HGAME2020 Week1 Writeup

原创

wh201906 于 2020-01-24 22:17:12 发布 1802 收藏

分类专栏： CTF试水

本文链接：https://blog.csdn.net/wh201906/article/details/104081772

纯正新人CTF选手的误打误撞上分过程

“这题目上头是上头但是咱不上分啊”

Crypto - InfantRSA

题目：

```
真*签到题
p = 681782737450022065655472455411;
q = 675274897132088253519831953441;
e = 13;
c = pow(m,e,p*q) = 27569846508236107014517368841149631154217290260855985901984l
```

还有一段用来加密的脚本

```
#!/usr/bin/env python3
from secret import flag
assert flag.startswith(b'hgame{') and flag.endswith(b'}')

m = int.from_bytes(flag, byteorder='big')

p = 681782737450022065655472455411
q = 675274897132088253519831953441
e = 13
c = pow(m, e, p*q)

assert c == 27569846508236107014517368841149631154217290260855985901984l
```

开始上手时没去看脚本（挖坑）直接去搜了一下RSA的相关知识，因为平时不太使用Python所以最初想到的是用WolframAlpha来手动计算大数。草稿如下：

```
p 681782737450022065655472455411
q 675274897132088253519831953441
n=p*q 460390767897997184102969941508880171690097589571068900519251
ϕ=(p-1)(q-1) 460390767897997184102969941507523114055515479251893596110400
e(1<e<ϕ, gcd(e,ϕ)=1) 13
d(e*d mod ϕ=1) 14165869781476836433937536661769941970938937823135187572627 7
c=m ^ e mod n 27569846508236107014517368841149631154217290260855985901984 1

m 390621104726693889143894280640873352363348319913332 45
```

其中计算私钥d时将公式代入WolframAlpha直接作为方程求解，得到的解为

```
460390767897997184102969941507523114055515479251893596110400 * n + 14165869781476836433937536661769941970938937823135187572627 7, n ∈ Z
```

取n=0（n为其它整数时也有效），用WolframAlpha计算c ^ d mod n得到明文m

m怎么转换为flag呢？查看题目里面的脚本并结合百度，得知在Python中使用m.to_bytes()函数即可。当中有两个必填参数，byteorder根据脚本内容设置为big，length经测试只要大于flag文本的长度就能得到flag文本，所以填个100就差不多了
最终flag：hgame{t3Xt6O0k_R5A!!!}

后记：
《如果早知道，Python也能算大数…》(下次要把题目信息先理清楚再下手，如果一开始就查看题目里面的脚本则会发现py的这一特性并提高解题速度)
理论上公钥e和私钥d可以互换，即明文通过d加密得到密文，密文通过e解密得到明文
后期百度到d的一个解为：e ^ -1 mod φ，即d为e模φ的逆元，这里的e ^ -1不是1/e的意思

Crypto - Affine

题目：

```
Some basic modular arithmetic...
```

题干在一个py脚本中

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import gmpy2
from secret import A, B, flag
assert flag.startswith('hgame{') and flag.endswith('}')

TABLE ='zxcvbnmasdfghjklqwertyuiop1234567890QWERTYUIOPASDFGHJKLZXCVBNM'
MOD = len(TABLE)

cipher = ''
for b in flag:
    i = TABLE.find(b)
    if i == -1:
        cipher += b
    else:
        ii = (A*i + B) % MOD
        cipher += TABLE[ii]

print(cipher)
# A8I5z{xr1A_J7ha_vG_TpH410}
```

分析一下程序的逻辑，大概是取flag当中的每个字符，将其在TABLE中的下标i经过运算得到新的下标ii，并用ii在TABLE中对应的字符替换原字符，得到密文

```
A8I5z{xr1A_J7ha_vG_TpH410}
```

因为加密操作是针对单个字符进行的，并且加密过程与字符在flag当中的位置无关，所以可以尝试恢复部分flag

首先是flag的头部。A815z一定是由hgame加密得来的，通过分析每个字符的下标得到了以下对应关系（括号内数字为字符在TABLE中的下标）

```
h(12)->A(46)
g(11)->8(33)
a(7)->I(43)
m(6)->5(30)
e(18)->z(0)
```

分析用的python脚本如下

```
TABLE = 'zxcvbnmasdfghjklqwertyuiop1234567890QWERTYUIOPASDFGHJKLZXCVBNM'
cry = 'hgame-A8I5z{xr1A_J7ha_vG_TpH410}'
for b in cry:
    print(TABLE.find(b),end=' ')
```

加密的核心在于ii = (A * i + B) % MOD，MOD是TABLE的长度(62)，虽然对于取余的性质不太熟悉，但是考虑到g+h，m+a两组变换中均存在i+1->ii+13，有理由相信A的值就是13（不会仔细求证，因为不会）
同时注意到m+g,a+h两组变换的i差值在乘上A后反而大于ii差值，说明(A * 11 + B)的值已经大于MOD了,也就说明((6 * 13+B)-30) * n=((11 * 13+B)-33),n为大于等于2的整数
整理一下得到关系式

```
B=-48+62/(n-1)
```

得到B的可能取值为14、-17、-46、-47（因为小数好像无法mod？所以就直接取整数解了）
考虑字符e的变换结果刚好为0，则又能得到一个关键的式子

