HGAME 2022 Week1 writeup by sleeper

web

easy_auth

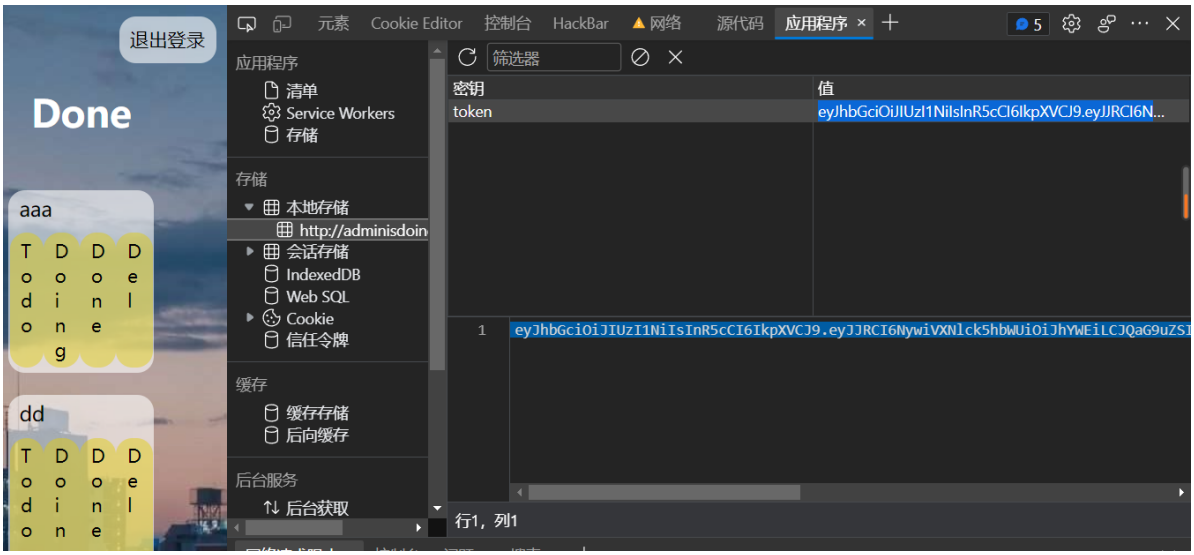
根据提示，应该是权限验证相关。先注册个账号进入，找到网页里的 /app.js，发现大多数函数都使用了 token (`localStorage.getItem("token")`)

```
function putData(url, data) {
    return fetch(baseUrl + url, {
        body: JSON.stringify(data),
        headers: {
            'content-type': 'application/json',
            token: localStorage.getItem("token")
        },
        method: 'PUT',
        mode: "cors",
        // credentials: "include"
    }).then((response) => {
        // console.log(response);
        if (response.status === 403) {
            window.location.href = "/login.html"
        }
        return response
    }).then(response => response.json())
}
```

```
19
```

结合 bp 抓包也只有 token 一个可疑项，尝试对 token 进行 b64 解码

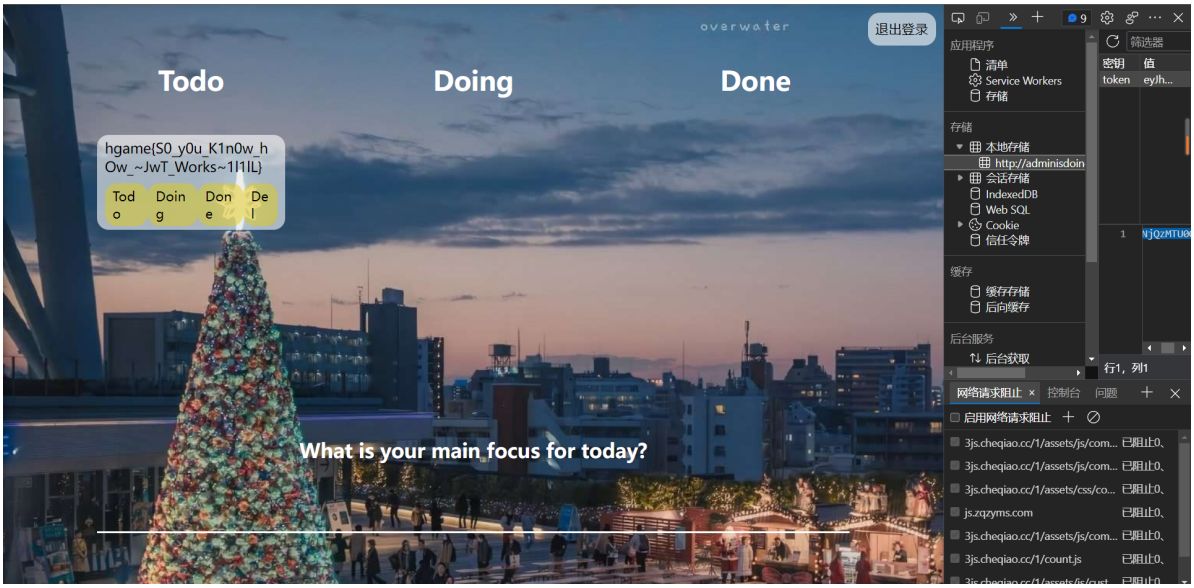
这里从 浏览器开发者工具里“应用程序” 模块进行 token 的复制和修改。



解码结果

可以确定为 JWT 伪造，进入官网 https://jwt.io，发现不设密码时验证通过。

修改用户名为 admin，ID 为 1，替换原 token 成功看到 admin 的 todo。

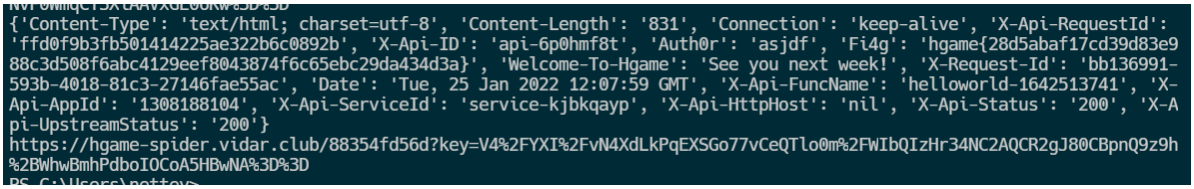


蛛蛛...嘿嘿♥我的蛛蛛

点击几次发现按钮越来越多，查看 html 源码发现只有一个按钮有效，写脚本遍历到最后一个网页

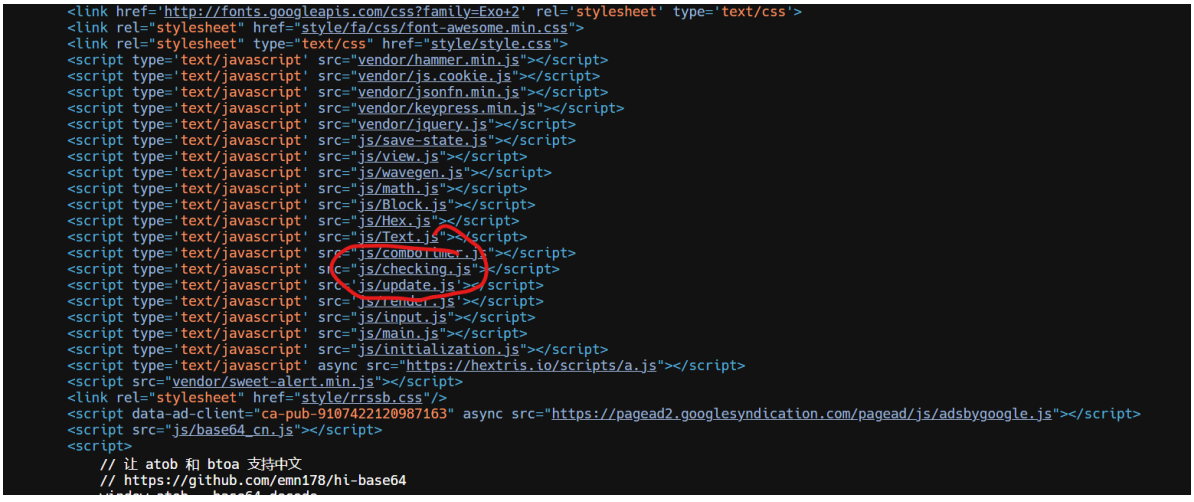
```python
import re
import requests

krym = re.compile('key=')

s = 'https://hgame-spider.vidar.club/88354fd56d'
x=""
def digme(x):
    req = requests.get(s+x,headers={'User-Agent':'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/97.0.4692.71 Safari/537.36 Edg/97.0.1072.62'})
    print(req.headers)
    print(s+x)
    xt = [x for x in req.text.split("\"") if 'key' in x]
    for i in xt:
        deepme(i)
    if "hgame" in req.text:
        print(req.text)
        return

digme(x)
```

