逆向学习newstarweek1和2的题目

逆向题目wp

张程思

一,X0r

1,先用die查看发现无壳,之后ida打开

```
_main();
puts("Please input your flag: ");
scanf("%25s", Str);
n24 = strlen(Str);
if ( n24 == 24 )
{
  for ( n24_1 = 0; n24_1 < n24; ++n24_1 )
  {
    if ( n24_1 % 3 )
    {
      if ( n24_1 % 3 == 1 )
        Str[n24_1] ^= 0x11u;
      else
        Str[n24_1] ^= 0x45u;
    }
    else
    {
      Str[n24_1] ^= 0x14u;
    }
  }
  v5[0] = 19;
  v5[1] = 19;
  v5[2] = 81;
  for ( n24_2 = 0; n24_2 < n24; ++n24_2 )
    Str[n24_2] ^= v5[n24_2 % 3];
  strcpy(Str2, "anu`ym7wKLl$P]v3q%D]lHpi");
  if ( !strcmp(Str, Str2) )
    puts("Right flag!");
  else
    puts("Wrong flag!");
  return 0;
}
else
{
  puts("Wrong flag length!");
```

根据主函数可知该加密方式为纯异或加密,解题脚本如下

```
# decode_flag.py
target = "anu`ym7wKLl$P]v3q%D]lHpi"
keys = [0x07, 0x02, 0x14]
```

```
flag = ''.join(chr(ord(ch) ^ keys[i % 3]) for i, ch in enumerate(target))
print(flag)
```

```
flag{y0u_Kn0W_b4s1C_xOr}
```

二,Strange Base

1,还是无壳,查看伪代码

```
1  __int64 __fastcall main()
2  {
3    int binlength; // eax
4    size_t Size; // rax
5    char enc[48]; // [rsp+20h] [rbp-90h] BYREF
6    unsigned __int8 output[48]; // [rsp+50h] [rbp-60h] BYREF
7    unsigned __int8 input[48]; // [rsp+80h] [rbp-30h] BYREF
8
9    _main();
0    memset(input, 0, sizeof(input));
1    memset(output, 0, sizeof(output));
2    puts("It's time to show your flag to me~~~");
3    strcpy(enc, "T>6uTqOatL39aP!YIqruyv(YBA!8y7ouCa9=");
4    scanf_s("%s", input);
5    binlength = strlen((const char *)input);
6    base64_encode(input, (char *)output, binlength);
7    Size = strlen(enc);
8    if ( !memcmp(output, enc, Size) )
9      printf("Oh! You're awesome!!!");
0    else
1      puts("Wrong!");
2    return 0;
3  }
```

Address	Length	Type	String
.rdata:0…	00000041	C	HElLo!A=CrQzy-B4S3|is'waITt1ng&Y0u^{/(>v<)*}GO~256789pPqWXVKJNMF
.rdata:0…	00000025	C	It's time to show your flag to me~~~
.rdata:0…	00000016	C	Oh! You're awesome!!!

2,发现是base64编码,但是码表被换了,脚本如下(在线网站也可解)

```
# decode_custom_b64.py
import base64
import sys
import string
```

```
# 用户输入的密文和被替换后的 base64 字母表
cipher = "T>6uTqOatL39aP!YIqruyv(YBA!8y7ouCa9="
custom_alphabet = "HElLo!A=CrQzy-
B4S3|is'waITt1ng&Y0u^{/(>v<)*}GO~256789pPqWXVKJNMF"

# 标准 Base64 字母表
std_alphabet =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"

if len(custom_alphabet) != 64:
    print("Error: custom alphabet length must be 64.")
    sys.exit(1)

# 构建替换表：custom -> standard
trans_table = str.maketrans({custom_alphabet[i]: std_alphabet[i] for i in
range(64)})

# 把密文中的字符从 custom 映射回 standard
translated = cipher.translate(trans_table)

try:
    decoded_bytes = base64.b64decode(translated, validate=False)
except Exception as e:
    print("base64 decode error:", e)
    sys.exit(1)

# 尝试把字节解为 utf-8 文本，失败则显示原始 bytes
try:
    decoded_text = decoded_bytes.decode('utf-8', errors='ignore')
except:
    decoded_text = None

# 去掉末尾不可打印字符（控制字符），更整洁地显示
def strip_trailing_control(s):
    # 从末尾剔除非打印字符（只保留 printable 和常见换行制表）
    while s and s[-1] not in (string.printable):
        s = s[:-1]
    # 进一步去掉末尾非字母数字标点（若是控制字符）
    while s and (ord(s[-1]) < 32 or ord(s[-1]) == 127):
        s = s[:-1]
    return s

if decoded_text is not None:
    clean = strip_trailing_control(decoded_text)
    print("Decoded (utf-8, cleaned):")
    print(clean)
```

```
else:
    print("Decoded bytes (hex):", decoded_bytes.hex())
    print("Decoded bytes (raw):", decoded_bytes)
```