```
13 * 18+B=62 * n
B=62n-234
```

基本确定B是14
将A,B代入原始加密脚本中并对hgame加密，发现当A=13,B=14时可以得到结果A8I5z，说明A,B应该是正确的加密参数
（将原代码中的from secret行和所有assert行删除，手动定义A,B和flag即可验证加密）
进一步猜想，密文当中有长度为6的部分(TpH410)，很有可能是crypto
将其加密，得到的密文是TjR41q，与原密文非常相似，基本确定flag最后一段是crypto
对crypto进行大小写变换以及相似数字的替换并加密验证，最终得到cRYpt0
同理，结合语义分析，密文中长度为2的部分(vG)很可能是is，但s、S、5加密后的结果均不是G,遂尝试in，最终得到iN
在上述过程中得到了1<-t,A<-h的变换，遂结合语义猜测xr1A对应的结果为myth，但经多次尝试后证明了这一猜想是错误的
之前考虑到mod会导致i与ii之间并不是一一对应关系（其实是的），因此没有逐个寻找字符与密文的对应关系，但现在无路可走且考虑到有了对应关系后可以缩小搜索范围，遂对A ~ Z,a ~ z和0 ~ 9全部加密，得到对应关系如下

```
原文：abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890
密文：IbTazt8AvBfi5wq4QjX1JKF2RkLWgeh6OClsy9xG3YDpnEcoMNHSUZmr7PVdu0
```

（其实是一一对应的）
查表即可得到flag
最终flag：hgame{M4th_u5Ed_iN_cRYpt0}

后记：
这里其实是仿射密码，解密表达式为i=(A ^ -1)(ii - B) % MOD，A ^ -1为A模MOD的逆元

Crypto - not_One-time

题目：

```
In cryptography, the one-time pad (OTP) is an encryption technique that cannot be cracked, but...
Just XOR ;P
nc 47.98.192.231 25001
hint: reduced key space
```

还有一个脚本：

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import os, random
import string, binascii, base64

from secret import flag
assert flag.startswith(b'hgame{') and flag.endswith(b'}')

flag_len = len(flag)

def xor(s1, s2):
    #assert len(s1)==len(s2)
    return bytes( map( (lambda x: x[0] ^ x[1]), zip(s1, s2) ) )

random.seed( os.urandom(8) )
keystream = ''.join( [ random.choice(string.ascii_letters+string.digits) for _ in range(flag_len) ] )
keystream = keystream.encode()
print( base64.b64encode(xor(flag, keystream)).decode() )
```

先尝试直接试着在linux系统下运行了这个命令

```
nc 47.98.192.231 25001
```

得到了以下结果

```
JTcLBwQzE3czQV8UPkoIYWtbVT4PYnB2CUFxUQEzVhw8IVYvR0cdZXUJTg==
```

经过尝试，每一次运行命令所得到的结果都不一样，猜测得到的密文便是由上述脚本加密而得的，而题目应该是需要从密文当中恢复出flag
题目给了提示“reduced key space”，同时校内群里面也给了一个相关知识的网站，然而看完网站之后只大概了解到当密钥空间小于明文空间时加密并不安全（应该是这个意思）。
对代码进行分析，发现里面出现了自定的随机数种子，猜测此题可能和伪随机数有关，然而，进一步搜索发现os.urandom()函数就是用来做随机加密key的，其执行结果难以预料，说明伪随机数这个方向行不通
在从“reduced key space”角度多次进行搜索无果后，我尝试从题目“one time pad”的角度进行搜索，发现OTP极其安全。。。这题没法做了
然而，在搜索过程中接触到MTP(Many Time Pad)，通过进一步搜索，找到了一篇文章讲解相关内容，发现文中提到的Many Time Pad与本题有些类似。文中题目是若干明文由相同key加密得到若干密文，根据密文求key，并通过key解密目标密文；而本题是flag被随机key加密，需要解出flag。看上去两者所求不同，但是由于加密算法当中的异或运算满足交换律，本题也可看作是明文(随机key)经相同key(flag)加密得到一系列密文，求key(flag)。

继续研究文章，得到了几条重要信息

1.若a ^ x=b，则b ^ x=a（异或的自反性）

2.设明文为m1,m2，密文为c1,c2，则有c1 ^ c2=(m1 ^ key) ^ (m2 ^ key)=m1 ^ key ^ key ^ m2=m1 ^ m2

3.文章中推论得当c1 ^ c2为有意义的英文字母时，对应的m1,m2很可能一个是英文字母一个是空格

第三条概括下来就是，通过寻找有意义的c1 ^ c2来获得m1,m2可能的解

对应到本题，可以写出以下伪代码：

```
定义密文集合
定义每一位的解集S
定义临时解集TMP

for(密文中的每一位i):

    for(密文集合中的密文1,密文2):
        结果=密文1[i] ^ 密文2[i]

        for(任意字符j,任意字符k):
            if(任意字符j ^ 任意字符k==结果):
                TMP.加入元素(密文1[i] ^ j,密文2[i] ^ j,密文1[i] ^ k,密文2[i] ^ k)

        if(S[i]是空集):
            S[i]=TMP
        else:
            S[i]=求交集(TMP,S[i])
输出结果
```

根据题目中的脚本所提供的条件，j和k均在字母和数字的范围内，这就相当于一个c1 ^ c2有意义的限定条件。而求交集的过程，则是取每一组密文对所获得的解的公共部分，以此缩小解的范围，不过感觉本质上还是暴力