当然，flag 在最后一个网页的 header 里，其实脚本已经打印了 header 了，不过眼瞎没发现，如图，Fi4g 段即为 flag



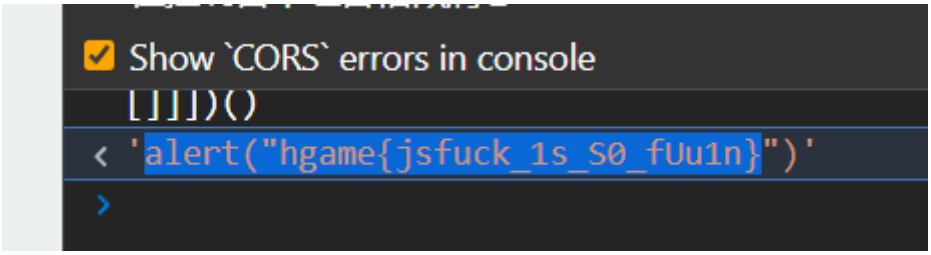
Tetris plus

玩游戏，ctrl+u 查看 html 源代码



感觉 checking.js 比较可疑，在最下面发现 winned 和 3000 字样，基本确定为 flag 所在

最下面的注释叫做 jsfuck，brainfuck 的变种，直接作为 js 代码在 console 里运行即可。

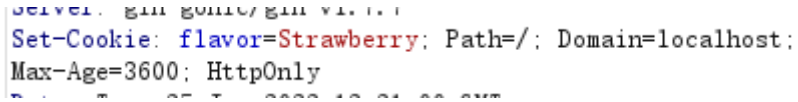


Fujiwara Tofu Shop

http 协议的传统套娃题。

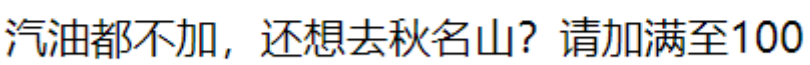
前两个直接设 referer 和 UA 为 qiumingshan.net 与 Hachi-Roku，不多说。

主要是设 cookies 的部分，看请求发现服务器返回了 set-cookie



但是 cookie 的域名对的是 localhost，和 shop.summ3r.top 没对应，~~这里本文通过 bp 手动修改返回包使其对应~~

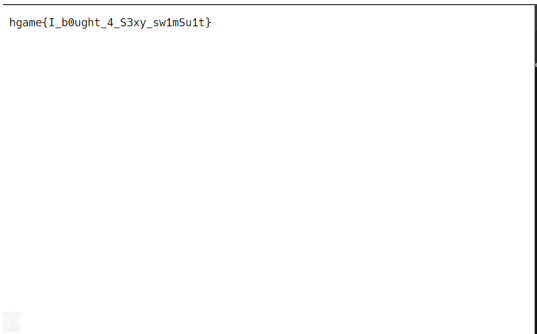
根据响应包，把 cookie 字段 对应内容手动设为 flavor=Raspberry。

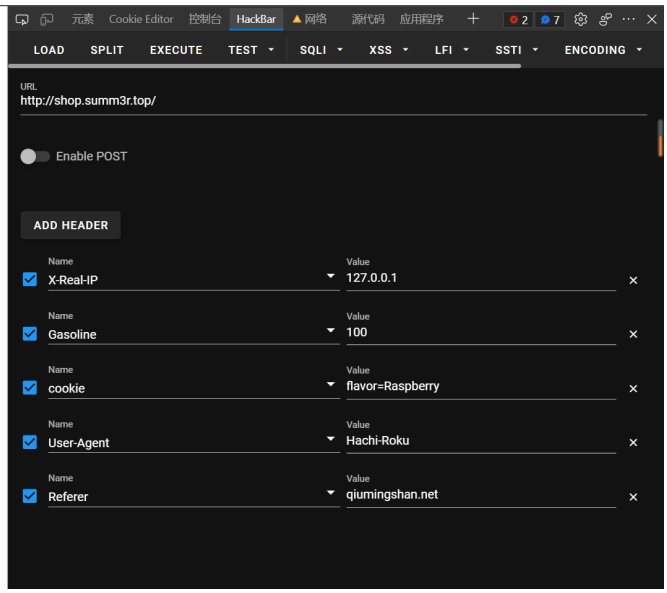


加汽油，发现响应头中出现了 `Gasoline: 0` 字样，改为 100。

最后需要本地请求，这里 *X*-Forwarded-For 不行，尝试 X-Real-IP 成功。

所有涉及的字段和最终 flag





CRYPTO

Dancing Line

盯着线半天，黑点个数有30多个（记不清了），猜测没两个黑点间的线段代表了一段数据，将向左，向下记为 0 与 1，

```python
from turtle import pen
from PIL import Image

import string

Im = Image.open('Dancing Line.bmp')
# 获取图片指定像素点的像素


def getcolor(IMG, pixelX=1, pixelY=1):
    img_src = IMG
    img_src = img_src.convert('RGB')
    str_strlist = img_src.load()
    data = str_strlist[pixelX, pixelY]
    img_src.close()
    return data


def getNextPix(IMG, pixelX=1, pixelY=1):
    assert(pixelX < Im.size[0])
    assert(pixelY < Im.size[1])
    goleft = 'err'
    next_X,next_Y = 0,0
    R, G, B = getcolor(IMG, pixelX, pixelY+1)
    if R == 255 and G == 255 and B == 255:
        goleft = '0'
        R, G, B = getcolor(IMG, pixelX+1, pixelY)
        next_X,next_Y = pixelX+1, pixelY
```

```
    else:
        goleft = '1'
        next_X,next_Y = pixelX, pixelY+1
    if R == 0 and G == 0 and B == 0:
        goleft += '-1'
    return(next_X,next_Y,goleft)


# 136 162
i, j = 0, 0

anl = []
tmp = ""

while i < Im.size[0]-1 or j < Im.size[1]-1:
    i, j, ll = getNextPix(Im, i, j)
    tmp = tmp + ll[0]
    if '-1' in ll:
        anl.append(tmp)
        tmp = ""

print("\n".join(anl))
ant = [int(x,2) for x in anl ]

for i in ant:
    print((chr(i)),end="")

'''
i g a m e { E a o c 1 o g _ M 1 o e _ 1 5 _ g u o - _ 1 5 o ' u _ 1 u ? }
  0 0 0 0 0   0   0 0   0 0   0   0 0 0 0 0 0 0     0 0 0     0 0   0 0
-1        -1  -1   -1   -1  -1           ?-1 ?     -1 0-1   -1
h g a m e { D a n c 1 n g _ L 1 n e _ 1 5 _ g a m e _ 1 5 n ' t _ 1 t ? }
'''

```

这里要将每个线段与最后一个线段相连的黑点也计入其中，否则就丢了一位，因为这一位搞了好久。

flag 在 注释里。

Easy RSA

对每一个字符都产生了一组 p q 进行加密，但是把所有 RSA 解密的信息都给了，所以只要写个解密脚本即可

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

def decrypt(enc):
    e, p, q, m = enc
    d = invert(e,(p - 1) * (q - 1))
    return chr(pow(m, d, p * q))

if __name__ == '__main__':
```

```
    enc = [(12433, 149, 197, 104), (8147, 131, 167, 6633), (10687, 211, 197,
35594), (19681, 131, 211, 15710), (33577, 251, 211, 38798), (30241, 157,
251, 35973), (293, 211, 157, 31548), (26459, 179, 149, 4778), (27479, 149,
223, 32728), (9029, 223, 137, 20696), (4649, 149, 151, 13418), (11783, 223,
251, 14239), (13537, 179, 137, 11702), (3835, 167, 139, 20051), (30983, 149,
227, 23928), (17581, 157, 131, 5855), (35381, 223, 179, 37774), (2357, 151,
223, 1849), (22649, 211, 229, 7348), (1151, 179, 223, 17982), (8431, 251,
163, 30226), (38501, 193, 211, 30559), (14549, 211, 151, 21143), (24781,
239, 241, 45604), (8051, 179, 131, 7994), (863, 181, 131, 11493), (1117,
239, 157, 12579), (7561, 149, 199, 8960), (19813, 239, 229, 53463), (4943,
131, 157, 14606), (29077, 191, 181, 33446), (18583, 211, 163, 31800),
(30643, 173, 191, 27293), (11617, 223, 251, 13448), (19051, 191, 151,
21676), (18367, 179, 157, 14139), (18861, 149, 191, 5139), (9581, 211, 193,
25595)]
    print(*list(map(decrypt, enc)))

```

