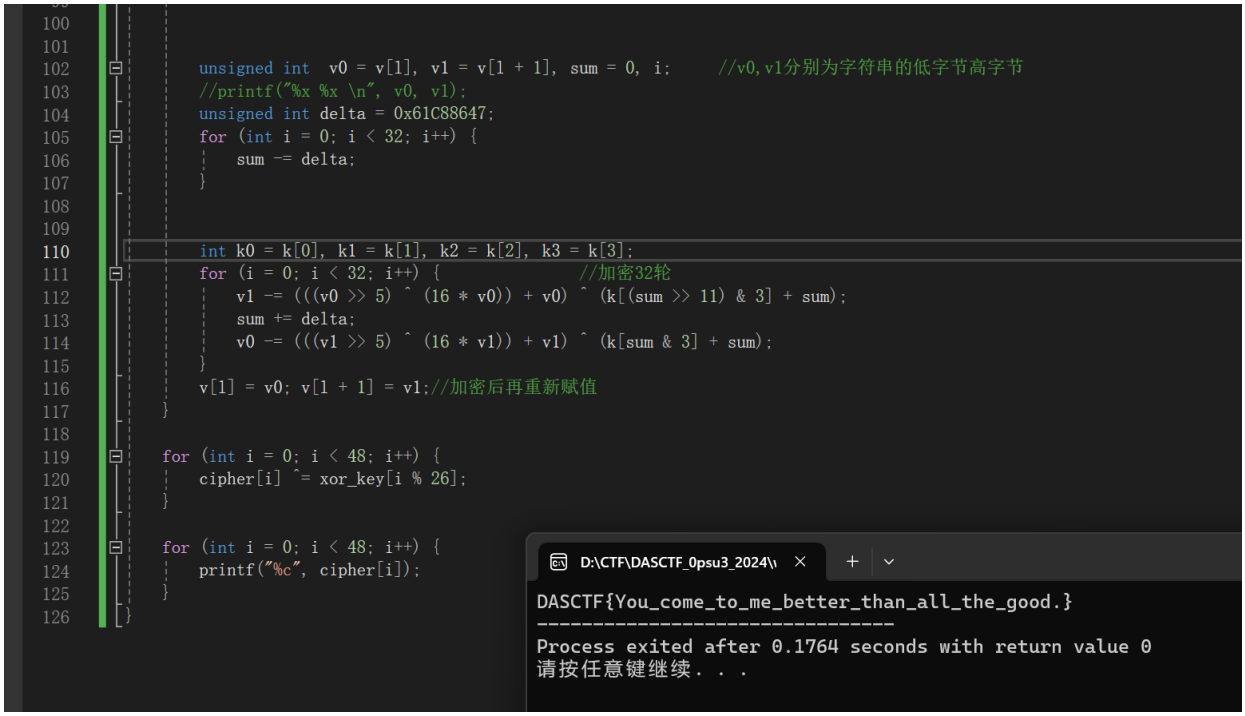
```
#include<stdio.h>
#include<stdint.h>

#define DELTA 0x11451419
#define MX (((z>>5^y<<2) + (y>>3^z<<4)) ^ ((sum^y) + (key[(p&3)^e] ^ z)))
unsigned char xor_key[] = "Laughter_is_poison_to_fear";
```

```
unsigned char xtea_xor_key[] =
{
  0xDA, 0x30, 0x23, 0xE3, 0xDC, 0x39, 0x82, 0x60, 0xA5, 0x44,
  0x68, 0xC2, 0x43, 0x7A, 0xBB, 0xE4, 0x50, 0xE1, 0x02, 0xC2,
  0x81, 0x59, 0xEA, 0x1E, 0xC6, 0x8B, 0x71, 0x38, 0x27, 0x83,
  0x94, 0xD8, 0xF4, 0x8D, 0x1A, 0x2A, 0x56, 0x8A, 0x4A, 0xD4,
  0x54, 0xDC, 0x24, 0x3F, 0xB9, 0xED, 0x7B, 0x9A
};
unsigned char xxtea[] =
{
   0x8f, 0x6c, 0xa6, 0x3f, 0x94, 0x3d, 0xf5, 0xd9, 0x36, 0x66, 0x51, 0xd7, 0x66,
0x2f, 0xb3, 0x8f,
        0xc0, 0x61, 0x9e, 0xce, 0xe9, 0xd7, 0xe1, 0xbf, 0x13, 0x14, 0x16, 0x14,
        0xc2, 0xe7, 0xc3, 0x3a, 0x7f, 0x94, 0xa1, 0xe7, 0x24, 0x0e, 0xa7, 0x5c,
0xd3, 0x77, 0xfe, 0x4f, 0x11, 0xdc, 0x69, 0x23
};

unsigned char cipher[] =
{
  0x18, 0x09, 0x1C, 0x14, 0x37, 0x1D, 0x16, 0x2D, 0x3C, 0x05,
  0x16, 0x3E, 0x02, 0x03, 0x10, 0x2C, 0x0E, 0x31, 0x39, 0x15,
  0x04, 0x3A, 0x39, 0x03, 0x0D, 0x13, 0x2B, 0x3E, 0x06, 0x08,
  0x37, 0x00, 0x17, 0x0B, 0x00, 0x1D, 0x1C, 0x00, 0x16, 0x06,
  0x07, 0x17, 0x30, 0x03, 0x30, 0x06, 0x0A, 0x71
};

void btea(uint8_t* input)
{
    int n = -12;
    uint32_t* v = (uint32_t*)input;
    uint8_t k[] = { 0x7b, 0x57, 0x68, 0x61, 0x74, 0x5f, 0x69, 0x73, 0x5f, 0x74,
0x68, 0x69, 0x73, 0x5f, 0x3f, 0x7d };
    uint32_t* key = (uint32_t*)k;
    uint32_t y, z, sum;
    unsigned p, rounds, e;
    if (n > 1)            /* Coding Part */
    {
        rounds = 6 + 52 / n;

        sum = 0;
        z = v[n - 1];
        do
        {
            sum += DELTA;
            e = (sum >> 2) & 3;
            for (p = 0; p < n - 1; p++)
            {
                y = v[p + 1];
                z = v[p] += MX;
            }
            y = v[0];
            z = v[n - 1] += MX;
```