之后自己构造了flag和key用原脚本加密，并自己写代码去解出flag。写代码过程中因为自己埋的坑太多（逻辑错误、flag范围错误等等）导致调试了很久，最终代码如下：

```
# # -*- coding: utf-8 -*-

import base64, string
from socket import socket

table = (string.ascii_letters + string.digits).encode() # 源于题目中随机key
# result = (string.ascii_letters + string.digits + '''{}_''').encode() #坑1
result = (string.printable).encode()

ciphers = []
for i in range(30): # 设定获取密文数量
    sock = socket()
    sock.connect(('47.98.192.231', 25001))
    text=sock.recv(500)
    ciphers.append(text)
    sock.close()

for i in range(len(base64.b64decode(ciphers.pop(0)))):
    lis=set()
    for cur1 in ciphers:
        for cur2 in ciphers:
            cy1 = base64.b64decode(cur1)
            cy2 = base64.b64decode(cur2)
            if (cy1 == cy2):
                continue
            res=set()
            tmp1 = cy1[i] ^ cy2[i]
            for j in range(len(table)):
                for k in range(len(table)):
                    if (j == k):
                        continue
                    tmp2 = table[j] ^ table[k]
                    if (tmp1 == tmp2): # 使c1^c2有意义
                        res1 = int(cy2[i] ^ table[j]).to_bytes(1, 'big')
                        res2 = int(cy2[i] ^ table[k]).to_bytes(1, 'big')
                        if (result.find(res1) != -1): # 缩小解范围
                            res.add(res1)
                        if (result.find(res2) != -1): # 坑2
                            res.add(res2)
            if (lis == set()):
                lis = res
            if (res != set()):
                tmp = lis.intersection(res)
                if (tmp != set()):
                    lis.intersection_update(res)
                else:
                    lis.update(res)
    print(i,''.encode().join(lis))
```

坑1：
天真地认为flag当中除了数字和字母意外就只有}{_了，导致索引为13、16、21、22、23、24、36的解很奇怪（即使密文数量很大解却总是不唯一，而且没有公共部分）
坑2：
一开始的判断条件是当res1和res2都在result表内时才将res1,res2加入res内，很明显逻辑错误，因为当res1符合条件时res2的结果是未知的，错误的判断条件导致解集为空
sockXXX：
公告里面有给nc命令的替代方案，借此可以实现自动获取密文

把密文数量开到40～60基本就能唯一确定flag了，经测试开到100大概能在30min内跑完，性能还是可以（吧）
最终flag：hgame{r3us1nG+M3$5age-&&~rEduC3d_k3Y-5P4Ce}

Crypto - Reorder

题目：

```
We found a secret oracle and it looks like it will encrypt your input…
nc 47.98.192.231 25002
```

用nc访问该ip和端口，发现是一个交互式的程序，输入的字符会被打乱顺序输出

```
root@tesla:~# nc 47.98.192.231 25002
> 123
  2     1      3
> acbvasdsdfasdfwefagda
fdcseaadadfswvsb  a   a f    d g
> 1234567890
 726  591 08 4 3
```

而且在若干次输入后直接得到了flag？

```
Rua!!!
mjg{L+e$hItUpm5a!u_m}iRA3nTT!e0P
```

很明显flag的顺序被打乱了，结合题目名称，应该是根据一个对应关系将flag还原，而这个对应关系应该也是输入输出的对应关系
测试情况如下

```
root@tesla:~# nc 47.98.192.231 25002
> 1234567890zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz
zz3687952zz0z14zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz zzzzzzz  z zz
> zzzzzzzzzz1234567890zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz
14zzzzzzz23z6zz5zz9zzzzz8zzzz70zzzzzzzzzzzzzzzzzz zzzzzzz  z zz
> zzzzzzzzzzzzzzzzzzz1234567890zzzzzzzzzzzzzzzzzzzzzzzzzzzzz
zzzzzzzzzzzzzzzz70z24351z896zzzzzzzzzzzzzzzzzzzz zzzzzzz  z zz
> zzzzzzzzzzzzzzzzzzzzzzzzzzzzzz1234567890zzzzzzzzzzzzzzzzzz
zzzzzzzzzzzzzzzzzzzzzzzzzzzzzz2zz1zz5809z74zzzz36zz zzzzzzz  z zz
> zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz1234567890zzzzzzzzz
zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz36zzzz1zz4528zz7z zzzzzz0  z 9z
> zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz1234567890
zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz9 146573z0 8 z2
> zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz1234567890
zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz2zzzzzzzz1z4zz3  70   96    58
> z
             z
> z
             z
> z
             z
Rua!!!
+ma{Uj$eg5ItLhmpi!PmTuAR_0nT}3e!
```

对应找一下就能知道flag了
最终flag：hgame{jU$t+5lmpL3_PeRmuTATi0n!!}

Misc - 欢迎参加HGame！

题目：

```
欢迎大家参加 HGAME 2020！
来来来，签个到吧～
Li0tIC4uLi0tIC4tLi4gLS4tLiAtLS0tLSAtLSAuIC4uLS0uLSAtIC0tLSAuLi0tLi0gLi4tLS0gLS0tLS0gLi4tLS0gLS0tLS0gLi4tLS4tIC4u
Li4gLS0uIC4tIC0tIC4uLi0t
注：若解题得到的是无hgame{}字样的flag花括号内内容，请手动添加hgame{}后提交。
【Notice】解出来的字母均为大写
```

按照CTF的套路（可能是这样），不知道的明文先丢base64解密一遍，得到一串摩尔斯电码

```
.-- ...-- .-.. -.-. ----- -- . ..--.- - --- ..--.- ..--- ----- ..--- ----- ..--.- .... --. .- -- ...--
```

再对应解密，得到flag括号中的内容（解密时不能使用中文摩尔斯电码加解密工具，因为此工具会先将明文/密文转为Unicode，无法将以上电码直接对应到字符）

```
w3lc0me_to_2020_hgam3
```

之前题目没有【Notice】部分，导致我尝试更改字形提交了很久，最后全改为大写后才提交成功
最终flag：hgame{W3LC0ME_TO_2020_HGAM3}