```
hgame{L00ks_l1ke_y0u've_mastered_RS4!}
```

最后整理提供的 flag，

hgame{L00ks_l1ke_y0u've_mastered_RS4!}

Matryoshka

老套娃题了，解密摩斯密码的时候怎么弄都有不在对照表上的，通过最后的提示发现需要反转盲文，这里在 CyberChef 里放一下解密过程

Recipe

Substitute

Plaintext
.. : .

Ciphertext
- . /

Reverse

By
Character

From Morse Code

Letter delimiter
Forward slash

Word delimiter
Semi-colon

From Hex

Delimiter
Comma

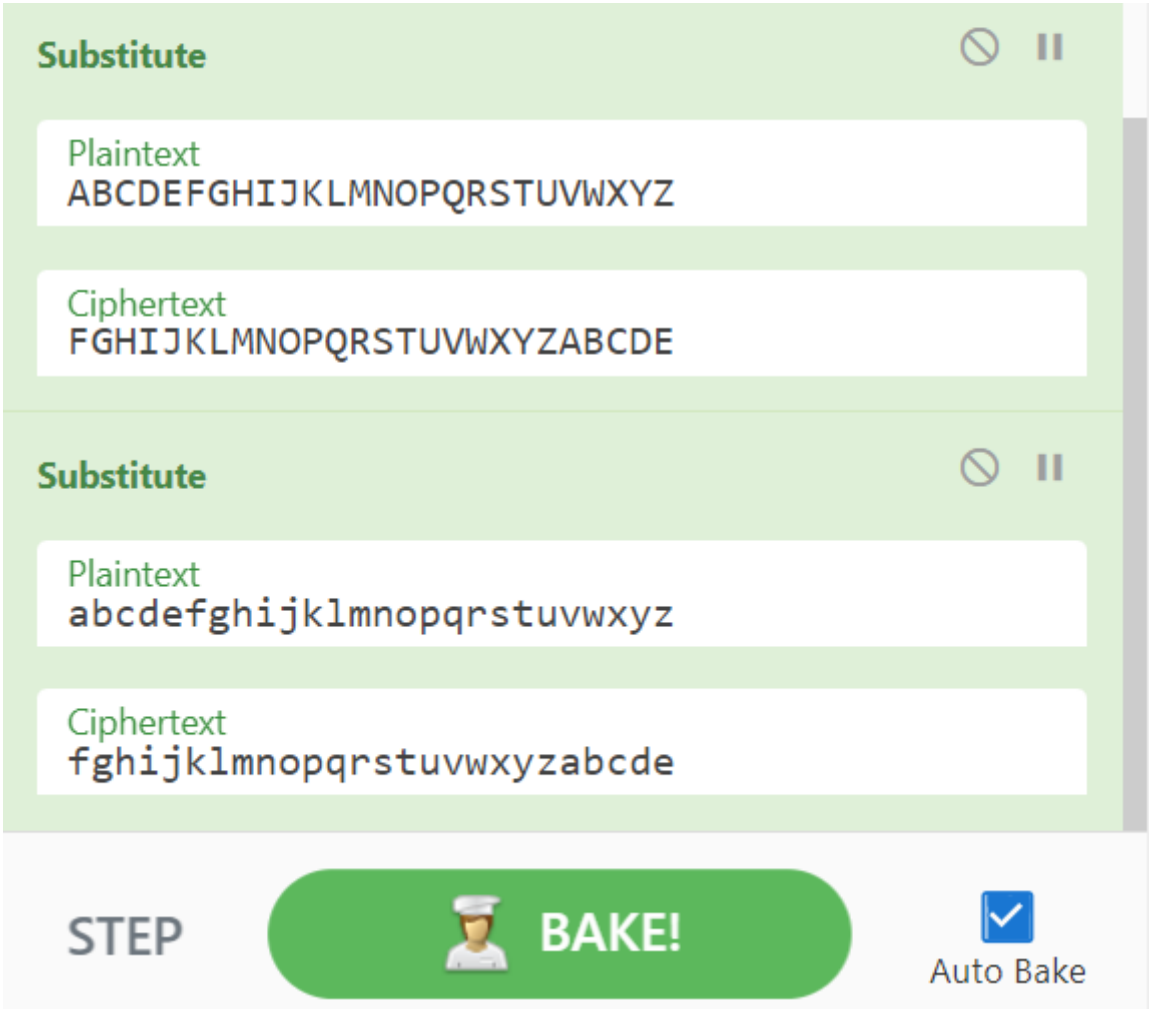
Vigenère Decode

Key
hgame

From Base64

Alphabet
A-Za-z0-9+/=

Remove non-alphabet chars



得到字符串 `hOgralmde_{OWfe_lccrOympet_OtgOr_atphhey_!w}`,

最后套了个 2 栏的栅栏解密

栅栏密码

在下面的文本框输入明文或密文，点加密或解密，文本框中即可出现所得结果

加密 解密 列举加密 列举解密 栏数: 2 只列举完整匹配的

密文框:

```
hOgralmde_{OWfe_lccrOympet_OtgOr_atphhey_!w}

2栏:
hgame{Welc0me_t0_the_w0rld_0f_crypt0graphy!}

4栏:
haeWl0et_h_0l_fcytgah!gm{ecm_0tewrd0_rp0rpy}

11栏:
h0ma0Wptgfepretha__hll0emctydcg_er0!_0rw{y_}
```

English Novel

给了加密脚本，写出对应脚本

```
def decrypt(result, key):
    assert len(result) <= len(key)
    data = ""
    for i in range(len(result)):
        if result[i].isupper():
            data += chr((ord(result[i]) - ord('A') - key[i]) % 26 +
ord('A'))
        elif result[i].islower():
            data += chr((ord(result[i]) - ord('a') - key[i]) % 26 +
ord('a'))
        else:
            data += result[i]
    return data

def getkey(data, result):
    assert len(data) == len(result)
    key = []
    for i in range(len(data)):
        key.append((ord(result[i])-ord(data[i])) % 26)
    return key

key = []
for i in range(409,0,-1):
    with open(f"encrypt/part{i}.txt")as enc:
        result = enc.read()
    for z in range(409,0,-1):
        with open(f"original/part{z}.txt") as ori:
            data = ori.read()
            try:
                key = getkey(data,result)
            except:
                continue
            flag = "klsyf{wO_jOv_caOz_'KsOao-bln1qstxp_juqfqy'?}"
            for j in range(0,len(key)-len(flag)):
                hg = decrypt(flag,key[j:])
                if("hgame" in hg):
                    print(i,z)
                    print(hg)

```

脚本会输出可能的 flag

```
409 15
hgame{D0_y0v_kn0w_'Kn0wn-bln1ntext_jttack'?}
404 408
odplv{B0_u0m_yh0x_'Pr0ig-dcq1idwpm_uhgame'?}
393 247
ianxd{I0_x0j_eh0u_'Ki0ul-ndo1qyhuk_phgame'?}
393 22
hgame{D0_y0v_kn0w_'Kn0wo-pla1ntexp_attfck'?}
392 290
ectns{J0_t0w_wq0w_'Rg0yt-uaz1ciaez_vhgame'?}
392 140
hgame{D0_j0u_kn0w_'Kn0an-pla1ntext_attack'?}
383 183
382 366
hgame{D0_y0v_kn0w_'Kn0wn-pla1ntext_jttack'?}
367 248
```

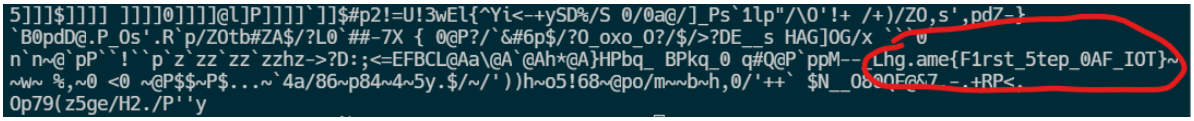
没等到真 flag，但是可以根据语义推出 flag

`hgame{D0_y0u_kn0w_'Kn0wn-pla1ntext_attack'?}`

IoT

饭卡的uno

给了 hex 文本，一开始以为要买 arduino 才能做，后来尝试把 hex 值转成字符串，过滤了非ascii字符



脚本

```python
import binascii
def hex2char(data):
#    binascii.a2b_hex(hexstr)
    output = binascii.unhexlify(data)
    return output

def char2hex(data):
    # data = b'data'
#    binascii.b2a_hex(data)
    output = binascii.hexlify(data)
    return output
if __name__ == '__main__':
```

