pofp面试题

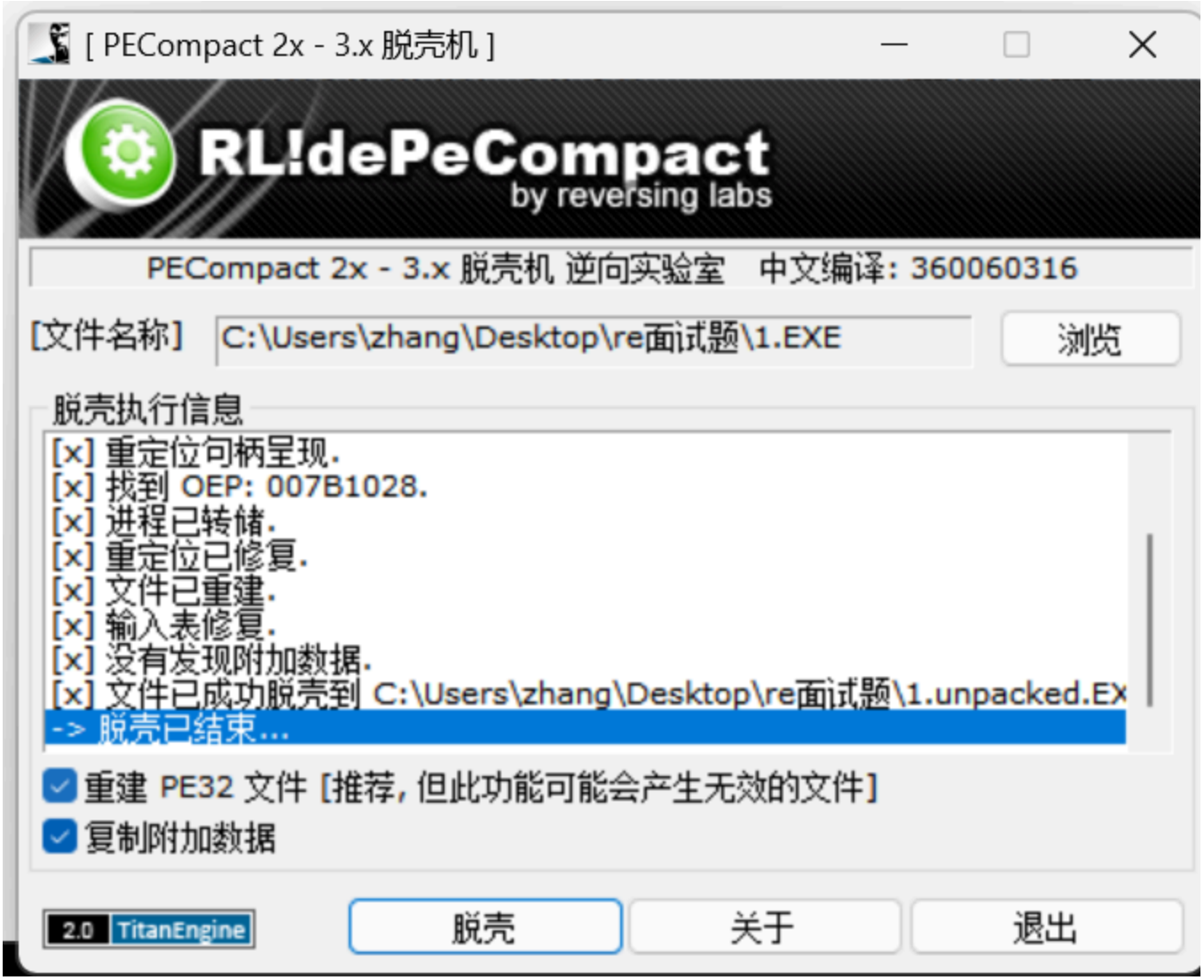
pofp面试题目wp

张程思

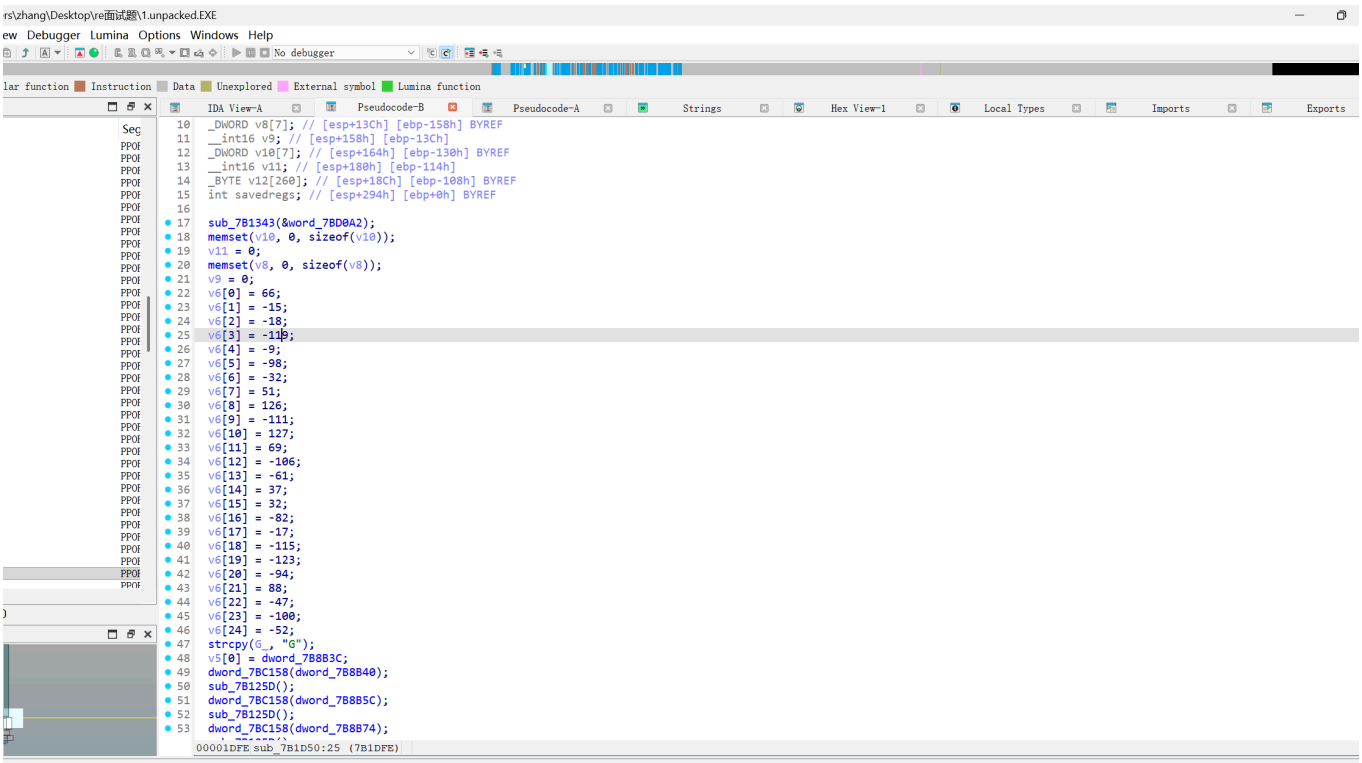
一,EXE

1,先检查一下发现是pe壳