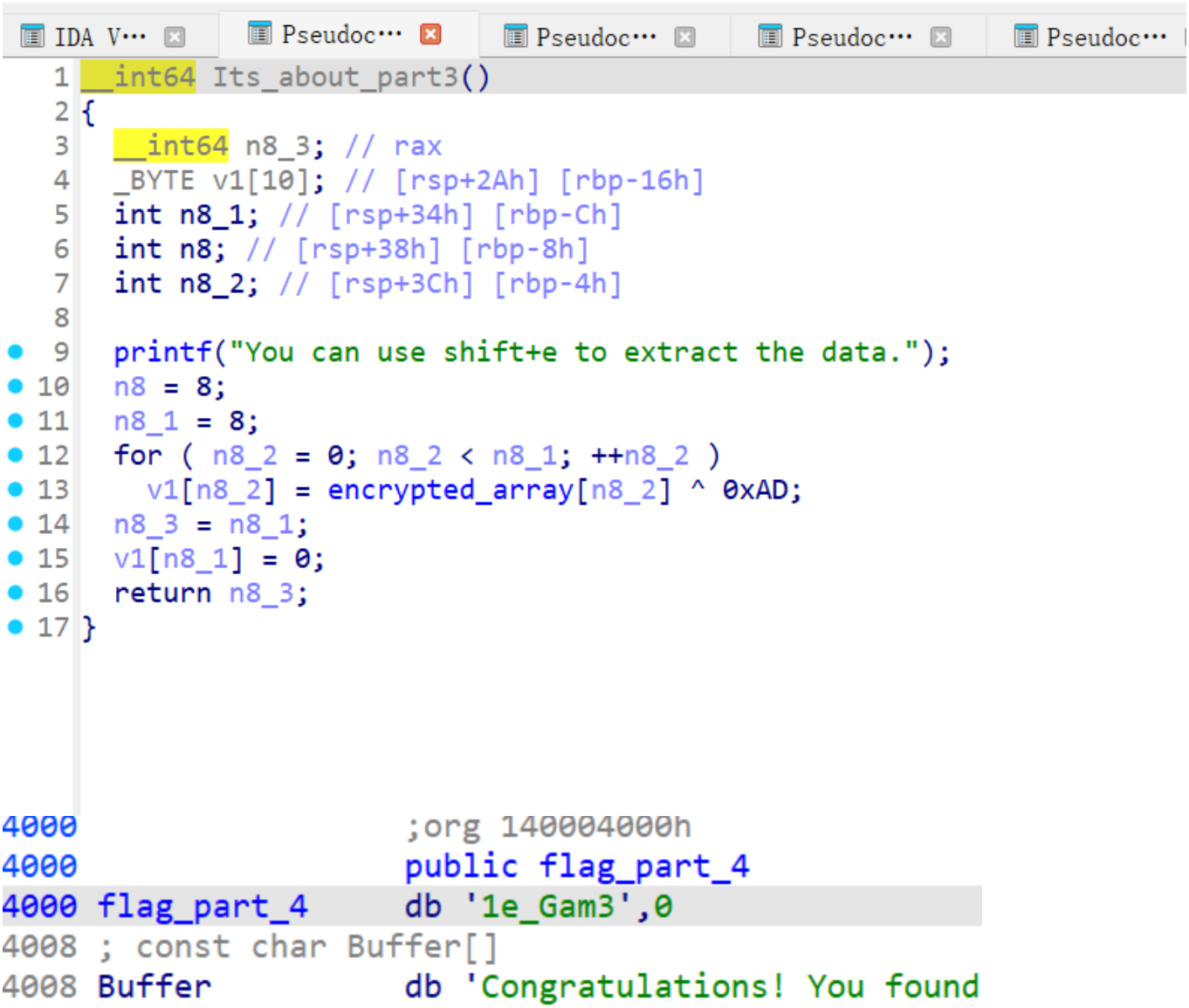
三,Pullze

1,无壳,ida分析代码,给的提示很多了这里面

```
int Puzzle_Challenge()
{
  char Destination[4]; // [rsp+26h] [rbp-1Ah] BYREF
  char *Y0u; // [rsp+30h] [rbp-10h]
  char *Source; // [rsp+38h] [rbp-8h]

  Source = "Do_";
  Y0u = "Y0u_";
  strcpy(Destination, "Do_");
  strcat(Destination, Y0u);
  return puts("Welcome to the Jigsaw Puzzle Game!");
}
```

IDA V··· | Pseudoc··· | Pseudoc··· | Pseudoc··· | Pseudoc··· | Pseudoc··· | Ps

```
int Like_7his_Jig()
{
  puts("Congratulations! You found the second part of the flag--The function name.");
  return printf("Observing function names is an important step in reverse engineering.");
}
```



2,第三处是异或加密,解题脚本如下

```
# 从 IDA 中找到的加密数据 (encrypted_array 的前 8 个字节)
encrypted_data = bytes([0xDE, 0xED, 0xDA, 0xF2, 0xDD, 0xD8, 0xD7, 0xD7])

# 从 Its_about_part3 函数中找到的异或密钥
xor_key = 0xAD

# 用于存储解密后的字节
decrypted_data = bytearray()

# 对每个字节执行异或操作
for byte in encrypted_data:
  decrypted_byte = byte ^ xor_key
  decrypted_data.append(decrypted_byte)

# 将解密后的字节串转换为字符串 (假设是 ASCII 或 UTF-8 编码)
```

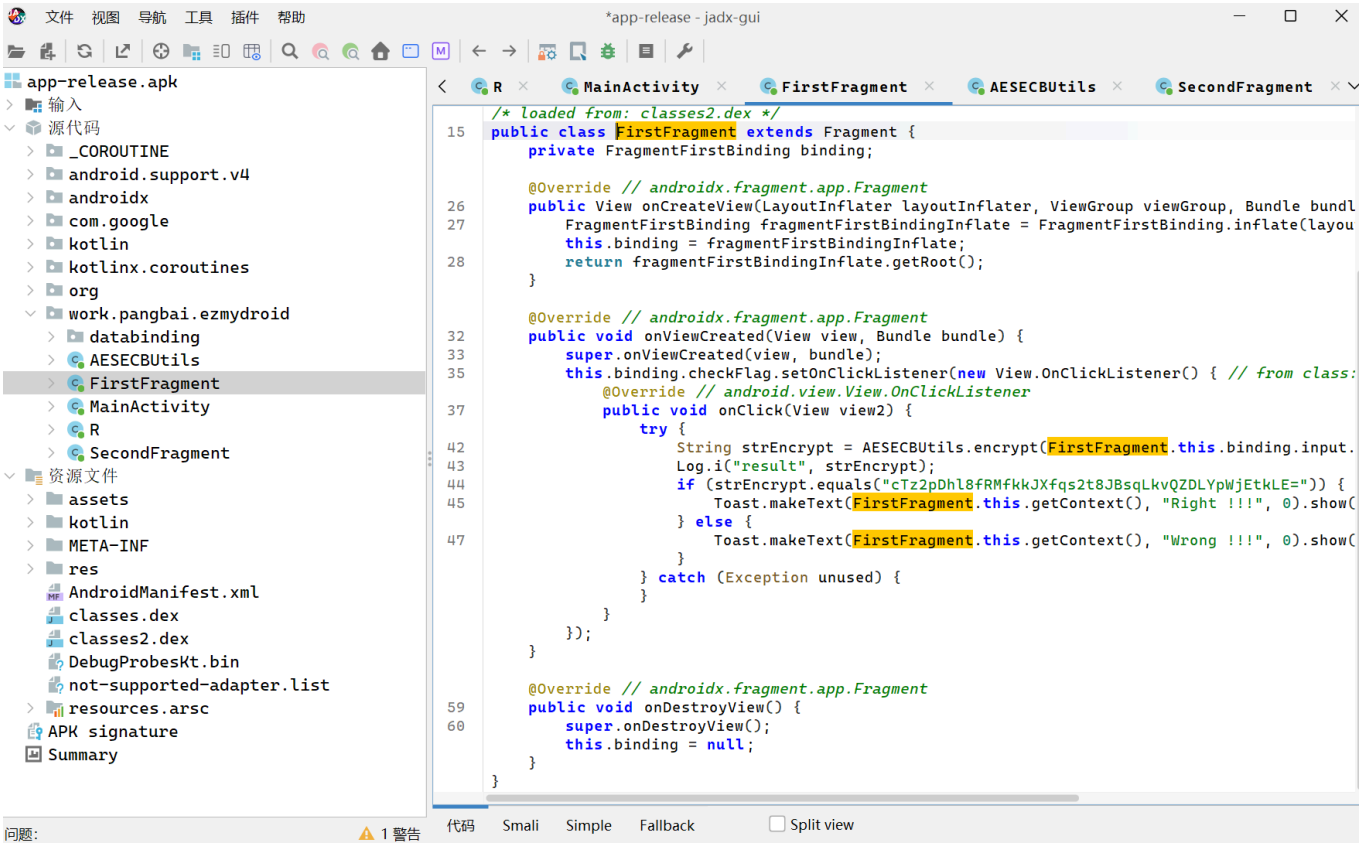
```
# 使用 .decode() 并处理可能的错误（虽然这里预期不会有）
try:
  decrypted_string = decrypted_data.decode('utf-8')
except UnicodeDecodeError:
  decrypted_string = "Error decoding bytes"

# 打印结果
print(f"Encrypted Hex: {encrypted_data.hex()}")
print(f"XOR Key:       0x{xor_key:02x}")
print(f"Decrypted Hex: {decrypted_data.hex()}")
print(f"Decrypted Part 3: {decrypted_string}")
```

3,最终拼出来的flag为

```
flag{Do_Y0u_Like_7his_Jigs@w_puzz1e_Gam3}
```

四,EzMyDroid

1,根据题目提示是安卓逆向,用jadx打开,找到关键逻辑的代码如下



2,发现该代码是AES-128 加密+ Base64 编码

```
# decrypt_aes_ecb.py
import base64
from Crypto.Cipher import AES
```

```
cipher_b64 = "cTz2pDhl8fRMfkkJXfqs2t8JBsqLkvQZDLYpWjEtkLE="
key = b"1145141919810000"  # 与 Java 代码中相同的 16 字节 key

ct = base64.b64decode(cipher_b64)
cipher = AES.new(key, AES.MODE_ECB)
pt_bytes = cipher.decrypt(ct)

def unpad_pkcs7(data):
    if len(data) == 0:
        return data
    pad = data[-1]
    if pad < 1 or pad > AES.block_size:
        return data
    if data[-pad:] != bytes([pad]) * pad:
        return data
    return data[:-pad]

unpadded = unpad_pkcs7(pt_bytes)
try:
    plaintext = unpadded.decode('utf-8')
except:
    plaintext = repr(unpadded)

print("Decrypted plaintext:", plaintext)
```