```
13      s =
hex2char(b'''020000040000FA100000000C9435000C945D000C945D000C945D00241000100
00C945D000C945D000C945D000C945D00EC100020000C945D000C945D000C945D000C945D00D
C100030000C945D000C945D000C945D00CC100040000C946C010C945D000C94DC010
C94B601D2100050000C945D000C945D000C945D000C945D00AC100060000C945D000C945D00A
10211241FBECFEF2310007000D8E0DEBFCDBF11E0A0E0B1E0ECEAF5E0F21000800002C005900
D92AE32B107D9F721E0AEE28110009000B1E001C01D92A43DB207E1F710E0C5E3551000A000D
0E004C02197FE010E94CE02C433D107E41000B000C9F70E940E020C94D4020C940000AF92771
000C000BF92CF92DF92EF92FF920F931F93CF93451000D000DF936C017B018B01040F151FEB0
15E01A71000E000AE18BF08C017D10759F06991D601ED913C1000F000FC910190F081E02DC60
10995892B79F7DB10010000C501DF91CF911F910F91FF90EF90DF908C10011000CF90BF90AF9
00895FC01538D448D252F531001200030E0842F90E0821B930B541710F0CF9691100130000089
501970895FC01918D828D981761F0C310014000A28DAE0FBF2FB11D5D968C91928D9F5FDA100
150009F73928F90E008958FEF9FEF0895FC01B910016000918D828D981731F0828DE80FF11D8
58D6C1001700090E008958FEF9FEF0895FC01918D228DFF10018000892F90E0805C9F4F821B9
1098F739927841001900008958EE291E00E94BD0021E0892B09F4D01001A00020E0822F08958
0E090E0892B29F00E94C21001B000C90081110C9400000895FC01A48DA80FC21001C000B92FB
11DA35ABF4F2C91848D90E00196991001D0008F739927848FA689B7892C93A089B189B91001E
0008C91837080648C93938D848D981306C05A1001F0000288F389E02D80818F7D80830895EF9
2BE10020000FF920F931F93CF93DF93EC0181E0888FD0100210009B8D8C8D98131AC0E889F98
9808185FFA01002200015C09FB7F894EE89FF896083E889F98942100230008081837080648 08
39FBF81E090E0DF914410024000CF911F910F91FF90EF900895F62E0B8D971002500010E00F5
F1F4F0F731127E02E8C8D8E1152100260000CC00FB607FCFACFE889F989808185FFB91002700
0F5CFCE010E94DD00F1CFEB8DEC0FFD2F0D10028000F11DE35AFF4FF0829FB7F8940B8FEA897
410029000FB8980818062CFCFCF93DF93EC01888D831002A0008823B9F0AA89BB89E889F9898
C9185FDF11002B00003C0808186FD0DC00FB607FCF7CF8C917F1002C00085FFF2CF808185FFE
DCFCE010E94DD005A1002D000E9CFDF91CF9108951F920F920FB60F92411002E00011242F933
F938F939F93AF93BF9380914C1002F000D0019091D101A091D201B091D30130916010030000C
F0123E0230F2D3758F50196A11DB11D14100310002093CF018093D0019093D101A093D2017B1
0032000B093D3018091CB019091CC01A091CD01EC10033000B091CE010196A11DB11D8093CB0
190938810034000CC01A093CD01B093CE01BF91AF919F910D100350008F913F912F910F900FB
E0F901F901895861003600026E8230F0296A11DB11DD2CF1F920F9236100370000FB60F92112
42F933F934F935F936F9378100380007F938F939F93AF93BF93EF93FF938EE2EF1003900091E
00E94DD00FF91EF91BF91AF919F919D1003A0008F917F916F915F914F913F912F910F901E100
3B0000FBE0F901F9018951F920F920FB60F92BD1003C00011242F938F939F93EF93FF93E0913
E011E1003D000F0913F018081E0914401F091450182FD5F1003E0001BC09081809147018F5F8
F7320914801DE1003F000821741F0E0914701F0E0E25DFE4F958FFA1004000080934701FF91E
F919F918F912F910F90D2100410000FBE0F901F9018958081F4CF789484B50B1004200082608
4BD84B5816084BD85B5826085BDF01004300085B5816085BD80916E00816080936E007E10044
000109281008091810082608093810080917010045000810081608093810080918000816080 9
32110046000800080091B10084608093B1008091B000E11004700081608093B00080917A00846
080937A00DC1004800080917A00826080937A0080917A00816006100490008093 7A0080917A0
0806880937A0010922D1004A000C100E0913E01F0913F0182E08083E091441004B0003A01F09
13B011082E0913C01F0913D01451004C0008FEC808310924601E0914201F09143014C1004D00
086E08083E0914001F0914101808180615C1004E0008083E0914001F0914101808188608083A
81004F000E0914001F0914101808180688083E0912A100500004001F091410180818F7D8083C
0E0D0E0871005100048E150E062E171E08EE291E00E945F000C1005200042E050E06BE271E08
EE291E00E945F00F910053000209771F30E94C900882351F30E940000A410054000E7CFEEE2F
1E01382128288EE93E0A0E0C210055000B0E084839583A683B78384E091E09183A0100560008
08385EC90E09587848784EC90E097878210057000868780EC90E0918B808B81EC90E0938B701
0058000828B82EC90E0958B848B86EC90E0978B4D10059000868B118E128E138E148E0895EE0
FFF1F101005A0000590F491E02D0994F894FFCF000000002D1005B000FF005F008C004C01BD0
09B00AF0068672E1005C000616D657B46317273745F357465705F30410A05D000465F494F547
D000D0A00FC107E0000112484B714BE81FFF0D085E080938100F7107E100082E08093C00088E
18093C10086E0809377107E2000C20080E18093C4008EE0C9D0259A86E02C107E300020E33CE
F91E0309385002093840096BBD3107E4000B09BFECF1D9AA8958150A9F7CC24DD24C4107E500
088248394B5E0AB2EA1E19A2EF3E0BF2EE7107E6000A2D0813461F49FD0082FAFD0023811F03
6107E7000013811F484E001C083E08DD089C08234E0107E800011F484E103C0853419F485E0A
```

```
6D080C0E4107E9000853579F488D0E82EFF2485D0082F10E0AE107EA000102F00270E291F290
00F111F8ED06801E7107EB0006FC0863521F484E090D080E0DECF843638107EC00009F040C07
0D06FD0082F6DD080E0C81688107ED00080E7D80618F4F601B7BEE895C0E0D1E017107EE0006
2D089930C17E1F7F0E0CF16F0E7DF06D8107EF00018F0F601B7BEE89568D007B600FCFDCFD41
07F0000A601A0E0B1E02C9130E011968C91119780107F100090E0982F8827822B932B1296FA0
10C0160107F200087BEE89511244E5F5F4FF1E0A038BF0790107F300051F7F601A7BEE89507B
600FCFDCF97BE46107F4000E89526C08437B1F42ED02DD0F82E2BD052107F50003CD0F601EF2
C8F010F5F1F4F84911BD097107F6000EA94F801C1F70894C11CD11CFA94CF0C13107F7000D11
C0EC0853739F428D08EE10CD085E9AC107F80000AD08FE07ACF813511F488E018D01DD067107
F900080E101D065CF982F8091C00085FFFCCF94107FA0009093C60008958091C00087FFFCCF8
09118107FB000C00084FD01C0A8958091C6000895E0E648107FC000F0E098E1908380830895E
DDF803219F02E107FD00088E0F5DFFFCF84E1DECF1F93182FE3DFCA107FE0001150E9F7F2DF1
F91089580E0E8DFEE27F6047FF000FF270994CA027FFE0004047900000001FF''')