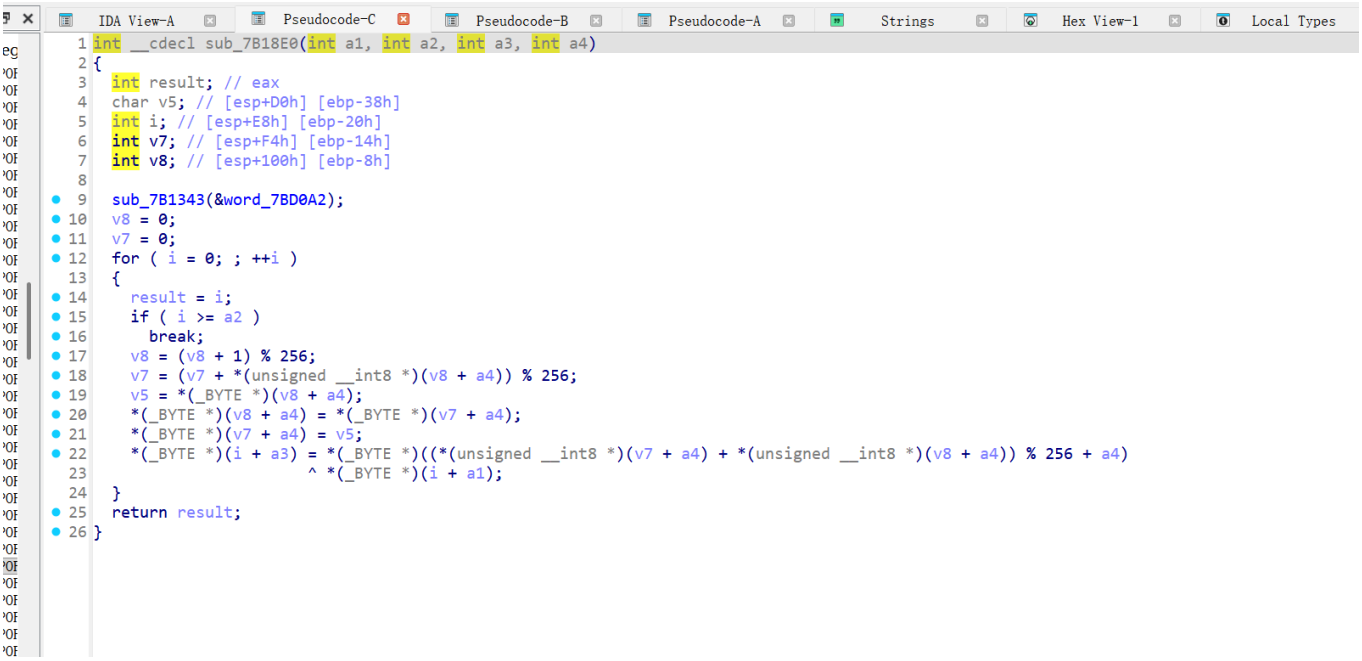
2,一开始按照标准upx的方式用x32dbg脱壳,我脱不出来,真的不会,从网上找了个自动脱pe壳的工具,估计只能脱标准壳,



3,按照第三题的一开始的步骤找主函数(因为先写的第三题的wp)