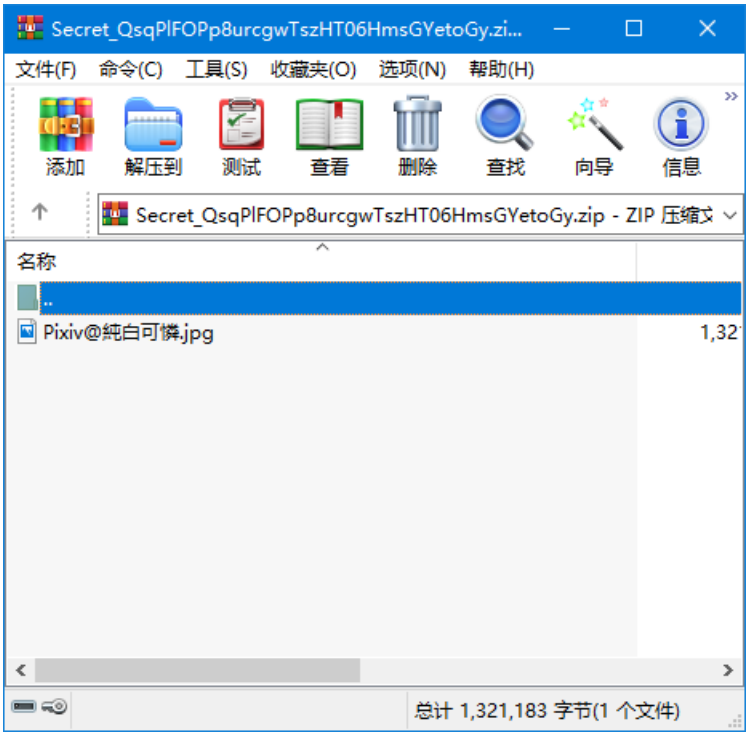
Misc - 壁纸

题目：

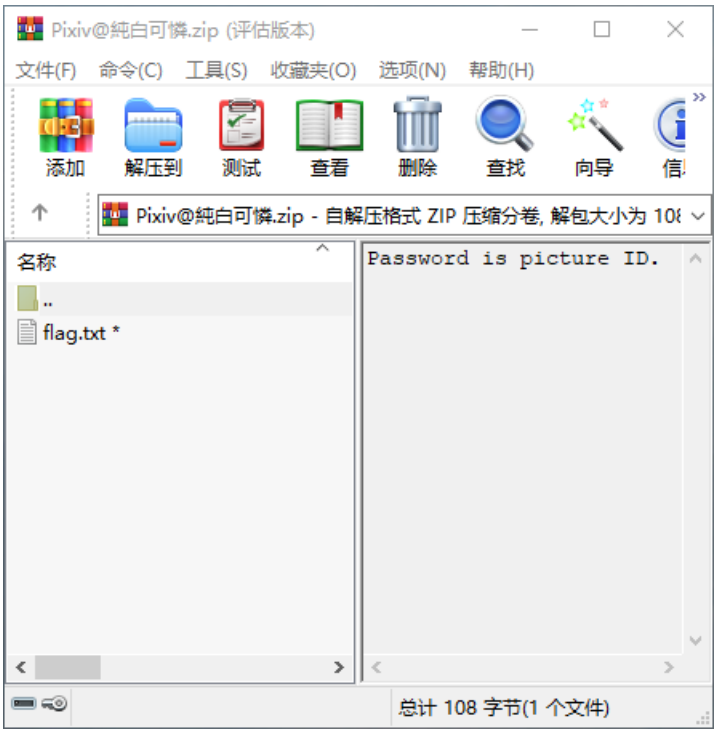
```
某天，ObjectNotFound给你发来了一个压缩包。
“给你一张我的新老婆的壁纸！怎样，好看吗？”
正当你疑惑不解的时候，你突然注意到了压缩文件的名字——“Secret”。
莫非其中暗藏玄机？
```