    for c in s:
        if c in range(20,128):
            print(chr(c),end="")
    # print(s)
```

MISC

欢迎欢迎！热烈欢迎！

签到题。已经取关了，就不放截图了。

这个压缩包有点麻烦

压缩包，先是 6 位密码爆破，本文直接用 bandzip 爆破出密码 483279。

接下来的过程全部使用 ARCHPR 完成。

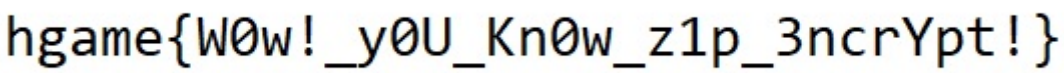
然后将 password-note 作为字典进行爆破。

得到下一层的 flag.zip 和 readme。

发现压缩包内 readme.txt 和包外同名文件 crc 校验相同,确定是明文爆破，这里较新版的 ARCHPR 会尝试爆破出密码，比较浪费时间，本文在中途停止爆破，ARCHPR 会得到解密所需的三个密钥，用这三个密钥就可以让 ARCHPR 解压出文件了。

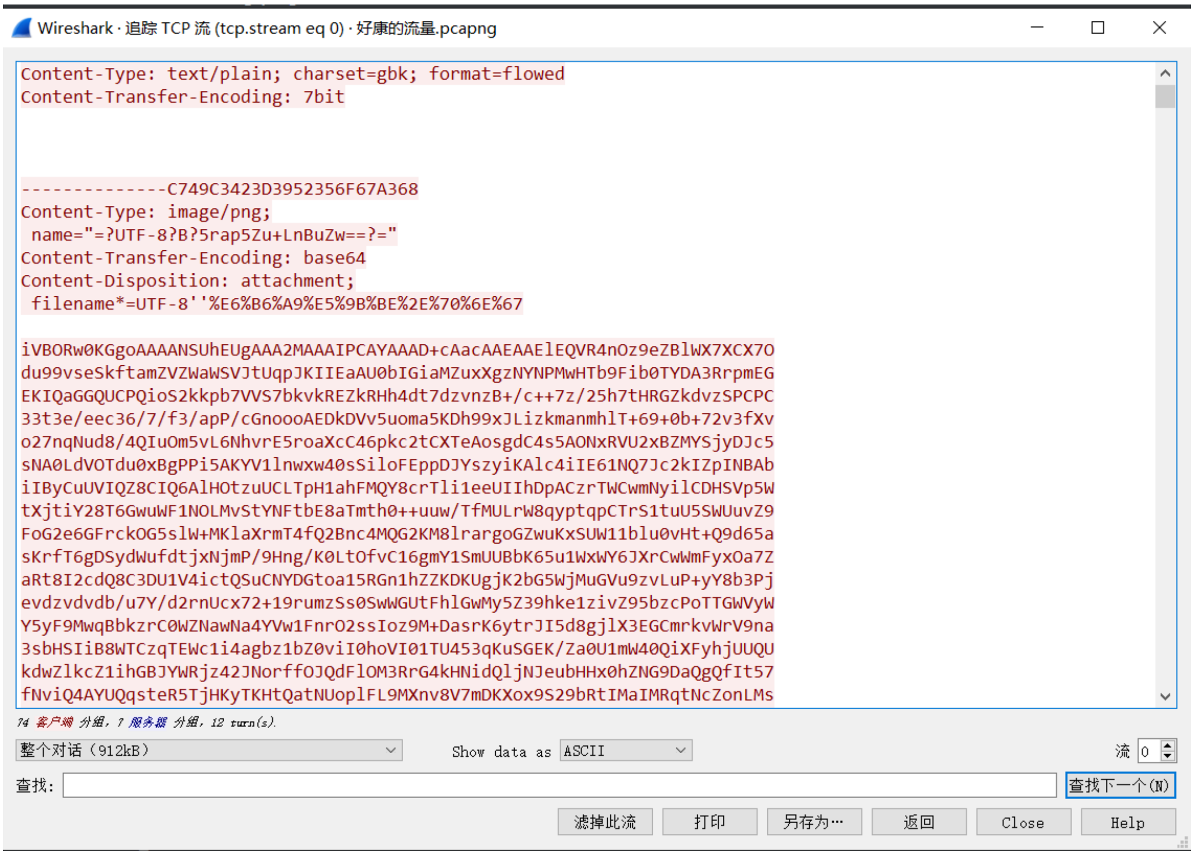
发现解压出的 flag.png 可以作为压缩文件打开，于是使用 foremost 分离。

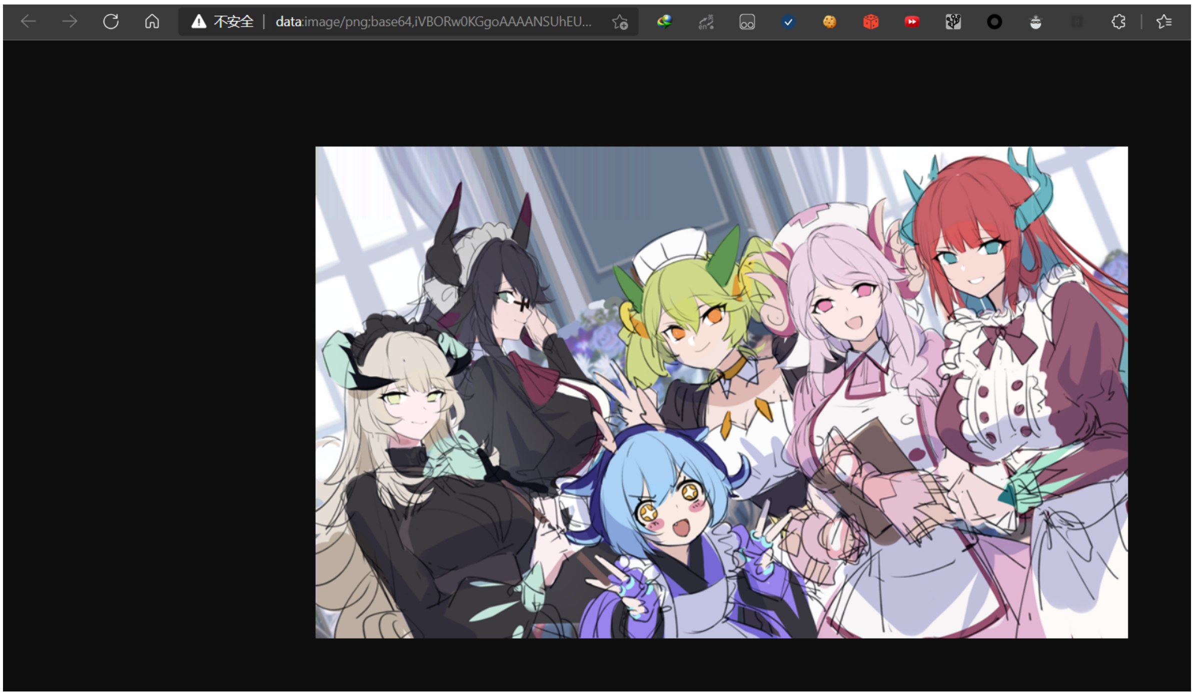
没有任何提示，猜测伪加密，使用 ZipCenOp.jar 处理分离出的压缩文件，成功得到 flag 图片。



好康的流量

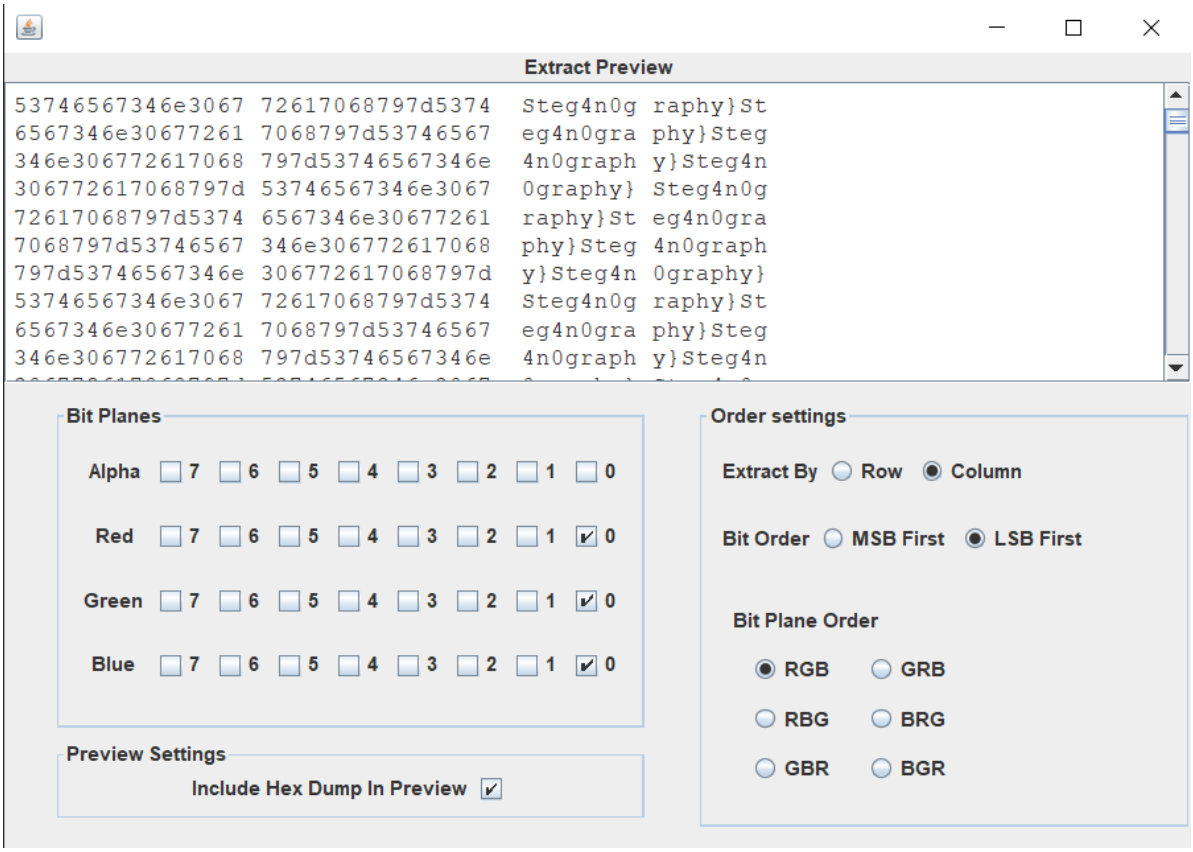
追踪 tcp 流可以发现邮件收发流量，从邮件里手动复制出涩图的base64，加上图片头（）即可在浏览器打开，