4,看一下加密函数,本以为是标准的rc4,但是没算出来结果,



5,求助ai之后把一处汇编发给它发现是有一处魔改了,就是这里

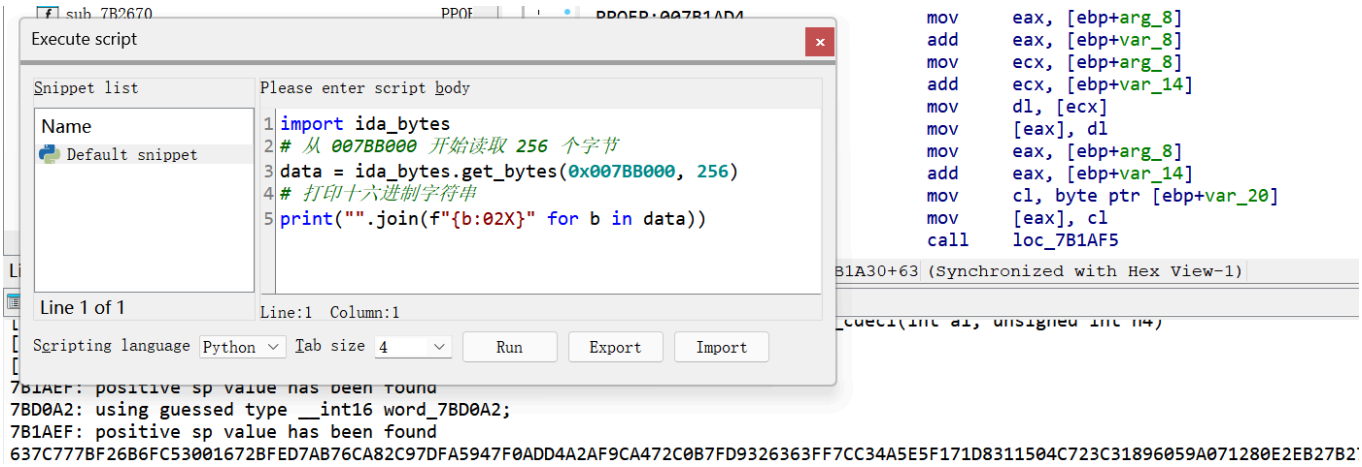
```
PPOFP:007B1B14                jmp     short loc_7B1B33
PPOFP:007B1B16 ; ---------------------------------------------------------------------------
PPOFP:007B1B16
PPOFP:007B1B16 loc_7B1B16:                            ; CODE XREF: sub_7B1A30+D3↑j
PPOFP:007B1B16                mov     eax, [ebp+var_8]
PPOFP:007B1B19                movzx   ecx, ds:byte_7BB000[eax]
PPOFP:007B1B20                mov     edx, [ebp+arg_8]
PPOFP:007B1B23                add     edx, [ebp+var_14]
PPOFP:007B1B26                movzx   eax, byte ptr [edx]
PPOFP:007B1B29                xor     eax, ecx
PPOFP:007B1B2B                mov     ecx, [ebp+arg_8]
PPOFP:007B1B2E                add     ecx, [ebp+var_14]
PPOFP:007B1B31                mov     [ecx], al
PPOFP:007B1B33
PPOFP:007B1B33 loc_7B1B33:                            ; CODE XREF: sub_7B1A30+E4↑j
PPOFP:007B1B33                jmp     loc_7B1A83
```

逻辑解析： 标准的 RC4 KSA 只做 `swap(S[i], S[j])`。但这个程序在交换之后，还多做了一步操作：`S[j] = S[j] ^ byte_7BB000[i]`

这相当于引入了**第二把密钥**（也就是那个 `byte_7BB000` 数组）。

6,由于我用工具脱的壳无法动调看,所以让ai写了个ptach找出来原始数据

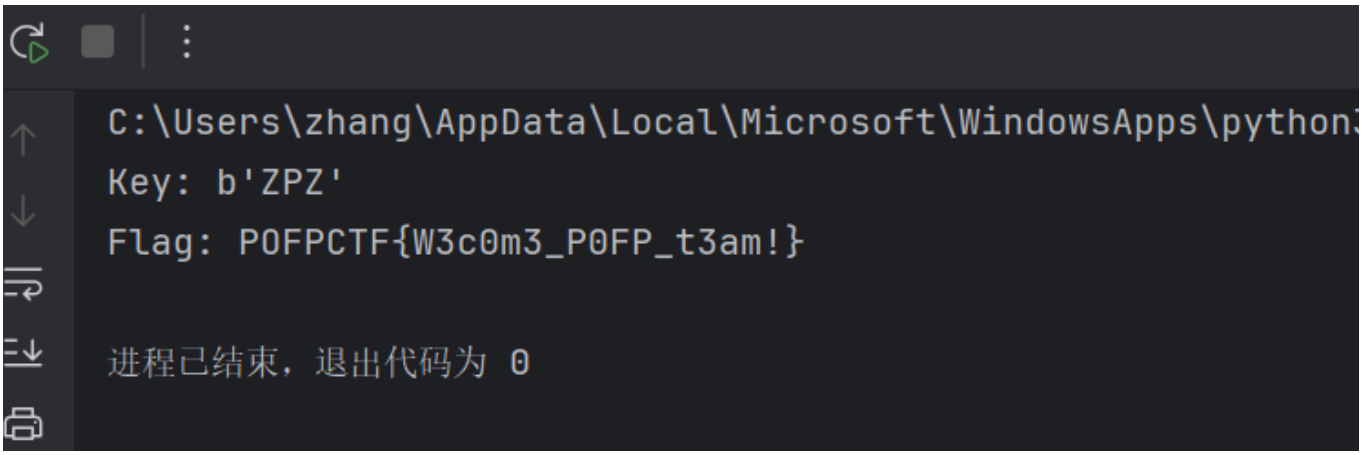
```
import ida_bytes
# 从 007BB000 开始读取 256 个字节
data = ida_bytes.get_bytes(0x007BB000, 256)
# 打印十六进制字符串
print("".join(f"{b:02X}" for b in data))
```