压缩包放不上来就不放了

看题目名字就知道是个图种了，直接把压缩包中的图片拖出来，重命名为zip后缀，然后打开



发现flag.txt被加密，注释里面说需要找到原图的P站ID才能打开



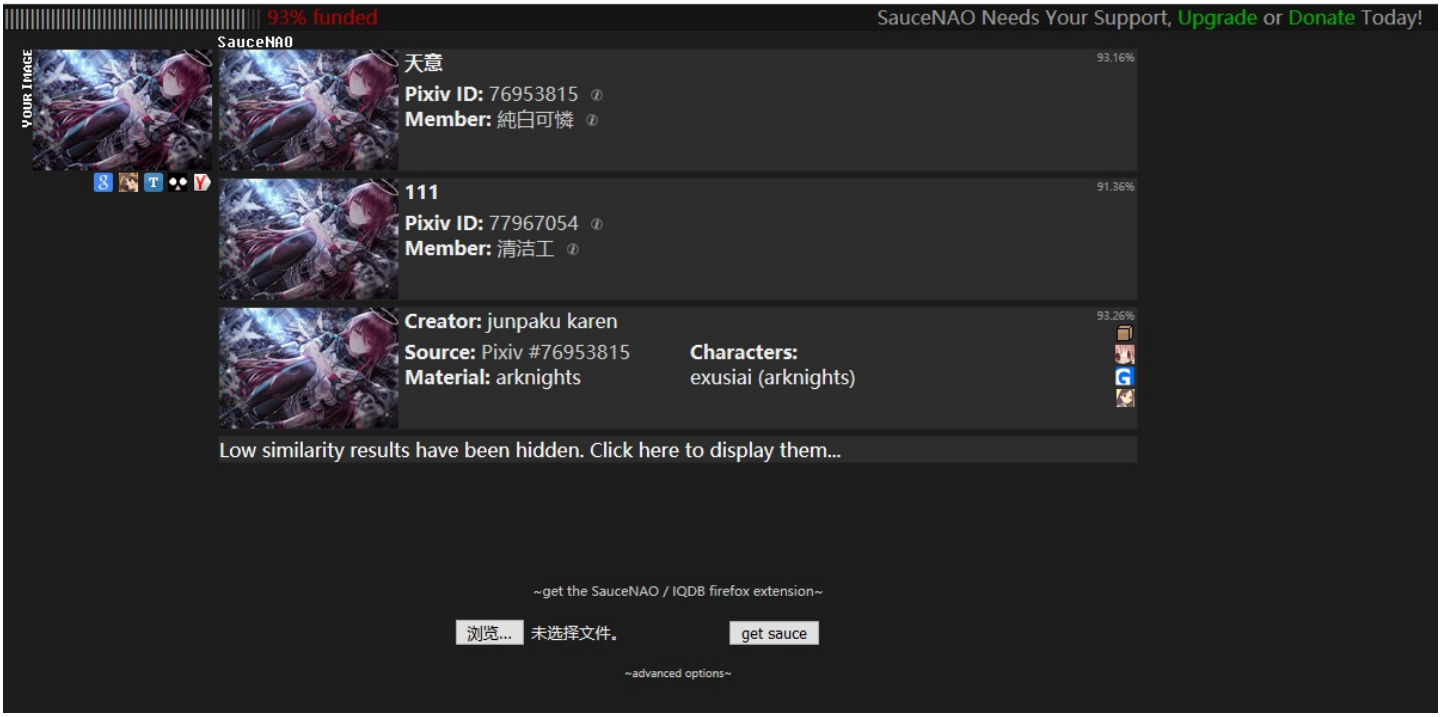
思路一：百度得知P站ID一般为8位数字，那么直接bruteforce应该可行（手上没工具，未实现）

思路二：识图

考虑到识图网站处理图片文件的方式应该和本地的图片查看器方式相同，因此没有做图片与压缩包的分离操作，直接把源文件用来识图了

先丢百度识图上面直接识图，无果

后百度“P站识图”，找到了一个P站以图找ID的网站,查到图片的ID是76953815



打开flag.txt，却并没有看到flag，而是以下内容

```
\u68\u67\u61\u6d\u65\u7b\u44\u6f\u5f\u79\u30\u75\u5f\u4b\u6e\u4f\u57\u5f\u75\u4e\u69\u43\u30\u64\u33\u3f\u7d
```

两个16进制数结合前面的u前缀，猜测以上内容为utf-8编码，而且下划线的16进制utf-8编码为5f，且密文中刚好有多处\u5f，基本确定这是utf-8编码。但是由于缺少解码工具，只能手动解码出flag
解到一半想起百度提交搜索内容时是将中文转为utf-8编码以%xx的形式通过GET提交，遂将flag.txt中的\u替换为%，然后在百度官网地址后面添加以下内容

```
baidu?wd=%68%67%61%6d%65%7b%44%6f%5f%79%30%75%5f%4b%6e%4f%57%5f%75%4e%69%43%30%64%33%3f%7d
```

访问后即可在搜索框位置得到flag
最终flag：hgame{Do_y0u_KnOW_uNiC0d3?}

后记：
utf-8指的并不是只用8位数据表示一个字符，而是指可变长编码以若干个8位（若干个字节）来表示一个字符，解题的时候望文生义了

Web - 鸡尼泰玫

题目：

```
听说你球技高超？
```

之后给出了一个游戏网址，打开之后是一个类似BBTAN的游戏，每一次击球可以获得100分，当总分超过30000分时即可获得flag
游戏菜鸡自然是不会玩游戏的，直接F12，发现是一个js写的小游戏，当中定义了几个游戏中的类

然后在game类当中找到了globalScore和storageScore两个重要参数

```
goodGame() {
  this.globalScore = this.globalScore + this.storageScore;
  clearInterval(this.timer)
  this.context.clearRect(0, 0, this.canvas.width, this.canvas.height)
  this.context.font = '32px Microsoft YaHei'
  this.context.fillStyle = '#000'
  this.context.fillText('CXK，下一关！', 308, 226)
}
finalGame() {
  this.globalScore = this.globalScore + this.storageScore;
  clearInterval(this.timer)
```

尝试用以下命令修改

```
_main.game.storageScore=30000
_main.game.globalScore=30000
```

发现修改之后数据确实变成了30000，但是每次球撞到方块之后分数会变回原始分数
之后尝试在球即将落地时修改参数，成功
最终flag：hgame{j4vASc1pt_w1ll_tel1_y0u_someth1n9_u5efu1?!}