另存为本地图片之后，扔进 Stegsolve.jar 里，在 Green Plane 2 发现条码，在网站 [Barcode Reader. Free Online Web Application (inliteresearch.com)]() 识别出 hgame{ez_1mg_

接下来找了好久，发现 LSB 隐写处有



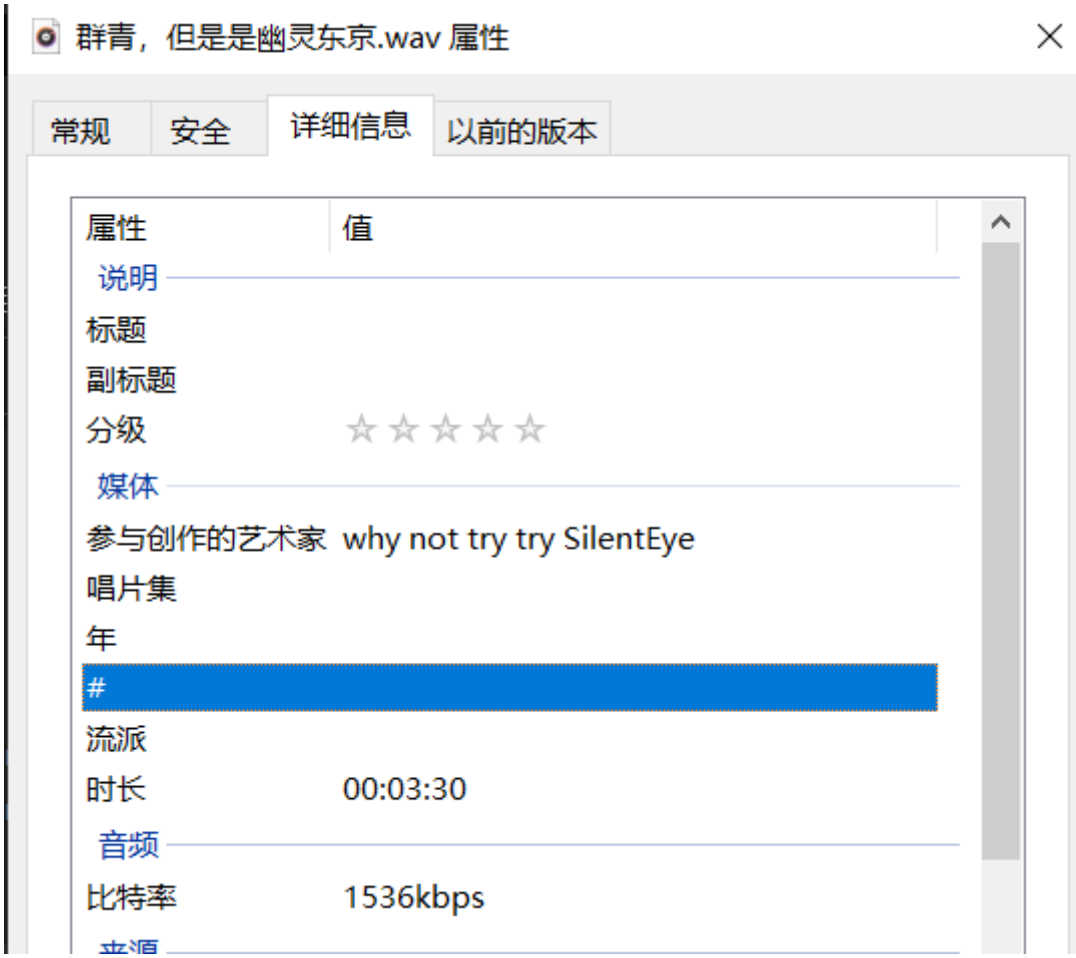
一开始还没发现是 flag 的后半部分，在狂找 hgame 字样，看了好久才突然发现这不就是后半部分吗。

最后组成 flag

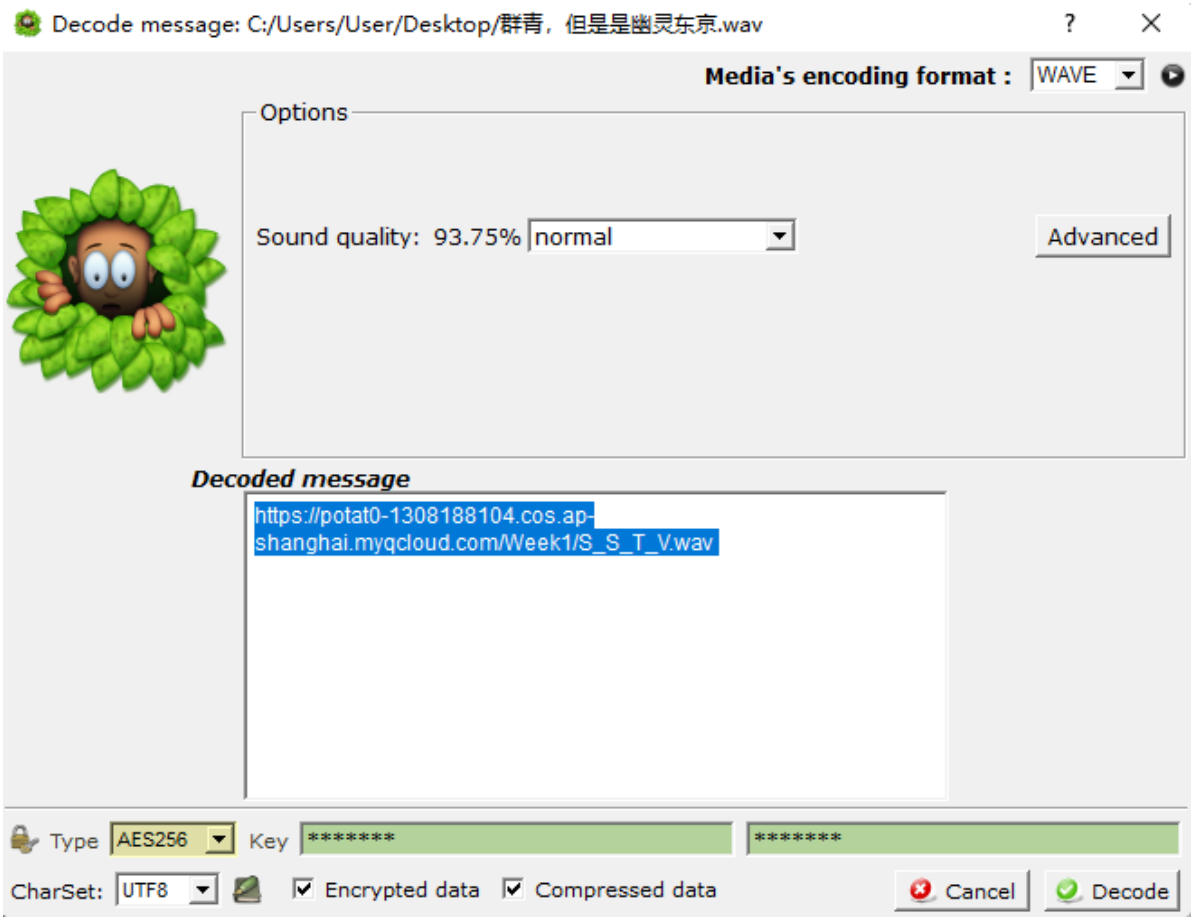
```
hgame{ez_1mg_Steg4n0graphy}
```

群青(其实是幽灵东京)

根据提示要用“多感官”，百度了好久，接着发现音频文件属性处有个

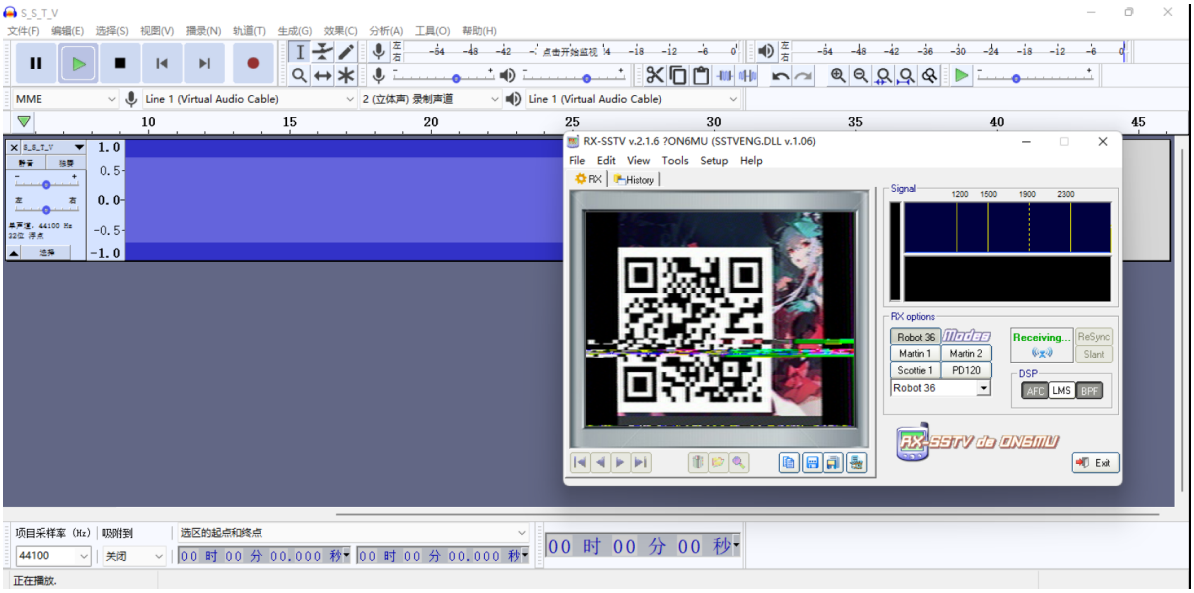


安装 SilentEye 进行解密，密码为题干的提示 Yoasobi。得到新的音频网址



搜索 SSTV，

加上关键字 ctf，搜出一篇文章，参考之。 [Misc SSTV慢扫描电视&无线电 - LEOGG - 博客园 (cnblogs.com)]



扫描二维码拿到 flag。

hgame{1_c4n_5ee_the_wav}

PWN

test_your_nc

nc 直接连，题目直接 给了 shell。

REVERSE

easyasm

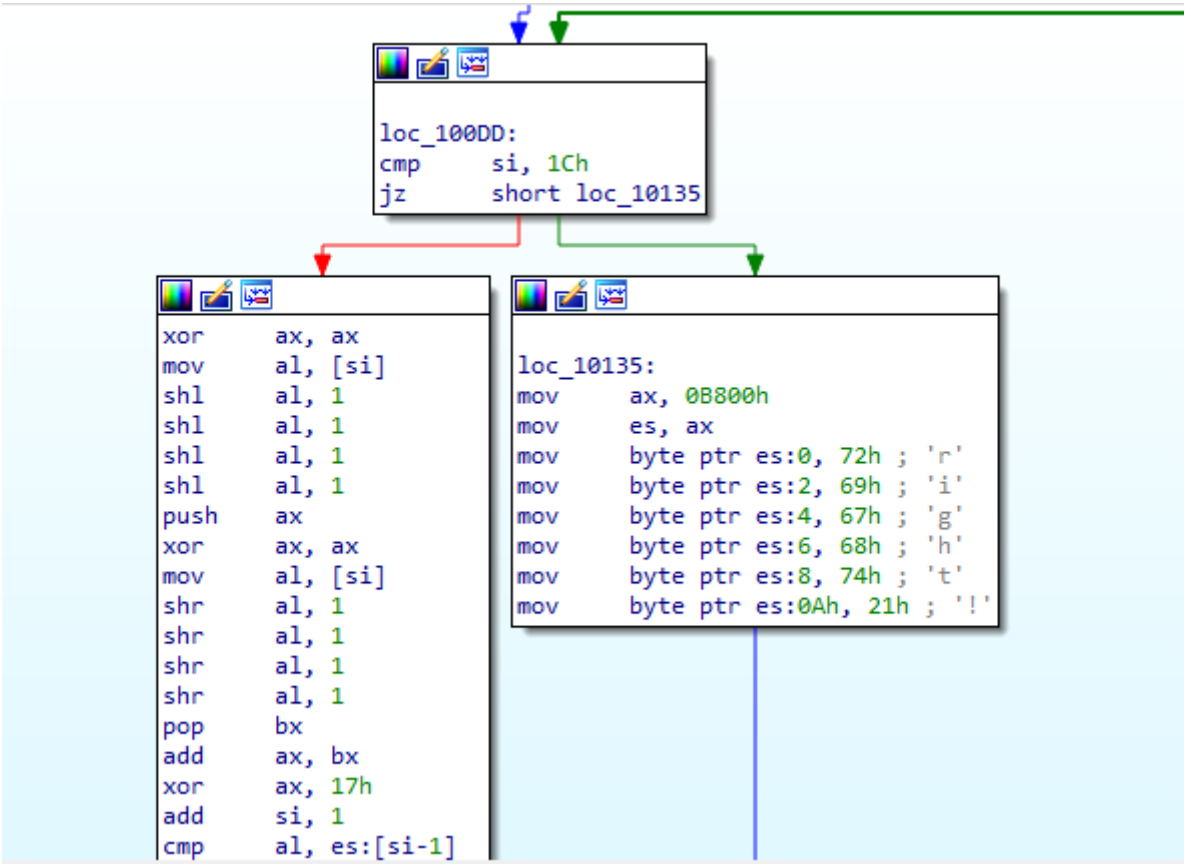
本题知识点在汇编代码上，使用 IDA无法反汇编，手动分析汇编代码，