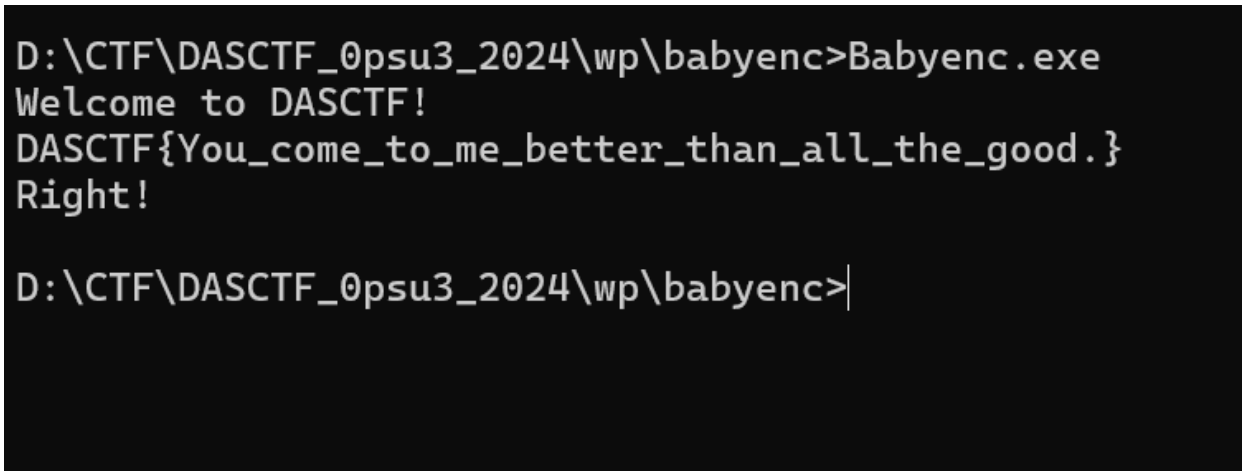
```
        } while (--rounds);
    }
    else if (n < -1)      /* Decoding Part */
    {
        n = -n;
        rounds = 6 + 52 / n;
        sum = rounds * DELTA;
        //printf("%d\n",rounds);
        y = v[0];
        do
        {
            e = (sum >> 2) & 3;
            for (p = n - 1; p > 0; p--)
            {
                z = v[p - 1];
                y = v[p] -= MX;
            }
            z = v[n - 1];
            y = v[0] -= MX;
            sum -= DELTA;
        } while (--rounds);
    }
}

int main()
{
    unsigned char a;

    for (int i = 0; i < 48; i++) {
        cipher[i] ^= xxtea[i];
        //printf("%c", a);
    }
    btea(cipher);

    for (int i = 0; i < 48; i++) {
        cipher[i] ^= xtea_xor_key[i];
        //printf("%c", a);
    }

    unsigned int* v = (unsigned int*)cipher;
    unsigned char key2[] = "{you_find_it_!?}";
    unsigned int* k = (unsigned int*)key2;
    for (int l = 0; l < 12; l += 2) {




        unsigned int  v0 = v[l], v1 = v[l + 1], sum = 0, i;      //v0,v1分别为字符串的低
字节高字节
        //printf("%x %x \n", v0, v1);
        unsigned int delta = 0x61C88647;
        for (int i = 0; i < 32; i++) {
```

```
            sum -= delta;
        }


        int k0 = k[0], k1 = k[1], k2 = k[2], k3 = k[3];
        for (i = 0; i < 32; i++) {                //加密32轮
            v1 -= (((v0 >> 5) ^ (16 * v0)) + v0) ^ (k[(sum >> 11) & 3] + sum);
            sum += delta;
            v0 -= (((v1 >> 5) ^ (16 * v1)) + v1) ^ (k[sum & 3] + sum);
        }
        v[l] = v0; v[l + 1] = v1;//加密后再重新赋值
    }

    for (int i = 0; i < 48; i++) {
        cipher[i] ^= xor_key[i % 26];
    }

    for (int i = 0; i < 48; i++) {
        printf("%c", cipher[i]);
    }
}
```

解出flag



验证flag


D:\CTF\DASCTF_0psu3_2024\wp\babyenc>Babyenc.exe
Welcome to DASCTF!
DASCTF{You_come_to_me_better_than_all_the_good.}
Right!
D:\CTF\DASCTF_0psu3_2024\wp\babyenc>