7,这样就得出解密脚本了

```
def custom_rc4_decrypt(key, ciphertext, T_hex):
    # 1. 准备 T 表
    T = bytes.fromhex(T_hex)

    # 2. 初始化 S 盒
    S = list(range(256))
    j = 0
```

```
    # 3. 魔改 KSA
    for i in range(256):
        j = (j + S[i] + key[i % len(key)]) % 256
        S[i], S[j] = S[j], S[i]
        # 核心修改点：再次异或 T 表
        S[j] = S[j] ^ T[i]

    # 4. 标准 PRGA
    i = j = 0
    out = []
    for char in ciphertext:
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]
        k = S[(S[i] + S[j]) % 256]
        out.append(char ^ k)

    return bytes(out)

# 数据部分
T_hex =
"637C777BF26B6FC53001672BFED7AB76CA82C97DFA5947F0ADD4A2AF9CA472C0B7FD9326363FF
7CC34A5E5F171D8311504C723C31896059A071280E2EB27B27509832C1A1B6E5AA0523BD6B329E
32F8453D100ED20FCB15B6ACBBE394A4C58CFD0EFAAFB434D338545F9027F503C9FA851A3408F9
29D38F5BCB6DA2110FFF3D2CD0C13EC5F974417C4A77E3D645D197360814FDC222A908846EEB81
4DE5E0BDBE0323A0A4906245CC2D3AC629195E479E7C8376D8DD54EA96C56F4EA657AAE08BA782
52E1CA6B4C6E8DD741F4BBD8B8A703EB5664803F60E613557B986C11D9EE1F8981169D98E949B1
E87E9CE5528DF8CA1890DBFE6426841992D0FB054BB16"
cipher_ints = [66, -15, -18, -119, -9, -98, -32, 51, 126, -111, 127, 69, -106,
-61, 37, 32, -82, -17, -115, -123, -94, 88, -47, -100, -52]
ciphertext = bytes([(x + 256) % 256 for x in cipher_ints]) + b'G'
key = b"ZPZ"

# 运行
print(f"Flag: {custom_rc4_decrypt(key, ciphertext, T_hex).decode()}")
```