```
public start
start proc near
mov     ax, seg dseg
mov     ds, ax
assume ds:dseg
mov     ax, seg seg001
mov     es, ax
assume es:seg001
mov     si, 0
```

其中 dseg 段存储了

```
dseg:0000                assume cs:dseg
dseg:0000 aHgameFillInYou db 'hgame{Fill_in_your_flag}',0
dseg:0019                db    0
```

回到输出 ”right“ 关键的判断逻辑：



可以看到经过 28 次的循环，循环中

```
shl     al, 1
shl     al, 1
shl     al, 1
shl     al, 1
```

相当于 c 的

```
al <<= 4;
```

shr 同理，而 al ah 是将 ax 这个 16 位寄存器分成了两个，

也就是高低位分别处理，可以还原出代码

```
#include <stdio.h>
#include <stdlib.h>

#define _DWORD uint32
#define CHAR_BITS 8
const char *my_itoa_buf(char *buf, size_t len, int num)
{
    static char loc_buf[sizeof(int) * CHAR_BITS]; /* not thread safe */

    if (!buf)
    {
        buf = loc_buf;
        len = sizeof(loc_buf);
    }

    if (snprintf(buf, len, "%d", num) == -1)
        return ""; /* or whatever */

    return buf;
}
const char *my_itoa(int num) { return my_itoa_buf(NULL, 0, num); }
int main()
{
    int si = 0;
    char *s2 = "hgame{Fill_in_your_flag}"; // 这个字符串不参与真实 flag 的计算。
    int es[28] = {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};
    while (si != 0x1C)
    {
        int b = es[si] ^ 0x17;
        int blow = b & 0b00001111;
        int bhigh = b & 0b011110000;
        b = (blow << 4 | bhigh >> 4);
        printf("%c", b);
        si++;
    }
    printf("right!");
}
```

运行后可以得到 flag

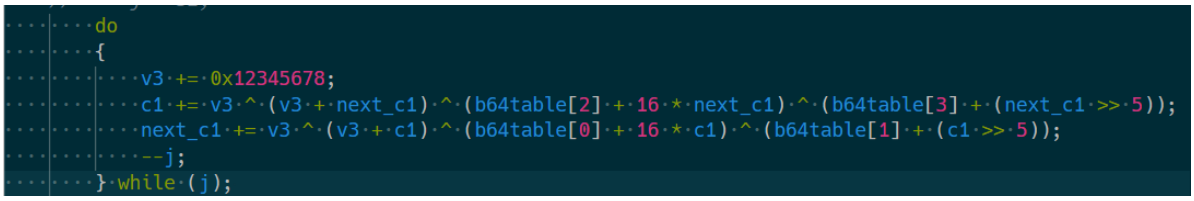
```
E:\data-structure\asm\build>"E:\data-structure\asm\build\demo.exe"
hgame{welc0me_to_4sm_w0rld}right!
E:\data-structure\asm\build>
```

creakme

这题对数据类型（字节长度）的控制很重要。

先无脑拖 IDA 看看伪代码。

抛开循环,可以发现关键逻辑一处