得出flag为

```
flag{@_g00d_st@r7_f0r_ANDROID}
```

五,plzdebugme

1,查看代码发现需要用gdb动态调试

```
int __fastcall main(int argc, const char **argv, const char **envp)
{
  int keylen; // eax
  char c; // [rsp+Fh] [rbp-5E1h]
  uint32_t v[2]; // [rsp+28h] [rbp-5C8h] BYREF
  uint32_t k[4]; // [rsp+30h] [rbp-5C0h] BYREF
  RC4_CTX rc4; // [rsp+40h] [rbp-5B0h] BYREF
  uint8_t rc4_key[4]; // [rsp+150h] [rbp-4A0h] BYREF
  uint8_t chacha_nonce[12]; // [rsp+154h] [rbp-49Ch]
  uint8_t aes_key[16]; // [rsp+160h] [rbp-490h] BYREF
  uint8_t aes_iv[16]; // [rsp+170h] [rbp-480h] BYREF
  uint8_t chacha_key[32]; // [rsp+180h] [rbp-470h]
  uint8_t rc4_out[32]; // [rsp+1A0h] [rbp-450h] BYREF
  uint8_t aes_out[32]; // [rsp+1C0h] [rbp-430h] BYREF
  unsigned __int8 out[1032]; // [rsp+1E0h] [rbp-410h] BYREF
  unsigned __int64 v17; // [rsp+5E8h] [rbp-8h]

  v17 = __readfsqword(0x28u);
  printf(" Debug is important!!!.\n");
  printf("You can get the flag directly by debugging\n");
  printf("The debugging process is similar to that of Windows\n");
  printf("HINT: You can find the flag character style in the function x0r(), break on it\n");
  hexchar2int(c);
  *(_QWORD *)chacha_key = 0xA9A8A7A6A5A4A3A2LL;
  *(_QWORD *)&chacha_key[8] = 0xC2C1B6B5B9B5B3B1LL;
  *(_QWORD *)&chacha_key[16] = 0;
  *(_QWORD *)&chacha_key[24] = 0;
  *(_QWORD *)chacha_nonce = 0x69D4D4DDDD4D2D5LL;
  *(_DWORD *)&chacha_nonce[8] = 0;
  strcpy((char *)rc4_key, "Wow");
  keylen = strlen((const char *)rc4_key);
  rc4_init(&rc4, rc4_key, keylen);
  rc4_crypt(&rc4, ciphertext, rc4_out, 32);
  *(_QWORD *)aes_key = 0xA6D2AE2816157E2BLL;
  *(_QWORD *)&aes_key[8] = 0x1141467597F7ABLL;
  *(_QWORD *)aes_iv = 0x4511144511144511LL;
  *(_QWORD *)&aes_iv[8] = 0x1114451114451114LL;
  aes_de(aes_out, rc4_out, 32, aes_key, aes_iv);
  memcpy(out, aes_out, 0x20u);
  v[0] = 1751720303;
  v[1] = 1633904993;
  k[0] = 1;
  k[1] = 2;
  k[2] = 3;
00002774 main:1 (2774)
```

2,gdb调试得出关键逻辑代码

```
(gdb) si x0r (cipher=0x7fffffffd798 "\260\323<\227\253\362{y\001",
len=11202057548) at plzdebugme_linux.c:376 376 in plzdebugme_linux.c (gdb)
info registers rip rdi rip 0x555555556687 0x555555556687 <x0r> rdi
0x7fffffffd950 140737488345424 (gdb) finish Run till exit from #0 x0r
(cipher=0x7fffffffd798 "\260\323<\227\253\362{y\001", len=11202057548) at
plzdebugme_linux.c:376 Breakpoint 1, x0r ( cipher=0x7fffffffd950
"7=06*\030%b\016\025b3$6\026\016\005``<4p\016=4b\016!=0(,\260\332\377\377\377\
177", len=32)` at plzdebugme_linux.c:377 377 in plzdebugme_linux.c (gdb) info
registers rdi rdi 0x7fffffffd950 140737488345424 (gdb) x/32c $rdi
0x7fffffffd950: 55 '7' 61 '=' 48 '0' 54 '6' 42 '*' 24 '\030' 37 '%' 98 'b'
0x7fffffffd958: 14 '\016' 21 '\025' 98 'b' 51 '3' 36 '$' 54 '6' 22 '\026' 14
'\016' 0x7fffffffd960: 5 '\005' 96 '`' 96 '`' 60 '<' 52 '4' 112 'p' 14 '\016'
61 '=' 0x7fffffffd968: 52 '4' 98 'b' 14 '\016' 33 '!' 61 '=' 48 '0' 40 '(' 44
```

```
',' (gdb) printf "%.*s\n", 32, (char*) $rdi `*' not supported for precision or
width in printf` (gdb) x/32c $rbp-0x410 printf "%.*s\n", 32, (char*)($rbp-
0x410) A syntax error in expression, near `printf "%.*s\n", 32, (char*)($rbp-
0x410)'.` (gdb)
```

3,解题脚本如下

```
python3 - <<'PY'
data =
bytes.fromhex("373d30362a1825620e1562332436160e0560603c34700e3d34620e213d30282
c")
plain = bytes(b ^ 0x51 for b in data)
print(plain.decode())
PY
```

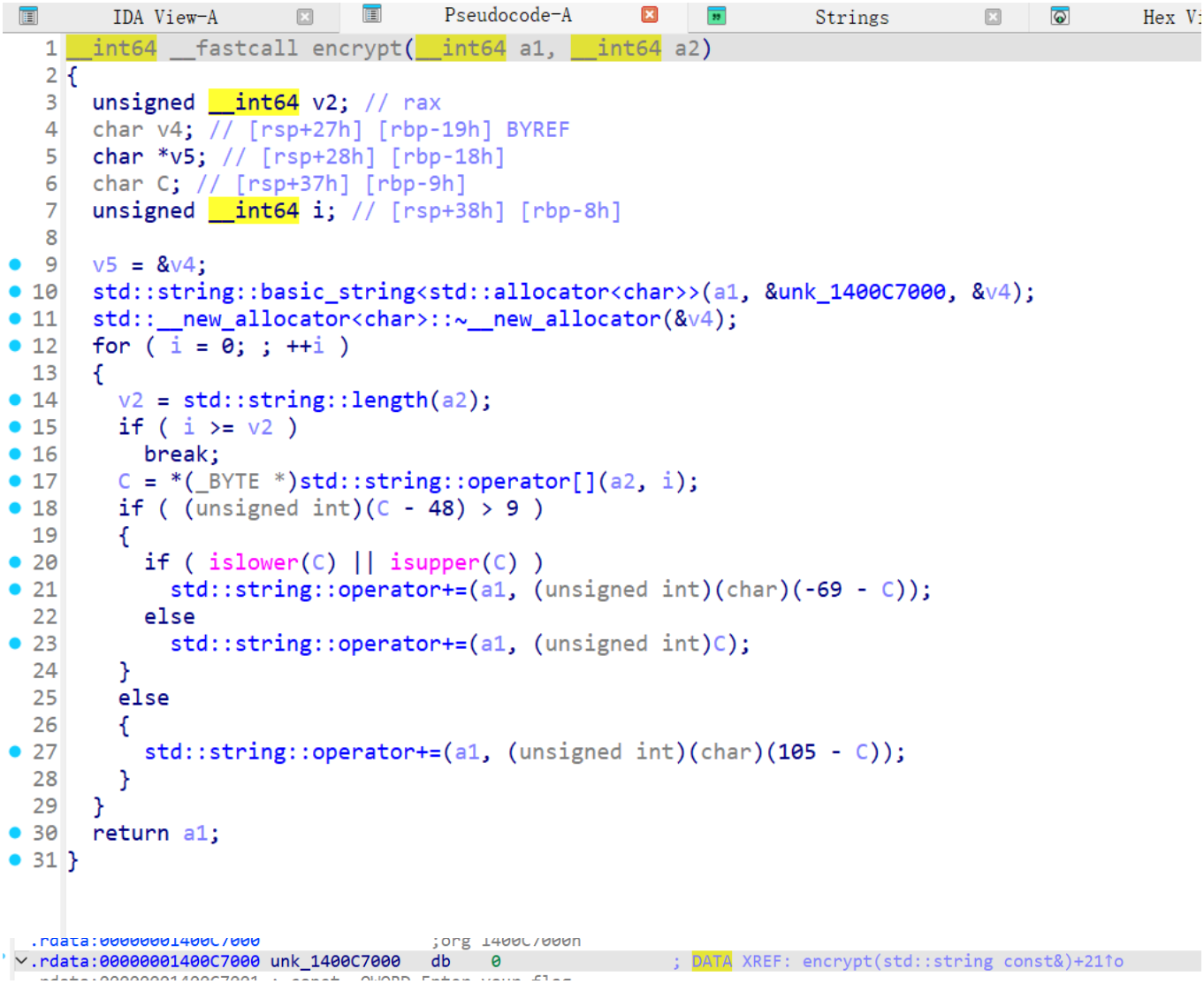
4,得出flag

```
flag{It3_D3bugG_T11me!_le3_play}
```