Web - Code World

题目：

```
Code is exciting!
```

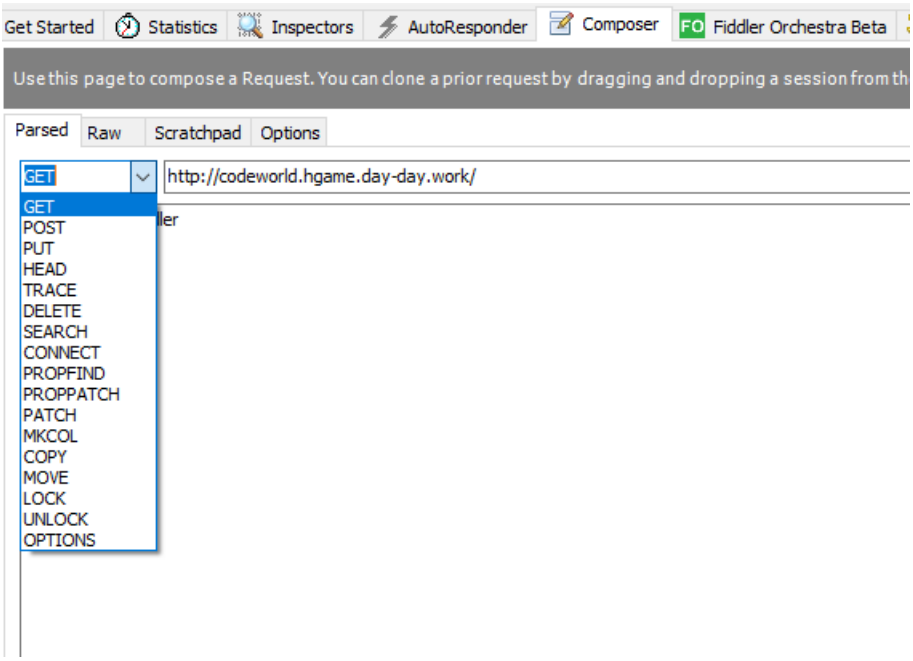
直接访问目标地址，看到了403 Forbidden
感觉挺无从下手的，于是试了一下F12，发现源码里面有这么一行

```
console.log("This new site is building....But our stupid developer Cosmos did 302 jump to this page..F**k!")
```

百度了一下有关302的信息，结果直接找到一篇文章讲CTF题目当中遇到302的处理方法。Linux下使用curl访问网址默认是不会根据302跳转跳到新网址的，遂用curl访问该网址，得到以下结果

```
root@tesla:~# curl http://codeworld.hgame.day-day.work/
    <html>
    <head><title>405 Not Allowed</title></head>
    <body bgcolor="white">
        <center><h1>405 Not Allowed</h1></center>
        <hr><center>nginx/1.14.0 (Ubuntu)</center>
    </body>
    </html>
```

405 Not Allowed说明当前的请求方法不被允许。因为curl不能发送自定义的请求头，所以决定还是用Fiddler的Composer功能试一下
用Fiddler抓了包，意外地发现Fiddler能抓到访问题目网址时的302返回和403返回，且302返回当中的内容正好是curl所得到的结果，说明用Fiddler直接访问题目网址即可。
打开Composer选项卡，填入网址，把所有的请求方法全部试了一遍



发现使用POST方法访问之后得到了以下结果

```
<center><h1>人鸡验证</h1><br><br>目前它只支持通过url提交参数来计算两个数的相加，参数为a<br><br>现在,需要让结果为10</center>
```

尝试直接在请求体部分添加a=10，没有反应；再尝试a=5+5，也没有反应
后来发现审题不仔细，应该是在url当中提交参数。遂在网址后添加?a=10，没有反应；再尝试?a=5%2B5，得到了以下结果（+号经url编码后得到%2B，可在百度搜索框当中尝试）

```
<center><h1>人鸡验证</h1><br><br>目前它只支持通过url提交参数来计算两个数的相加，参数为a<br><br>现在,需要让结果为10<br><h1>The result is: 10</h1><br>hgame{C0d3_1s_s0_S@_sO_C0ol!}</center>
```