二,joker.exe

1,一进去发现是汇编层并且全冒红且无法正常f5编译伪代码即判断出有花指令
先把上面的那些dd和刚才这个dd都按c键转为汇编

```
.text:00403F6D                 dec     ebp
.text:00403F6E                 rcl     byte ptr [ecx-18h], 2Bh
.text:00403F6E ; ---------------------------------------------------------------------------
.text:00403F72                 dw 0FFFFh
.text:00403F74                 dd 8C483FFh, 68B84589h, 3E8h
.text:00403F80                 db 0FFh, 15h
.text:00403F82                 dd offset Sleep
.text:00403F86                 dw 45C7h
.text:00403F88                 dd 60B75CC8h, 0CC45C7ADh, 0F3AF1F80h, 0D6D045C7h, 0C76C4659h
.text:00403F9C                 dd 3A20D445h, 45C7B72Ah, 0B3A9C7D8h, 0DC45C79Bh, 6E053A18h
.text:00403FB0                 dd 27E045C7h, 0C786B0F9h, 4149E445h, 45C7AC67h, 17E29AE8h
.text:00403FC4                 dd 0EC45C7F9h, 1FAF3A12h, 11F045C7h, 0C7FE1F67h, 1CA3F445h
.text:00403FD8                 dd 45C779FEh, 2DF8h
.text:00403FE0                 db 0, 68h
.text:00403FE2                 dd offset aWelcomeToPofpc ; "welcome to POFPCTF\n"
.text:00403FE6                 dw 35E8h
.text:00403FE8                 dd 83FFFFD0h, 558D04C4h, 458D52C4h, 0A5E850C0h, 83FFFFFEh
.text:00403FFC                 dd 458908C4h, 3E868B8h, 15FF0000h
.text:00404008                 dd offset Sleep
.text:0040400C                 dd 6DE8C933h, 50FFFFD0h
.text:00404014                 db 0FFh, 15h
.text:00404016                 dd offset srand
.text:0040401A                 dw 0C483h
.text:0040401C                 db 4, 0FFh, 15h
.text:0040401F                 dd offset rand
.text:00404023                 db 89h
.text:00404024                 db 45h, 0ACh, 68h
.text:00404027                 dd offset aInputYourLucky ; "input your lucky number:\n"
.text:0040402B                 db 0E8h
.text:0040402C                 dd 0FFFFCFF0h, 3304C483h, 110374C0h, 4D8D3322h
.text:0040403C                 db 0BCh, 51h, 68h
.text:0040403F                 dd offset aD            ; "%d"
.text:00404043                 db 0E8h
.text:00404044                 dd 0FFFFD008h, 8308C483h, 3E7401F8h
.text:00404050 ; ---------------------------------------------------------------------------
.text:00404050                 push    offset aInvalidInput ; "invalid input\n"
00003353 00403F53: .text:00403F53 (Synchronized with Hex View-1)
```

```
.text:00403F67                 xor     ecx, [ebp-72AF3BBBh]
.text:00403F6D                 dec     ebp
.text:00403F6E                 rcl     byte ptr [ecx-18h], 2Bh
.text:00403F6E ; ---------------------------------------------------------------------------
.text:00403F72                 dw 0FFFFh
.text:00403F74 ; ---------------------------------------------------------------------------
.text:00403F74                 inc     dword ptr [ebx+458908C4h]
.text:00403F7A                 mov     eax, 3E868h
.text:00403F7F                 add     bh, bh
.text:00403F81                 adc     eax, offset Sleep
.text:00403F86                 mov     dword ptr [ebp-38h], 0AD60B75Ch
.text:00403F8D                 mov     dword ptr [ebp-34h], 0F3AF1F80h
.text:00403F94                 mov     dword ptr [ebp-30h], 6C4659D6h
.text:00403F9B                 mov     dword ptr [ebp-2Ch], 0B72A3A20h
.text:00403FA2                 mov     dword ptr [ebp-28h], 9BB3A9C7h
.text:00403FA9                 mov     dword ptr [ebp-24h], 6E053A18h
.text:00403FB0                 mov     dword ptr [ebp-20h], 86B0F927h
.text:00403FB7                 mov     dword ptr [ebp-1Ch], 0AC674149h
.text:00403FBE                 mov     dword ptr [ebp-18h], 0F917E29Ah
.text:00403FC5                 mov     dword ptr [ebp-14h], 1FAF3A12h
.text:00403FCC                 mov     dword ptr [ebp-10h], 0FE1F6711h
.text:00403FD3                 mov     dword ptr [ebp-0Ch], 79FE1CA3h
.text:00403FDA                 mov     dword ptr [ebp-8], 2Dh ; '-'
.text:00403FE1                 push    offset aWelcomeToPofpc ; "welcome to POFPCTF\n"
.text:00403FE6                 call    sub_401020
.text:00403FEB                 add     esp, 4
.text:00403FEE                 lea     edx, [ebp-3Ch]
.text:00403FF1                 push    edx
.text:00403FF2                 lea     eax, [ebp-40h]
.text:00403FF5                 push    eax
.text:00403FF6                 call    sub_403EA0
.text:00403FFB                 add     esp, 8
.text:00403FFE                 mov     [ebp-48h], eax
.text:00404001                 push    3E8h
.text:00404006                 call    ds:Sleep
.text:0040400C                 xor     ecx, ecx
.text:0040400E                 call    sub_401080
.text:00404013                 push    eax
.text:00404014                 call    ds:srand
.text:0040401A                 add     esp, 4
.text:0040401D                 call    ds:rand
00003406 00404006: .text:00404006 (Synchronized with Hex View-1)
```

```
python@googlegroups.com>
---------------------------------------
time
runtime
```

2,然后开找花指令

```
.text:00403F50                 push    ebx
.text:00403F51                 push    esi
.text:00403F52                 push    edi
.text:00403F53                 mov     dword ptr [ebp-3Ch], 0
.text:00403F5A                 mov     dword ptr [ebp-48h], 0
.text:00403F61                 xor     eax, eax
.text:00403F63                 nop
.text:00403F64                 nop
.text:00403F65                 nop
.text:00403F66                 nop
.text:00403F67                 xor     ecx, [ebp-72AF3BBBh]
.text:00403F6D                 dec     ebp
.text:00403F6E                 rcl     byte ptr [ecx-18h], 2Bh
.text:00403F6E ; ---------------------------------------------------------------------------
.text:00403F72                 dw 0FFFFh
.text:00403F74 ; ---------------------------------------------------------------------------
.text:00403F74                 inc     dword ptr [ebx+458908C4h]
```