六,尤皮·埃克斯历险记（1）

1,die查看有壳,用upx去壳后ida分析

```
10  _QWORD v10[4]; // [rsp+60h] [rbp-40h] BYREF
11  __int16 n16753; // [rsp+80h] [rbp-20h]
12  char v12; // [rsp+86h] [rbp-1Ah]
13  char v13; // [rsp+87h] [rbp-19h]
14  __int64 n34; // [rsp+88h] [rbp-18h]
15  unsigned __int64 i; // [rsp+90h] [rbp-10h]
16  char v16; // [rsp+9Fh] [rbp-1h]
17
18  _main(*(__int64 *)&argc, (__int64)argv, (__int64)envp);
19  qmemcpy(v10, "isfhGJ\tt~cU\ny\nuTjcj\tT~cj", 24);
20  v10[3] = 0x5047B777E756451LL;
21  n16753 = 16753;
22  n34 = 34;
23  std::string::basic_string(v9);
24  std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, "Enter your flag: ");
25  std::operator>><char>(refptr__ZSt3cin, v9);
26  encrypt((__int64)v8, (__int64)v9);
27  n34_1 = std::string::length(v8);
28  if ( n34 == n34_1 )
29  {
30    v16 = 1;
31    for ( i = 0; ; ++i )
32    {
33      v5 = std::string::length(v8);
34      if ( i >= v5 )
35        break;
36      if ( IsDebuggerPresent() )
37      {
38        v12 = *(_BYTE *)std::string::operator[](v8, i) ^ 0xC3;
39        if ( v12 != *((_BYTE *)v10 + i) )
40        {
41          v16 = 0;
42          break;
43        }
44      }
45      else
46      {
47        v13 = *(_BYTE *)std::string::operator[](v8, i) ^ 0x3C;
48        if ( v13 != *((_BYTE *)v10 + i) )
49        {
50          v16 = 0;
51          break;
52        }
53      }
00000BB9 main:10 (1400015B9)
```



2,打开encrypt里面是异或加密,脚本如下

```
# reconstruct expected bytes used for comparison in main
prefix = b"isfhGJ\tt~cU\ny\nuTjcj\tT~cj"          # 24 bytes literal from
qmemcpy
v10_3 = (0x5047B777E756451).to_bytes(8, 'little')  # the 8-byte QWORD assigned
to v10[3]
n16753 = (16753).to_bytes(2, 'little')             # 0x4171 -> bytes 0x71 0x41

v10_full = prefix + v10_3 + n16753   # total 34 bytes

def recover(v10_bytes, debugger=False):
    xor_key = 0xC3 if debugger else 0x3C
    out = []
    for b in v10_bytes:
        enc = b ^ xor_key   # this is what encrypt produced (out)
        # inverse of encrypt:
        # if out corresponds to digit: out = 105 - C  => C = 105 - out
        pd = 105 - enc
```

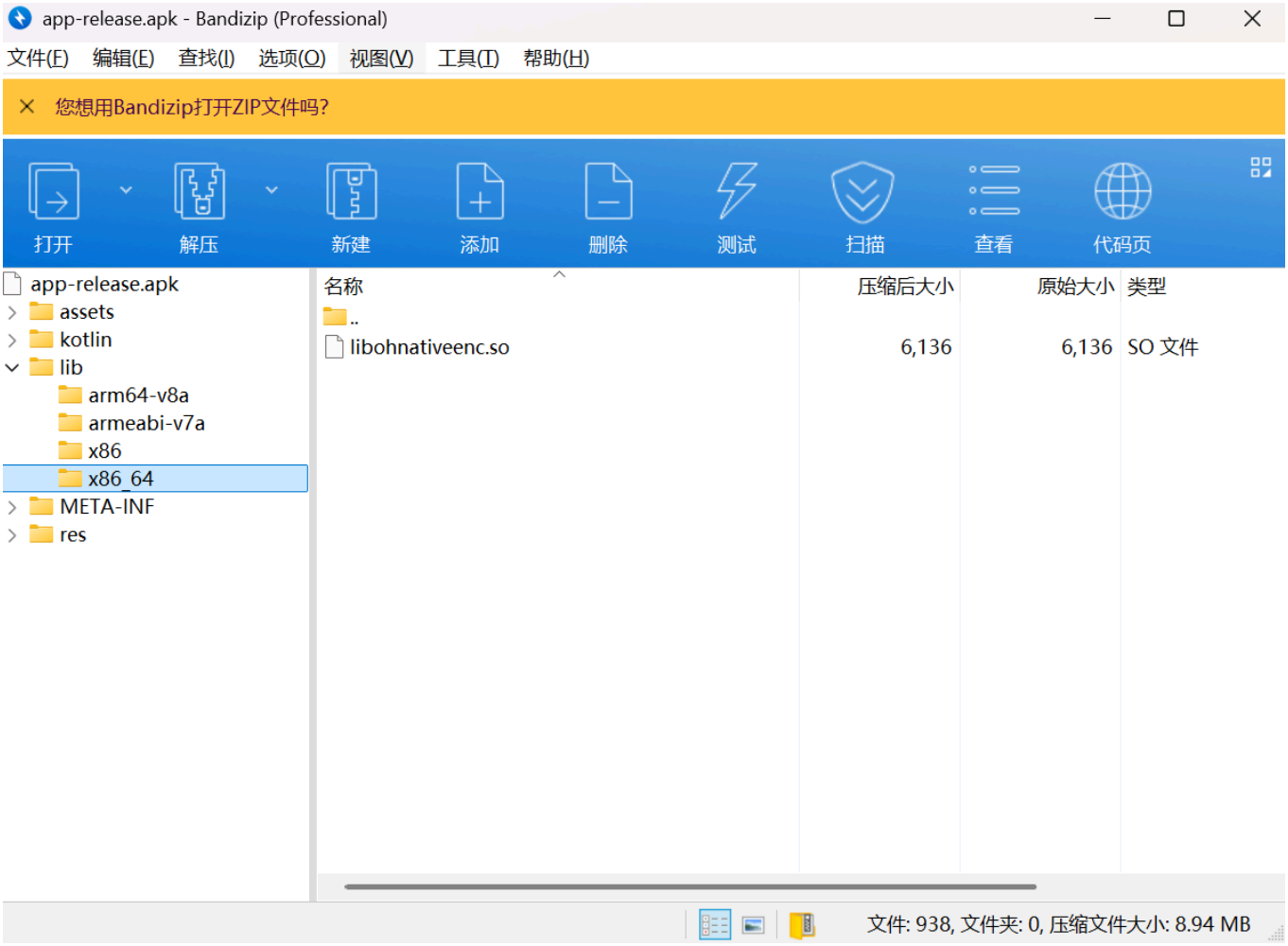
```
        if 48 <= pd <= 57:
            out.append(chr(pd)); continue
        # if out corresponds to letter: out = (char)(-69 - C) => C = (-69 -
out) mod 256
        pl = (-69 - enc) & 0xFF
        if (65 <= pl <= 90) or (97 <= pl <= 122):
            out.append(chr(pl)); continue
        # otherwise symbol unchanged
        out.append(chr(enc))
    return ''.join(out)

print("v10 bytes hex:", v10_full.hex())
print("Recovered (non-debug) flag:", recover(v10_full, debugger=False))
print("Recovered (debug) flag  :", recover(v10_full, debugger=True))
```