最终flag：hgame{C0d3_1s_s0_S@_sO_C0ol!}

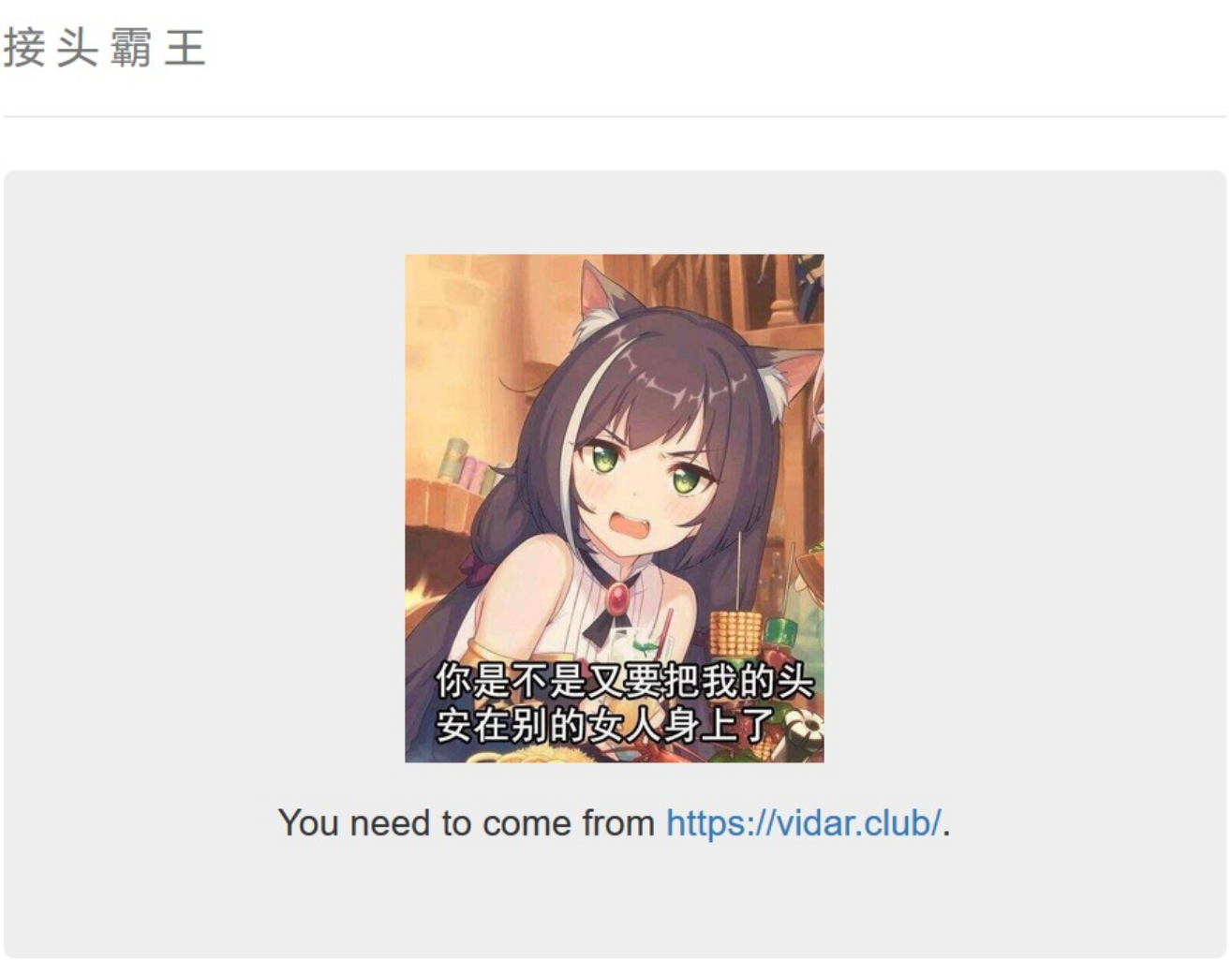
Web - 接 头 霸 王

题目：

```
HGAME Re:Dive 开服啦~
```

直接访问题目地址，得到以下提示

```
You need to come from https://vidar.club/.
```



通过百度得知需要在请求头里面加上

```
Referer: https://vidar.club/
```

于是用Fiddler的Composer补完请求头之后提交，得到以下提示

```
You need to visit it locally.
```

之后基本就是得到提示→百度→改头的过程了，题目中涉及到的修改操作如下

```
You need to come from https://vidar.club/.
Referer: https://vidar.club/

You need to visit it locally.
X-Forwarded-For: 127.0.0.1

You need to use Cosmos Brower to visit.
User-Agent: Cosmos Brower

Your should use POST method :)
```
把请求方法由默认的GET改为POST

```
The flag will be updated after 2077, please wait for it patiently.
If-Unmodified-Since: Fri, 01 Jan 2077 00:00:00 GMT
```

其中最后一条只需要这个时间比响应头中的Last-Modified时间晚即可（相同也可以）
前几条百度上都挺好找的，就是最后一条网上似乎没有明确的介绍，于是就把所有和日期有关的头都试了一遍，找到了结果
最终flag：hgame{W0w!Your_heads_@re_s0_many!}

Web - Cosmos的博客

题目：

```
这里是 Cosmos 的博客，虽然什么东西都还没有，不过欢迎大家！
```

打开网页后内容如下：

```
大茄子让我把 flag 藏在我的这个博客里。但我前前后后改了很多遍，还是觉得不满意。不过有大茄子告诉我的版本管理工具以及 GitHub，我改起来也挺方便的。
```