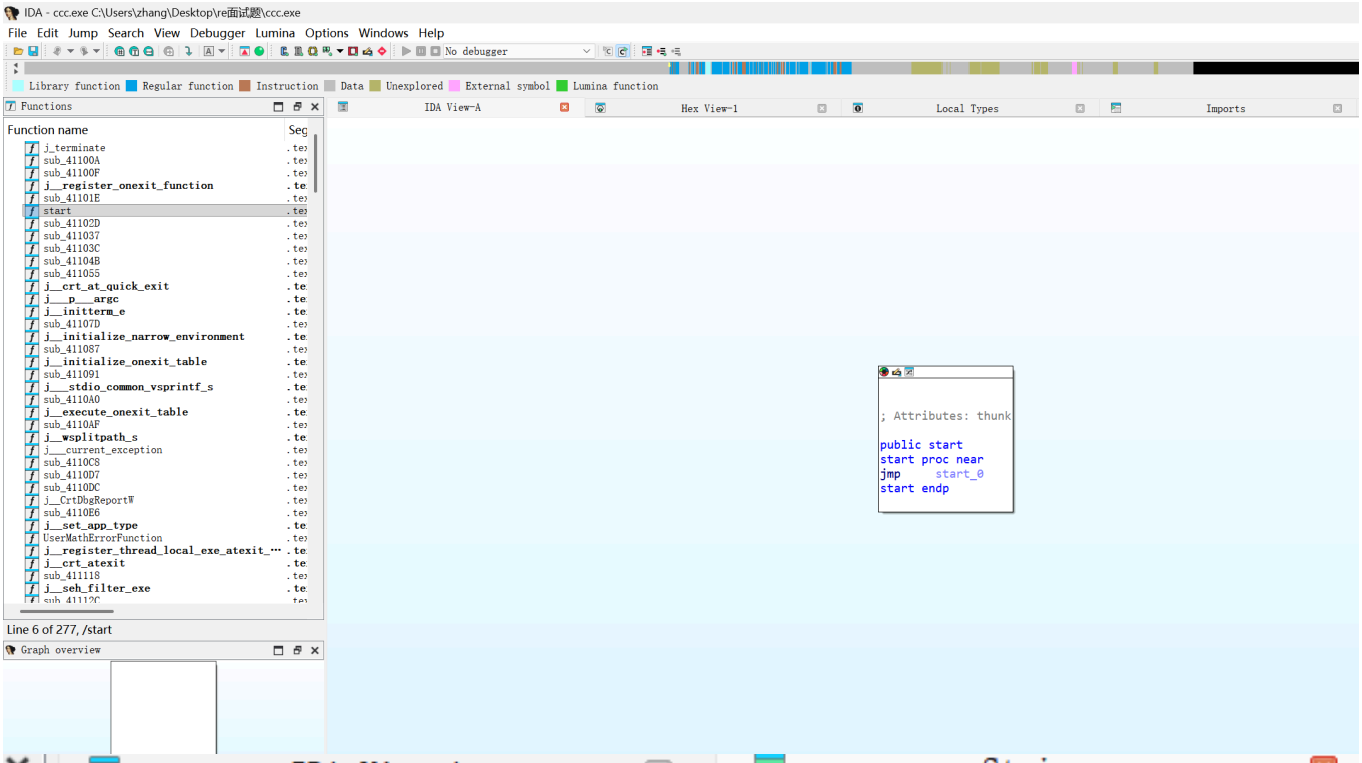
3,找了三处,一处堆栈跳转,两处混淆代码,之后点main点u取消定义再点p,然后我还是无法f5反编译,干脆把汇编层丢给ai梭哈了

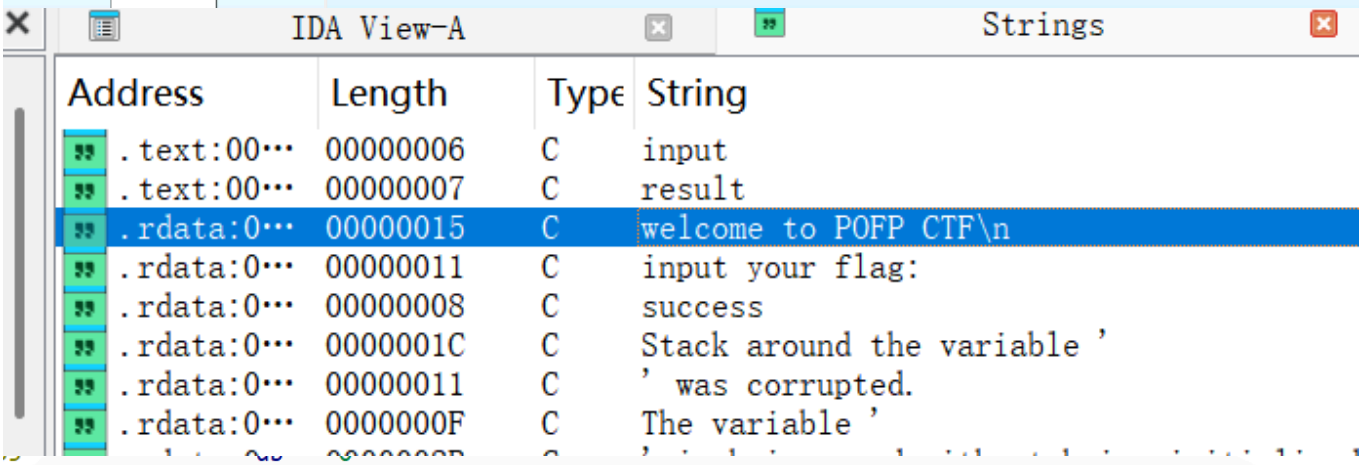
```python
            z = v[p - 1]
            y = v[p] = (v[p] - mx(sum, y, z, p, e, k)) & 0xFFFFFFFF
        z = v[n - 1]
        sum = (sum - 0x9E3779B9) & 0xFFFFFFFF
    return v

# 密文数据 (来自栈上的硬编码)
ciphertext = [
    0xAD60B75C, 0xF3AF1F80, 0x6C4659D6, 0xB72A3A20,
    0x9BB3A9C7, 0x6E053A18, 0x86B0F927, 0xAC674149,
    0xF917E29A, 0x1FAF3A12, 0xFE1F6711, 0x79FE1CA3
]

# 密钥 (来自数据段字符串 "®igUg4oìi#Á?3_gBÀ")
key_bytes = b'\xAE\x69\x67\x55\x67\x34\x6F\xEC\x69\x23\xC1\x3F\x33\x5F\x67\x42'
key = list(struct.unpack('<4I', key_bytes))

# 解密
decrypted = xxtea_decrypt(ciphertext, key)
flag = b''.join([struct.pack('<I', x) for x in decrypted])
print("Flag:", flag.decode('utf-8', errors='ignore').strip('\x00'))
```