4,得出flag

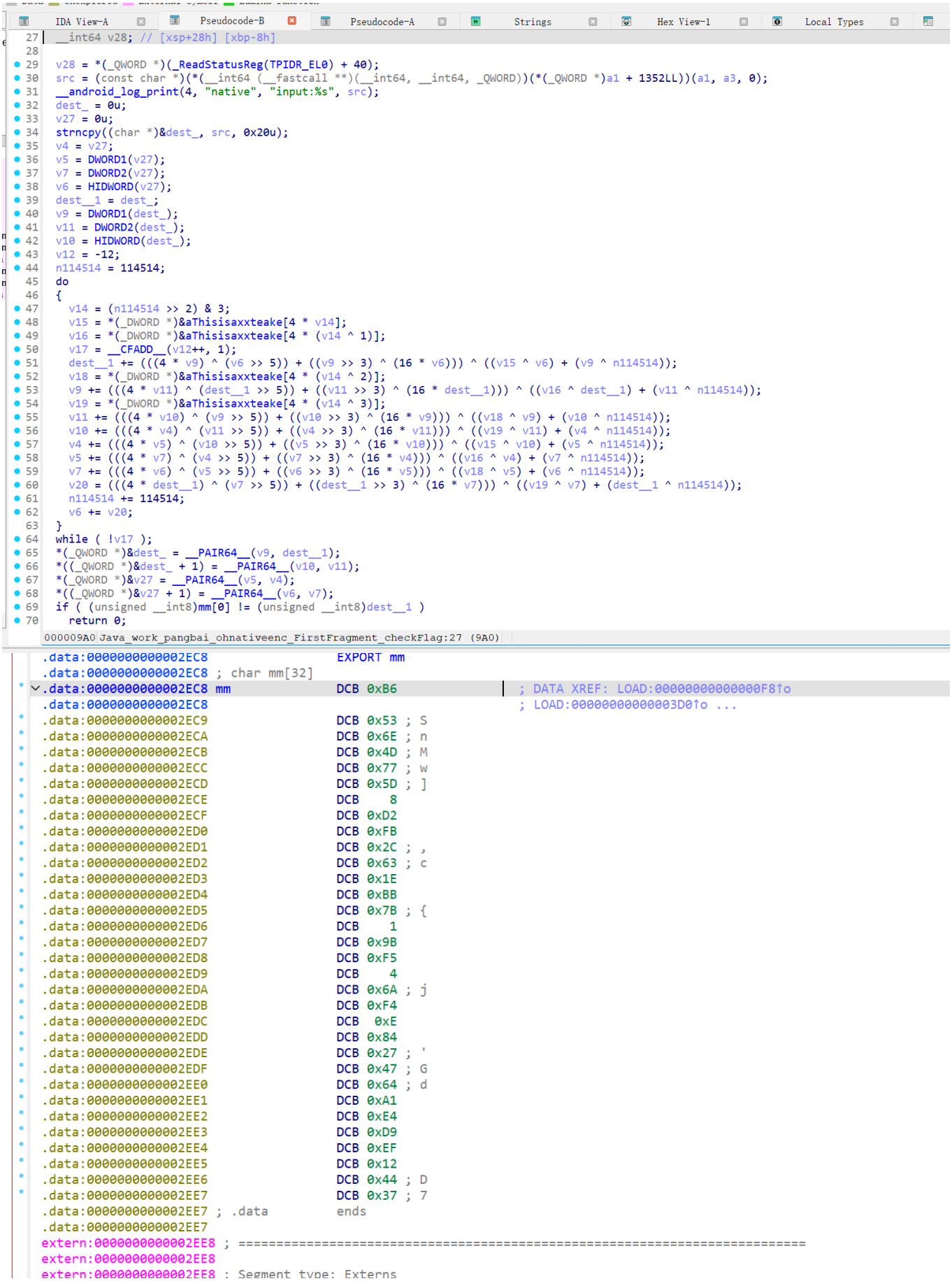
```
flag{E4sy_R3v3rSe_e4Sy_eNcrypt10n}
```

七,OhNativeEnc

1,还是安卓逆向,这个提示“安卓的 native 代码在哪呢”是在引导去找 C/C++ 写的代码，而不是 JADX 显示的 Java/Kotlin 代码,并且安卓的 Native 代码通常被编译成 `.so` (Shared Object) 文件，JADX 并不擅长直接分析它们,由于apk本质是压缩包,于是用bandzip打开它,找到其中的so文件并提取出来;

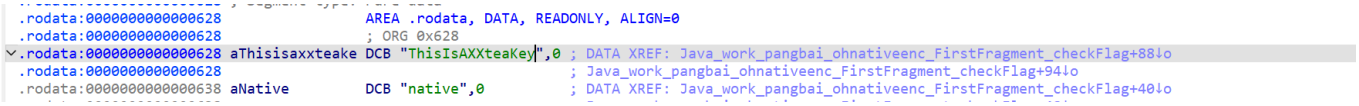


2,这是提取出文件的伪代码和其中的密文与密钥



```
__int64 v28; // [xsp+28h] [xbp-8h]

v28 = *(_QWORD *)(_ReadStatusReg(TPIDR_EL0) + 40);
src = (const char *)(*(__int64 (__fastcall **)(__int64, __int64, _QWORD))(*(_QWORD *)a1 + 1352LL))(a1, a3, 0);
__android_log_print(4, "native", "input:%s", src);
dest_ = 0u;
v27 = 0u;
strncpy((char *)&dest_, src, 0x20u);
v4 = v27;
v5 = DWORD1(v27);
v7 = DWORD2(v27);
v6 = HIDWORD(v27);
dest__1 = dest_;
v9 = DWORD1(dest_);
v11 = DWORD2(dest_);
v10 = HIDWORD(dest_);
v12 = -12;
n114514 = 114514;
do
{
  v14 = (n114514 >> 2) & 3;
  v15 = *(_DWORD *)&aThisisaxxteake[4 * v14];
  v16 = *(_DWORD *)&aThisisaxxteake[4 * (v14 ^ 1)];
  v17 = __CFADD__(v12++, 1);
  dest__1 += (((4 * v9) ^ (v6 >> 5)) + ((v9 >> 3) ^ (16 * v6))) ^ ((v15 ^ v6) + (v9 ^ n114514));
  v18 = *(_DWORD *)&aThisisaxxteake[4 * (v14 ^ 2)];
  v9 += (((4 * v11) ^ (dest__1 >> 5)) + ((v11 >> 3) ^ (16 * dest__1))) ^ ((v16 ^ dest__1) + (v11 ^ n114514));
  v19 = *(_DWORD *)&aThisisaxxteake[4 * (v14 ^ 3)];
  v11 += (((4 * v10) ^ (v9 >> 5)) + ((v10 >> 3) ^ (16 * v9))) ^ ((v18 ^ v9) + (v10 ^ n114514));
  v10 += (((4 * v4) ^ (v11 >> 5)) + ((v4 >> 3) ^ (16 * v11))) ^ ((v19 ^ v11) + (v4 ^ n114514));
  v4 += (((4 * v5) ^ (v10 >> 5)) + ((v5 >> 3) ^ (16 * v10))) ^ ((v15 ^ v10) + (v5 ^ n114514));
  v5 += (((4 * v7) ^ (v4 >> 5)) + ((v7 >> 3) ^ (16 * v4))) ^ ((v16 ^ v4) + (v7 ^ n114514));
  v7 += (((4 * v6) ^ (v5 >> 5)) + ((v6 >> 3) ^ (16 * v5))) ^ ((v18 ^ v5) + (v6 ^ n114514));
  v20 = (((4 * dest__1) ^ (v7 >> 5)) + ((dest__1 >> 3) ^ (16 * v7))) ^ ((v19 ^ v7) + (dest__1 ^ n114514));
  n114514 += 114514;
  v6 += v20;
}
while ( !v17 );
*(_QWORD *)&dest_ = __PAIR64__(v9, dest__1);
*((_QWORD *)&dest_ + 1) = __PAIR64__(v10, v11);
*(_QWORD *)&v27 = __PAIR64__(v5, v4);
*((_QWORD *)&v27 + 1) = __PAIR64__(v6, v7);
if ( (unsigned __int8)mm[0] != (unsigned __int8)dest__1 )
  return 0;
```

```
.data:0000000000002EC8                 EXPORT mm
.data:0000000000002EC8 ; char mm[32]
.data:0000000000002EC8 mm              DCB 0xB6              ; DATA XREF: LOAD:00000000000000F8↑o
.data:0000000000002EC8                                       ; LOAD:00000000000003D0↑o ...
.data:0000000000002EC9                 DCB 0x53 ; S
.data:0000000000002ECA                 DCB 0x6E ; n
.data:0000000000002ECB                 DCB 0x4D ; M
.data:0000000000002ECC                 DCB 0x77 ; w
.data:0000000000002ECD                 DCB 0x5D ; ]
.data:0000000000002ECE                 DCB    8
.data:0000000000002ECF                 DCB 0xD2
.data:0000000000002ED0                 DCB 0xFB
.data:0000000000002ED1                 DCB 0x2C ; ,
.data:0000000000002ED2                 DCB 0x63 ; c
.data:0000000000002ED3                 DCB 0x1E
.data:0000000000002ED4                 DCB 0xBB
.data:0000000000002ED5                 DCB 0x7B ; {
.data:0000000000002ED6                 DCB    1
.data:0000000000002ED7                 DCB 0x9B
.data:0000000000002ED8                 DCB 0xF5
.data:0000000000002ED9                 DCB    4
.data:0000000000002EDA                 DCB 0x6A ; j
.data:0000000000002EDB                 DCB 0xF4
.data:0000000000002EDC                 DCB  0xE
.data:0000000000002EDD                 DCB 0x84
.data:0000000000002EDE                 DCB 0x27 ; '
.data:0000000000002EDF                 DCB 0x47 ; G
.data:0000000000002EE0                 DCB 0x64 ; d
.data:0000000000002EE1                 DCB 0xA1
.data:0000000000002EE2                 DCB 0xE4
.data:0000000000002EE3                 DCB 0xD9
.data:0000000000002EE4                 DCB 0xEF
.data:0000000000002EE5                 DCB 0x12
.data:0000000000002EE6                 DCB 0x44 ; D
.data:0000000000002EE7                 DCB 0x37 ; 7
.data:0000000000002EE7 ; .data         ends
.data:0000000000002EE7
extern:0000000000002EE8 ; ===========================================================================
extern:0000000000002EE8
extern:0000000000002EE8 ; Segment type: Externs
```