按F12，没有发现任何有用信息。尝试直接

```
git clone http://cosmos.hgame.n3ko.co/
```

也没有结果

之后直接找了一下CTF中有关git的题目，根据网上的信息，访问

```
http://cosmos.hgame.n3ko.co/.git/HEAD
```

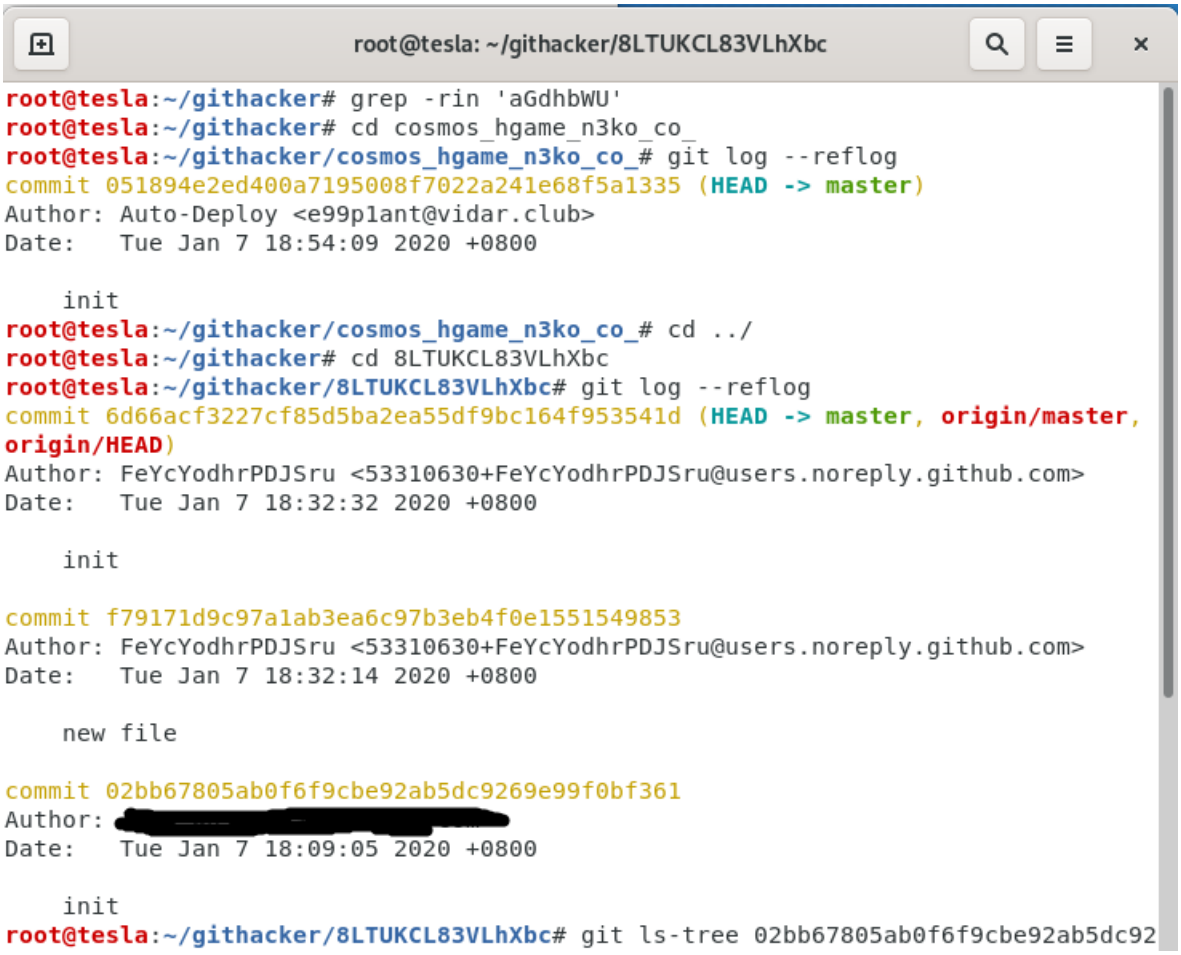
得到以下内容

```
ref: refs/heads/master
```

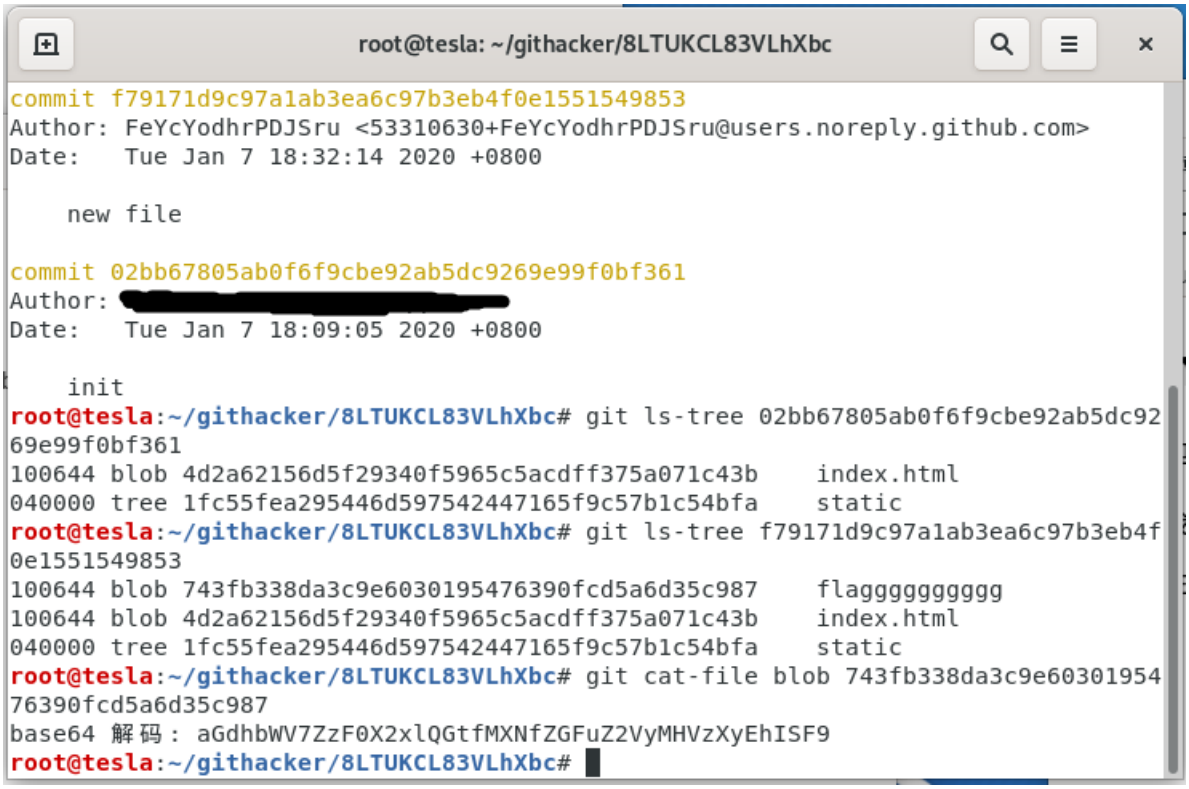
说明该网站存在git泄露。在网上下了一个githacker的脚本，把网站上的git仓库直接下载下来，然后直接目录内搜索hgame，无果
根据网上的操作步骤，尝试查看仓库的历史版本，无果

之后随便翻阅仓库内的文件，意外发现GitHub上的仓库地址

clone到本地后再次尝试查看历史版本，结果与上一次不一样



尝试恢复，得到了flag文件，base64解码即可得到flag



最终flag：hgame{g1t_le@k_1s_danger0us_!!!}

第一次做CTF类的题目，感觉自己还是太菜了，做出来的题目不足1/2，而且用时很长。之后看一下别人的writeup学习一下吧

2020.01.24