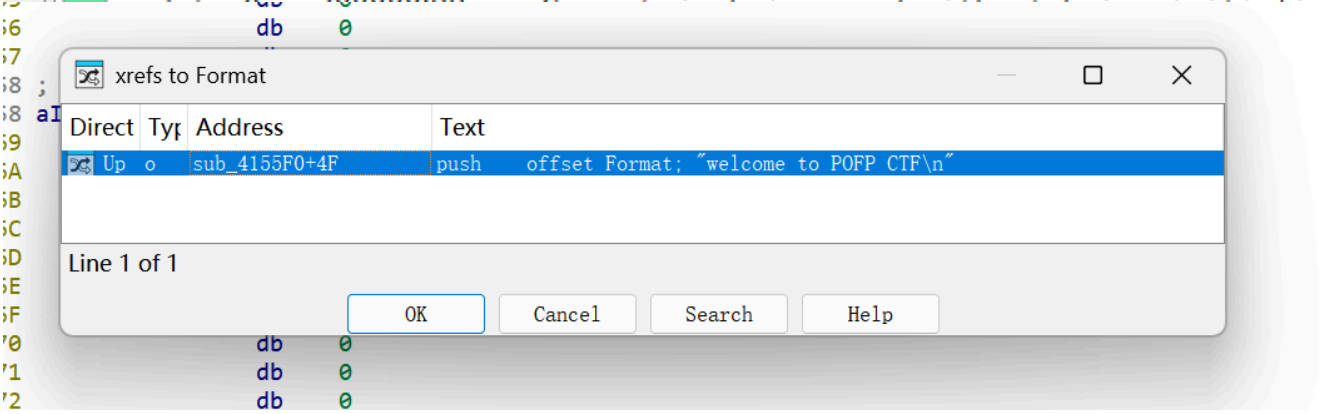
解密逻辑说明： 程序虽然设置了 `rand()` 随机数检查，但实际上 Flag 的数据是静态存储在栈上的。只要我们知道了加密算法（XXTEA）和密钥（数据段中的字符串），就可以绕过随机数检查，直接解密出 Flag。之前遇到的花指令（Junk Code）主要是为了阻碍你发现这段 XXTEA 解密逻辑。

三,ccc.exe

1,先找一下主函数在哪里吧,shift+f12看字符串,点击welcome这里,再点x找到引用位置







2,找一下真正的算法函数,发现是xxtea,再去找一下密文,然后我发现这个密文解不出来flag,

IDA View-A | Pseudocode-A | Strings | Hex View-1 | Local Types | Import

```
11  int n7; // [esp+F4h] [ebp-BCh]
12  char Buffer[72]; // [esp+100h] [ebp-B0h] BYREF
13  _BYTE Buf1[40]; // [esp+148h] [ebp-68h] BYREF
14  _DWORD v12[6]; // [esp+170h] [ebp-40h] BYREF
15  _DWORD v13[9]; // [esp+188h] [ebp-28h] BYREF
16  int savedregs; // [esp+1B0h] [ebp+0h] BYREF
17
18  sub_41132F(&unk_41C00F);
19  v12[0] = 69;
20  v12[1] = 86;
21  v12[2] = 327;
22  v12[3] = 69;
23  sub_4110D7("welcome to POFP CTF\n", v4);
24  sub_4110D7("input your flag:", v5);
25  _acrt_iob_func(0);
26  Stream = (FILE *)sub_411253();
27  fgets(Buffer, 64, Stream);
28  if ( sub_411253() )
29  {
30    if ( j_strlen(Buffer) == 32 )
31    {
32      j_memset(v13, 0, 0x20u);
33      for ( n7 = 0; n7 <= 7; ++n7 )
34        v13[n7] = Buffer[4 * n7] | (Buffer[4 * n7 + 1] << 8) | (Buffer[4 * n7 + 2] << 16) | (Buffer[4 * n7 + 3] << 24);
35      for ( n7_1 = 0; n7_1 <= 7; n7_1 += 2 )
36        sub_4113CA(&v13[n7_1], v12);
37      for ( n7_2 = 0; n7_2 <= 7; ++n7_2 )
38      {
39        for ( n3 = 0; n3 <= 3; ++n3 )
40          Buf1[4 * n7_2 + n3] = v13[n7_2] >> (8 * n3);
41      }
42      if ( !j_memcmp(Buf1, &Buf2_, 0x20u) )
43        puts("success");
44      else
45        puts("fake");
46      sub_411253();
47      LODWORD(v1) = 0;
48    }
49    else
50    {
51      puts("fake");
52      sub_411253();
53      LODWORD(v1) = 0;
54    }
```