3,脚本如下

```
# 可复现脚本：把 mm_bytes、key_bytes 按你环境运行
import struct

def u32(x): return x & 0xFFFFFFFF

def F(y, z, k, n):
    return u32(((((4*y) ^ (z >> 5)) + ((y >> 3) ^ ((16 * z) & 0xFFFFFFFF))) &
0xFFFFFFFF) ^ ((k ^ z) + (y ^ n)))

def decrypt(mm_bytes, key_bytes, rounds=12):
    key = list(struct.unpack("<4I", key_bytes))
    v = list(struct.unpack("<8I", mm_bytes))
    delta = 114514
    for k in range(rounds, 0, -1):
        n = u32(delta * k)
        v14 = (n >> 2) & 3
        k0 = key[v14]; k1 = key[v14 ^ 1]; k2 = key[v14 ^ 2]; k3 = key[v14 ^ 3]
        # 逆序做减法（与伪代码加法相反）
        v20 = F(v[0], v[6], k3, n); v[7] = u32(v[7] - v20)
        t = F(v[7], v[5], k2, n); v[6] = u32(v[6] - t)
        t = F(v[6], v[4], k1, n); v[5] = u32(v[5] - t)
        t = F(v[5], v[3], k0, n); v[4] = u32(v[4] - t)
        t = F(v[4], v[2], k3, n); v[3] = u32(v[3] - t)
        t = F(v[3], v[1], k2, n); v[2] = u32(v[2] - t)
        t = F(v[2], v[0], k1, n); v[1] = u32(v[1] - t)
        t = F(v[1], v[7], k0, n); v[0] = u32(v[0] - t)
    return struct.pack("<8I", *v)

if __name__ == "__main__":
    # 从 IDA 拷出的 mm（32 bytes），按你贴出的顺序
    mm_bytes = bytes([
        0xB6,0x53,0x6E,0x4D,0x77,0x5D,0x08,0xD2,
        0xFB,0x2C,0x63,0x1E,0xBB,0x7B,0x01,0x9B,
        0xF5,0x04,0x6A,0xF4,0x0E,0x84,0x27,0x47,
        0x64,0xA1,0xE4,0xD9,0xEF,0x12,0x44,0x37
    ])

    # key = "ThisIsAXXteaKey",0
    key_bytes = b"ThisIsAXXteaKey\x00"

    plain = decrypt(mm_bytes, key_bytes, rounds=12)
```

```
    print("raw bytes:", plain)
    try:
        print("as utf-8:", plain.rstrip(b"\x00").decode('utf-8'))
    except:
        print("as latin-1:", plain.decode('latin-1'))
```

4,得出flag

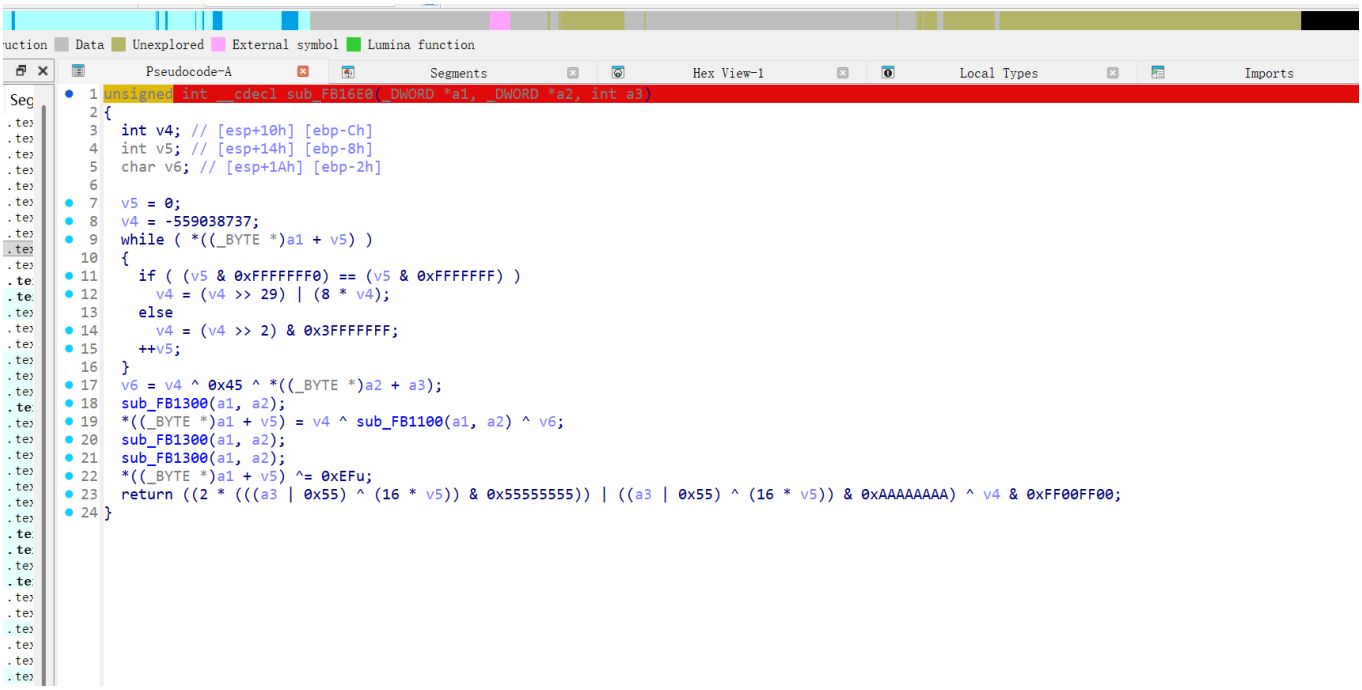
```
flag{Ur_G00d_@_n@tive_Func}
```

八,Look at me carefully

1,ida先查看源代码

Pseudocode-A | Segments | Hex View-1

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char *v4; // [esp+18h] [ebp-64h]
  char cH4_1elo_ookte_0dv__alafle___5yygume[40]; // [esp+1Ch] [ebp-60h] BYREF
  char v6[52]; // [esp+44h] [ebp-38h] BYREF