```
do
{
    v3 += 0x12345678;
    c1 += v3 ^ (v3 + next_c1) ^ (b64table[2] + 16 * next_c1) ^ (b64table[3] + (next_c1 >> 5));
    next_c1 += v3 ^ (v3 + c1) ^ (b64table[0] + 16 * c1) ^ (b64table[1] + (c1 >> 5));
    --j;
} while (j);
```

直接百度这部分代码，发现高度疑似 tea 加密，所以猜测本题加密也为对称加密算法，

按照上述猜测爆改一下伪代码逻辑

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>

#define _DWORD uint32_t
#define _BYTE uint8_t

// __int128_t xmmword_402170 = 0x65E0F2E3CF9284AABA5A126DAE1FEDE6;
// __int128_t xmmword_402180 = 0x0ED9CE5ED52EB78C203C144C48D93488;

char xmmword_402170[] = {0x65, 0xE0, 0xF2, 0xE3, 0xCF, 0x92, 0x84, 0xAA, 0xBA, 0x5A, 0x12, 0x6D, 0xAE, 0x1F, 0xED, 0xE6};
char xmmword_402180[] = {0xED, 0x9C, 0xE5, 0xED, 0x52, 0xEB, 0x78, 0xC2, 0x03, 0x0C, 0x14, 0x4C, 0x48, 0xD9, 0x34, 0x88};

// char newtype[] = {0xE3, 0xF2, 0xE0, 0x65, 0xAA, 0x84, 0x92, 0xCF, 0x6d, 0x12, 0x5a, 0xba, 0xe6, 0xed, 0x1f, 0xae, 0xed, 0xe5, 0x9c, 0xed, 0xc2, 0x78, 0xeb, 0x52, 0x4c, 0x14, 0x0c, 0x03, 0x88, 0x34, 0xd9, 0x48};
char newtype[] = {0x65, 0xE0, 0xF2, 0xE3, 0xCF, 0x92, 0x84, 0xAA, 0xBA, 0x5A, 0x12, 0x6D, 0xAE, 0x1F, 0xED, 0xE6,0xED, 0x9C, 0xE5, 0xED, 0x52, 0xEB, 0x78, 0xC2, 0x03, 0x0C, 0x14, 0x4C, 0x48, 0xD9, 0x34, 0x88};

char *reverseArray(char a[], int len)
{
    char temp = 0;
    int n = len;
    for (int i = 0; i < n / 2; ++i)
    {
        temp = a[n - i - 1];
        a[n - i - 1] = a[i];
        a[i] = temp;
    }
    return a;
}


int __cdecl main(int argc, const char **argv, const char **envp)
{
    long int v3;     // edx
```

```
    int i;                // esi
    unsigned int c1;      // edi
    unsigned int next_c1; // ebx
    int j;                // esi
    int i_3;              // esi
    _DWORD b64table[17];  // [esp+Ch] [ebp-8Ch] BYREF
    __int128 v11[2];      // [esp+50h] [ebp-48h]
    char Arglist[32];     // [esp+70h] [ebp-28h] BYREF
    int v13;              // [esp+90h] [ebp-8h]
    int i_2;              // [esp+94h] [ebp-4h]

    // reverseArray(newtype, 32);
    memset(Arglist, 0, sizeof(Arglist));
    // scanf("%s", Arglist);
    strcpy((char *)b64table,
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=");
    strcpy((char *)Arglist, ("hgame{This_maybe_a_fake_flag}!!"));
    strcpy((char *)Arglist, newtype);
    reverseArray(Arglist,32);
    for (i = 0;i<16;i++){
        printf("%x ",Arglist[i]);
    }
    printf("\n%d",sizeof(*(_DWORD *)&Arglist[0]));
    printf("\n");
    long int delta=0x12345678;
    v3 = 0;
    i_2 = 0;
    // for (i = 0; i < 32; i_2 = i)
    // {
    //     // printf("%d %d %d\n",i,i_2,i_3);
    //     c1 = *(_DWORD *)&Arglist[i];
    //     next_c1 = *(_DWORD *)&Arglist[i + 4];
    //     printf("ori: %08x %08x\n", c1, next_c1);
    //     v13 = 0;
    //     j = 32;
    //     do
    //     {
    //         v3 += 0x12345678;
    //         c1 += v3 ^ (v3 + next_c1) ^ (b64table[2] + 16 * next_c1) ^
(b64table[3] + (next_c1 >> 5));
    //         next_c1 += v3 ^ (v3 + c1) ^ (b64table[0] + 16 * c1) ^
(b64table[1] + (c1 >> 5));
    //         --j;
    //     } while (j);
    //     i_3 = i_2;
    //     v3 = 0;
    //     *(_DWORD *)&Arglist[i_2] = c1;
    //     *(_DWORD *)&Arglist[i_3 + 4] = next_c1;
    //     i = i_3 + 8;
    //     printf("new: %08x %08x\n", c1, next_c1);
    // }
    i_2 = 0;
    for (i = 0; i < 32; i_2 = i)
    {
        c1 = *(_DWORD *)&Arglist[i];
        next_c1 = *(_DWORD *)&Arglist[i + 4];
        printf("ori: %08x %08x\n", c1, next_c1);
        v13 = 0;
```

```
        j = 32;
        v3 = delta << 5;
        do
        {
            next_c1 -= v3 ^ (v3 + c1) ^ (b64table[0] + 16 * c1) ^
(b64table[1] + (c1 >> 5));
            c1 -= v3 ^ (v3 + next_c1) ^ (b64table[2] + 16 * next_c1) ^
(b64table[3] + (next_c1 >> 5));
            v3 -= 0x12345678;
            --j;
        } while (j);
        i_3 = i_2;
        v3 = 0;
        *(_DWORD *)&Arglist[i_2] = c1;
        *(_DWORD *)&Arglist[i_3 + 4] = next_c1;
        i = i_3 + 8;
        // printf("new: %08x %08x\n", c1, next_c1);
        printf("new: %08x %08x\n", *(_DWORD *)&Arglist[i_2], *(_DWORD
*)&Arglist[i_3 + 4] );
        // printf("%d %d %d\n",i,i_2,i_3);
    }
    // v11[0] = (__int128_t)xmmword_402180;
    // v11[1] = (__int128_t)xmmword_402170;
    printf("%x\n",b64table[0]);
    printf("%x\n",b64table[1]);
    printf("%x\n",b64table[2]);
    printf("%x\n",b64table[3]);
    while (1)
    {
        printf("%x ", Arglist[v3]);
        if (++v3 >= 32)
        {
            printf("\n%s",Arglist);
            printf("right!", b64table[0]);
            return 0;
        }
    }
    printf("wrong!", b64table[0]);
    return 0;
}


void encrypt(unsigned long k[], unsigned long text[])
{
    unsigned int y = text[0];
    unsigned int z = text[1];
    unsigned int delta = 0x12345678;
    unsigned int sum = 0;
    int n;

    printf("%08x %08x\n", y, z);
    for (n = 0; n < 32; n++)
    {
        sum += delta;
        y += sum ^ ((z << 4) + k[2]) ^ (z + sum) ^ ((z >> 5) + k[3]);
        z += sum ^ ((y << 4) + k[0]) ^ (y + sum) ^ ((y >> 5) + k[1]);
    }

```

```
    text[0] = y;
    text[1] = z;
}

void decrypt(unsigned long k[], unsigned long text[])
{
    unsigned long y = text[0];
    unsigned long z = text[1];
    unsigned long delta = 0X12345678;
    unsigned long sum = delta << 5;
    printf("%08x %08x\n", y, z);
    int n;
    for (n = 0; n < 32; n++)
    {
        // z -= ((y << 4) + k[2]) ^ (y + sum) ^ ((y >> 5) + k[3]);
        // y -= ((z << 4) + k[0]) ^ (z + sum) ^ ((z >> 5) + k[1]);
        z -= sum ^ ((y << 4) + k[0]) ^ (y + sum) ^ ((y >> 5) + k[1]);
        y -= sum ^ ((z << 4) + k[2]) ^ (z + sum) ^ ((z >> 5) + k[3]);
        sum -= delta;
    }
    text[0] = y;
    text[1] = z;
}

```

成功解密出 flag。

```
ffffff88 34 ffffffd9 48 4c 14 c 3 ffffffc2 78 ffffffeb 52 fffffed ffffffe5 ffffff9c ffffffed
4
ori: 48d93488 030c144c
new: 6d616768 34487b65
ori: 52eb78c2 ed9ce5ed
new: 5f797070 34633476
ori: ae1fede6 ba5a126d
new: 6e306974 00007d21
ori: cf9284aa 65e0f2e3
new: 00000000 00000000
44434241
48474645
4c4b4a49
504f4e4d
68 67 61 6d 65 7b 48 34 70 70 79 5f 76 34 63 34 74 69 30 6e 21 7d 0 0 0 0 0 0 0 0 0 0
hgame{H4ppy_v4c4ti0n!}right!
```

Flag Checker

先作为 zip，提取出 apk 中的 class.dex，利用 d2j 将 dex 转化成 jar 文件，用 jd-gui 打开可以看见源码

进入 MainActivity

```
public class MainActivity extends AppCompatActivity {
  public static byte[] encrypt(String paramString1, String paramString2)
throws Exception {
    SecretKeySpec secretKeySpec = new SecretKeySpec(paramString2.getBytes(),
0, paramString2.length(), "RC4");
    Cipher cipher = Cipher.getInstance("RC4");
    cipher.init(1, secretKeySpec);
    return cipher.doFinal(paramString1.getBytes());
  }

  protected void onCreate(Bundle paramBundle) {
    super.onCreate(paramBundle);
```

```
    setContentView(2131296284);
    ((Button)findViewById(2131165218)).setOnClickListener(new
View.OnClickListener() {
          public void onClick(View param1View) {
            String str =
((EditText)MainActivity.this.findViewById(2131165238)).getText().toString();
            byte[] arrayOfByte = new byte[0];
            try {
              byte[] arrayOfByte1 = MainActivity.encrypt(str, "carol");
              arrayOfByte = arrayOfByte1;
            } catch (Exception exception) {
              exception.printStackTrace();
            }
            if (Base64.encodeToString(arrayOfByte, 0).replace("\n",
"").equals("mg6CITV6GEaFDTYnObFmENOAVjKcQmGncF90WhqvCFyhhsyqq1s=")) {
              Toast.makeText((Context)MainActivity.this,
"Congratulations!!!", 1).show();
            } else {
              Toast.makeText((Context)MainActivity.this, "Fail,try again.",
1).show();
            }
          }
        });
  }
}
```

可以看到，我们的输入 被 RC4 加密后与 `mg6CITV6GEaFDTYnObFmENOAVjKcQmGncF90WhqvCFyhhsyqq1s=` 进行比较，所以只需要用 key carol 对上述 base64 RC4解密即可。