IDA View-A | Pseudocode-A | Strings | Hex View-1 | Local Types

```
1  int __cdecl sub_414D30(unsigned int *a1, int a2)
2  {
3    int n4; // eax
4    unsigned int n0x23; // [esp+D0h] [ebp-2Ch]
5    unsigned int v4; // [esp+DCh] [ebp-20h]
6    unsigned int v5; // [esp+E8h] [ebp-14h]
7    unsigned int v6; // [esp+F4h] [ebp-8h]
8
9    sub_41132F(&unk_41C00F);
10   v4 = *a1;
11   v5 = a1[1];
12   v6 = 0;
13   for ( n0x23 = 0; n0x23 <= 0x23; ++n0x23 )
14   {
15     v4 += (v6 + *(_DWORD *)(a2 + 4 * (v6 & 3))) ^ (v5 + ((16 * v5) ^ (v5 >> 5)));
16     v6 += 1640531783;
17     v5 += (v6 + *(_DWORD *)(a2 + 4 * ((v6 >> 11) & 3))) ^ (v4 + ((16 * v4) ^ (v4 >> 5)));
18   }
19   *a1 = v4;
20   n4 = 4;
21   a1[1] = v5;
22   return n4;
23 }
```

```
.data:0041A0E4                  align 8
.data:0041A0E8 dword_41A0E8     dd 1                    ; DATA XREF: sub
.data:0041A0EC Buf2_            db 0Bh                  ; DATA XREF: sub
.data:0041A0EC                                          ; sub_411790+4E↑
.data:0041A0ED                  db 0F3h
.data:0041A0EE                  db 0A5h
.data:0041A0EF                  db 0DAh
.data:0041A0F0                  db 0ECh
.data:0041A0F1                  db  93h
.data:0041A0F2                  db 0DEh
.data:0041A0F3                  db 0F4h
.data:0041A0F4                  db  5Dh ; ]
.data:0041A0F5                  db  1Ch
.data:0041A0F6                  db  1Bh
.data:0041A0F7                  db 0EAh
.data:0041A0F8                  db  14h
.data:0041A0F9                  db  44h ; D
.data:0041A0FA                  db  99h
.data:0041A0FB                  db 0ADh
.data:0041A0FC                  db 0ACh
.data:0041A0FD                  db  5Fh ; _
.data:0041A0FE                  db  4Bh ; K
.data:0041A0FF                  db  12h
.data:0041A100                  db  20h
.data:0041A101                  db 0E9h
.data:0041A102                  db  3Eh ; >
.data:0041A103                  db 0A1h
.data:0041A104                  db  74h ; t
.data:0041A105                  db    6
.data:0041A106                  db 0BDh
.data:0041A107                  db    6
.data:0041A108                  db  99h
.data:0041A109                  db  1Bh
.data:0041A10A                  db 0EDh
.data:0041A10B                  db 0DAh
.data:0041A10C                  db    0
.data:0041A10D                  db    0
.data:0041A10E                  db    0
.data:0041A10F                  db    0
.data:0041A110                  db    0
```

3,猜测和isctf的recall那个题都差不多,密文被交叉引用改了数值,看一下另一个修改密文的函数,就推断出来真正的密文是需要被上次找的密文异或一下

IDA View-A | Pseudocode-A | Strings

```
1  int sub_411790()
2  {
3    int result; // eax
4    int n32; // [esp+D0h] [ebp-8h]
5
6    result = sub_41132F(&unk_41C00F);
7    for ( n32 = 0; n32 < 32; ++n32 )
8    {
9      Buf2_[n32] ^= 0x32u;
10     result = n32 + 1;
11   }
12   return result;
13 }
```

得出解密脚本

```
import struct

def decrypt_xtea(v0, v1, key):
    # 魔改参数
    delta = 1640531783
    rounds = 36
    mask = 0xFFFFFFFF

    # 初始化 Sum (加密结束时的值)
    sum_val = (delta * rounds) & mask

    for _ in range(rounds):
        # 逆向运算 v1
        term1 = (sum_val + key[(sum_val >> 11) & 3]) & mask
        term2 = (v0 + ((v0 << 4) ^ (v0 >> 5))) & mask
        v1 = (v1 - (term1 ^ term2)) & mask

        # 还原 sum
        sum_val = (sum_val - delta) & mask

        # 逆向运算 v0
        term1 = (sum_val + key[sum_val & 3]) & mask
        term2 = (v1 + ((v1 << 4) ^ (v1 >> 5))) & mask
        v0 = (v0 - (term1 ^ term2)) & mask

    return v0, v1
```

```
if __name__ == '__main__':
    # 1. 原始密文 (IDA .data)
    raw_hex =
"0BF3A5DAEC93DEF45D1C1BEA144499ADAC5F4B1220E93EA17406BD06991BEDDA"
    raw_bytes = bytes.fromhex(raw_hex)

    # 2. 关键一步：模拟程序运行时的预处理 (XOR 0x32)
    real_cipher = bytes([b ^ 0x32 for b in raw_bytes])

    # 3. 密钥 (注意 327 未截断)
    key = [69, 86, 327, 69]

    # 4. 分组解密
    v_list = list(struct.unpack('<8I', real_cipher))
    flag_parts = []

    for i in range(0, 8, 2):
        d0, d1 = decrypt_xtea(v_list[i], v_list[i+1], key)
        flag_parts.append(d0)
        flag_parts.append(d1)

    # 5. 输出 Flag
    flag = struct.pack('<8I', *flag_parts).decode()
    print(f"Flag: {flag}")
```

四,易语言

1,