  strcpy(cH4_1elo_ookte_0dv__alafle___5yygume, "cH4_1elo{ookte?0dv_}alafle___5yygume");
  memset(v6, 0, 0x32u);
  v4 = (char *)calloc(0x25u, 1u);
  sub_FB1050(Format);                           // "Please enter the flag: "
  sub_FB10C0("%s", v6);                         // "%s"
  if ( strlen(v6) == 36 )
  {
    sub_FB16E0(v4, v6, 27);
    sub_FB16E0(v4, v6, 5);
    sub_FB16E0(v4, v6, 6);
    sub_FB16E0(v4, v6, 9);
    sub_FB16E0(v4, v6, 28);
    sub_FB16E0(v4, v6, 18);
    sub_FB16E0(v4, v6, 32);
    sub_FB16E0(v4, v6, 29);
    sub_FB16E0(v4, v6, 4);
    sub_FB16E0(v4, v6, 11);
    sub_FB16E0(v4, v6, 15);
    sub_FB16E0(v4, v6, 17);
    sub_FB16E0(v4, v6, 22);
    sub_FB16E0(v4, v6, 8);
    sub_FB16E0(v4, v6, 34);
    sub_FB16E0(v4, v6, 16);
    sub_FB16E0(v4, v6, 19);
    sub_FB16E0(v4, v6, 7);
    sub_FB16E0(v4, v6, 26);
    sub_FB16E0(v4, v6, 35);
    sub_FB16E0(v4, v6, 2);
    sub_FB16E0(v4, v6, 14);
    sub_FB16E0(v4, v6, 21);
    sub_FB16E0(v4, v6, 0);
    sub_FB16E0(v4, v6, 1);
    sub_FB16E0(v4, v6, 25);
    sub_FB16E0(v4, v6, 13);
    sub_FB16E0(v4, v6, 23);
    sub_FB16E0(v4, v6, 20);
```



2,这个题是字符串重排的逻辑,需要动态调试,但是我调不出来,这个是ai的初始脚本

```
# 1. 这是程序内部存储的、重排后的正确字符串
target = "cH4_1elo{ookte?0dv_}alafle___5yygume"

# 2. 这是 sub_FB16E0 调用中使用的索引顺序
indices = [
    27, 5, 6, 9, 28, 18, 32, 29, 4, 11, 15, 17, 22, 8, 34, 16, 19, 7, 26, 35,
    2, 14, 21, 0, 1, 25, 13, 23, 20, 37, 30, 33, 10, 3, 12, 36, 24, 31
]

# 3. 创建一个 38 个字符长的空列表来存放 flag
#    (target 和 indices 列表的长度都是 38)
flag = ['?'] * len(target)

# 4. 执行反向操作:
#    我们知道 target[0] ( 'c' ) 来自 flag[27]
#    我们知道 target[1] ( 'H' ) 来自 flag[5]
#    ...
for i in range(len(target)):
    flag_index = indices[i]  # 应该存放的位置
    target_char = target[i]    # 应该存放的字符

    flag[flag_index] = target_char

# 5. 打印结果
print("".join(flag))
```

3,得出了不正确的flag

```
flau{H4ve_gom_lo0ked_at_?e_c1oy?ly?}e5
```

4,但是可以看出这是字符串移位,我将它手动拼成了可读的状态,得出了正确flag

```
flag{H4ve_you_lo0ked_at_me_c1o5ely?}
```

九,Forgotten_Code

1,这个题是文本文件/汇编语言代码,可以直接在txt中分析,也可以gcc转化为exe放在ida中分析,在txt中发现了main,进行查看

```
        .seh_proc  main
main:
.LFB189:
        push rbp
        .seh_pushreg    rbp
        mov rbp, rsp
        .seh_setframe    rbp, 0
        sub   rsp, 144
        .seh_stackalloc  144
        .seh_endprologue
        call      main
        lea    rax, .LC0[rip]
        mov rcx, rax
        call     Z6printfPKcz
        lea    rax, -112[rbp]
        lea    rcx, .LC1[rip]
        mov rdx, rax
        call     Z5scanfPKcz
        lea    rdx, .LC2[rip]
        lea    rax, -112[rbp]
        mov r8d, 5
        mov rcx, rax
        call   strncmp
        test   eax, eax
        jne    .L11
        lea    rax, -112[rbp]
        mov rcx, rax
        call   strlen
        sub   rax, 1
        movzx      eax, BYTE PTR -112[rbp+rax]
```

2,发现是用二进制用 TEA 算法对每个 8 字节块加密，密钥在每次调用 `fn` 时在 `ng` 上来回 XOR

`0x11`,脚本如下
#!/usr/bin/env python3

decrypt_tea.py

Reproduce TEA decryption and key toggling from the disassembly, then print recovered flag.

from struct import pack, unpack

def u32(x):
return x & 0xFFFFFFFF

def bytes_to_u32_list(b):
# little-endian -> list of uint32
return [unpack('<I', b[i:i+4])[0] for i in range(0, len(b), 4)]

def tea*decrypt(v0, v1, k):*
"""Standard TEA decrypt: v0,v1 are uint32, k is list of 4 uint32"""
delta = 0x9E3779B9
mask = 0xFFFFFFFF
sum = (delta * 32) & mask
v0 = u32(v0); v1 = u32(v1)
k0,k1,k2,k3 = [u32(x) for x in k]
for *in range(32):*
v1 = u32(v1 - ((((v0 << 4) + k2) ^ (v0 + sum) ^ ((v0 >> 5) + k3))))
v0 = u32(v0 - ((((v1 << 4) + k0) ^ (v1 + sum*) ^ ((v1 >> 5) + k1))))*
sum = u32(sum_ - delta)
return v0, v1

def main():
# ezgm dwords extracted from the binary (as signed in disasm); convert to u32
ezgm = [
1210405119, 710975774,
-90350153, -1958008304,
-745722482, 67707510,
-86515270, -1728462407
]
ezgm_u = [u32(x) for x in ezgm]

```
# original ng bytes from disassembly: "sp\177vuctp|xeb|hv~"
ng_orig = b"sp\x7fvuctp|xeb|hv~"  # 16 bytes
```

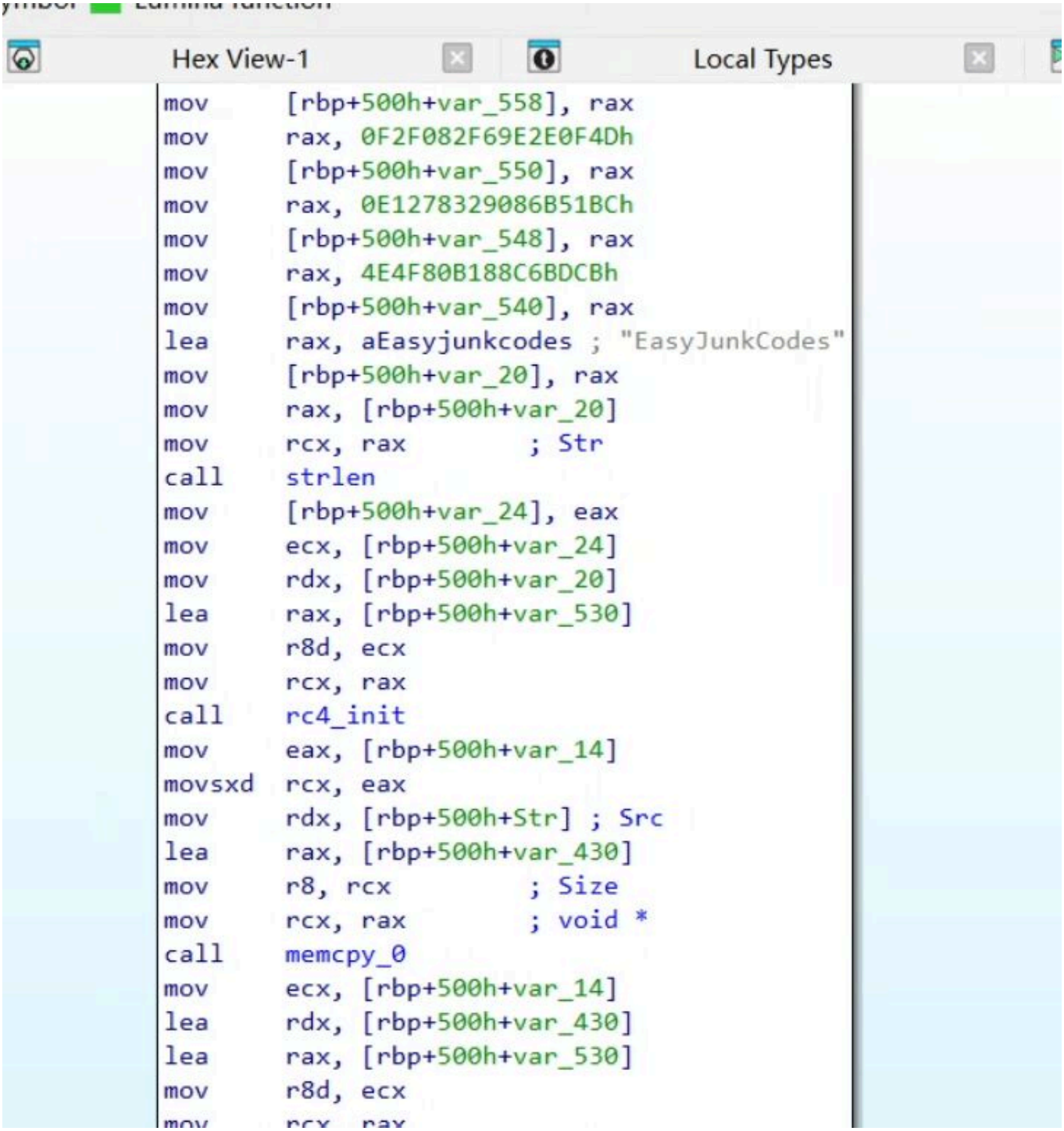
```
# key0 = ng_orig interpreted as 4 little-endian uint32
key0 = bytes_to_u32_list(ng_orig)
# key1 = ng_orig XOR 0x11 on each byte, then interpreted
ng_xored = bytes(byte ^ 0x11 for byte in ng_orig)
key1 = bytes_to_u32_list(ng_xored)

# There are 4 blocks of 8 bytes (32 bytes payload). For block i:
# encryption used key1 for even i (i=0 -> key1), key0 for odd i (matching
disasm).
plaintext_blocks = []
for i in range(4):
    key = key1 if (i % 2 == 0) else key0
    c0 = ezgm_u[2*i]
    c1 = ezgm_u[2*i + 1]
    p0, p1 = tea_decrypt(c0, c1, key)
    plaintext_blocks.append(pack('<II', p0, p1))

payload = b''.join(plaintext_blocks)[:32]
flag = b"flag{" + payload + b"}"

print("Recovered payload (hex):", payload.hex())
# print payload safely (latin1 to avoid decode errors)
print("Recovered payload (latin1):", payload.decode('latin1'))
print("\nRecovered flag:")
print(flag.decode('latin1'))
```

if **name** == "**main**":
main()
3,得出flag

```
flag{4553m81y_5_s0o0o0_345y_jD5yQ5mD9}
```

十,采一朵花，送给艾达（1）

1,这个题无法用伪代码分析了,只能在asm层面查看,观察出是rc4加密



2,但是rc4加密的算法被作者魔改了

```
__int64 __fastcall rc4_crypt(__int64 a1, __int64 a2, int i_2)
{
  __int64 i_1; // rax
  char v4; // [rsp+3h] [rbp-Dh]
  unsigned int i; // [rsp+4h] [rbp-Ch]
  int v6; // [rsp+8h] [rbp-8h]
  int v7; // [rsp+Ch] [rbp-4h]

  v7 = 0;
  v6 = 0;
  for ( i = 0; ; ++i )
  {
    i_1 = i;
    if ( (int)i >= i_2 )
      break;
    v7 = (v7 + 1) % 256;
    v6 = (*(unsigned __int8 *)(a1 + v7) + v6) % 256;
    v4 = *(_BYTE *)(a1 + v7);
    *(_BYTE *)(a1 + v7) = *(_BYTE *)(a1 + v6);
    *(_BYTE *)(v6 + a1) = v4;
    *(_BYTE *)((int)i + a2) = *(_BYTE *)(a2 + *(unsigned __int8 *)(a1 + v7)
                                           + (unsigned int)*(unsigned __int8 *)(a1 + v6))
                            + *(_BYTE *)(a1 + (unsigned __int8)(*(_BYTE *)(a1 + v7) + *(_BYTE *)(a1 + v6)));
  }
  return i_1;
}
```

3,以"EasyJunkCodes"为key，解密数据"C77FC1430364751188B88C55F6C023DF4D0F2E9EF682F0F2BC516B08298327E1CBBDC688B1804F4E",解题脚本如下

```
def rc4_init(key: bytes) -> list:
    """
    初始化魔改RC4的S-box。
    """
    key_len = len(key)
    # 1. S-box的魔改初始化
    s_box = [(256 - i) % 256 for i in range(256)]

    # 2. 标准的KSA置换过程
    j = 0
    for i in range(256):
        j = (j + s_box[i] + key[i % key_len]) % 256
        s_box[i], s_box[j] = s_box[j], s_box[i]

    return s_box

def rc4_decrypt(s_box_init: list, ciphertext: bytes) -> bytes:
    """
    使用初始化后的S-box解密数据。
    """
    # 复制S-box，因为解密过程会修改它的状态
    s_box = list(s_box_init)

    i = 0
    j = 0
    plaintext = bytearray()
```

```
    for char_code in ciphertext:
        # PRGA: 生成密钥流
        i = (i + 1) % 256
        j = (j + s_box[i]) % 256
        s_box[i], s_box[j] = s_box[j], s_box[i]

        # 计算密钥流字节 K
        keystream_byte = s_box[(s_box[i] + s_box[j]) % 256]

        # 解密操作: P = (C - K) % 256
        # 加上256再取模是为了确保结果为正
        decrypted_byte = (char_code - keystream_byte + 256) % 256
        plaintext.append(decrypted_byte)

    return bytes(plaintext)

# --- 主程序 ---
if __name__ == "__main__":
    # 输入的Key和密文
    key = b"EasyJunkCodes"
    ciphertext_hex =
"C77FC1430364751188B88C55F6C023DF4D0F2E9EF682F0F2BC516B08298327E1CBBDC688B1804
F4E"

    # 将十六进制字符串转换为字节
    ciphertext = bytes.fromhex(ciphertext_hex)

    print(f"Key: {key.decode()}")
    print(f"Ciphertext (hex): {ciphertext_hex}")
    print("-" * 30)

    # 1. 初始化S-box
    s_box_initialized = rc4_init(key)

    # 2. 解密数据
    plaintext = rc4_decrypt(s_box_initialized, ciphertext)

    # 3. 输出结果
    try:
        print(f"Decrypted Flag: {plaintext.decode('utf-8')}")
    except UnicodeDecodeError:
        print(f"Decrypted Data (bytes): {plaintext}")
```

4,得出flag

```
flag{Junk_C0d3s_4Re_345y_t0_rEc0gn1Ze!!}
```