HGAME 2019 week-1 writeup

WEB

1.谁吃了我的 flag?

这题真的 真的 做了 好久 好久。。刚开始想着是不是能从 bp 抓到的 http 头中找到些什么，结果发现一无所获。。

然后重新回去读了一下题目，

描述

呜呜呜，Mki一起床发现写好的题目变成这样了，是因为昨天没有好好关机吗T_T hint: 据当事人回忆，那个夜晚他正在用vim编写题目页面，似乎没有保存就关机睡觉去了,现在就是后悔，十分的后悔。

URL http://118.25.111.31:10086/index.html

基准分数 50

当前分数 50

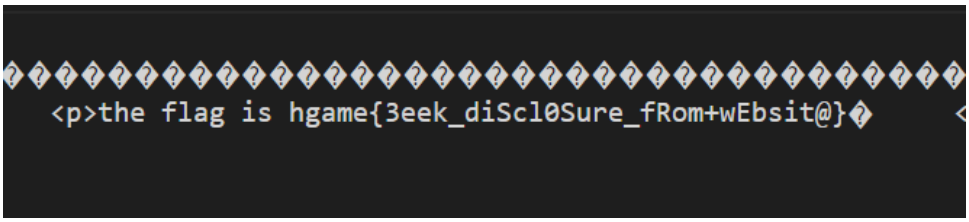
没好好关机啊。。而且还是用 vim，没保存，于是就上百度搜了一下，找到了一道实验吧的题的 wp：

https://blog.csdn.net/wy_97/article/details/76559354

讲了关于的 vim 临时文件，.filename.swp ,于是便访问：

http://118.25.111.31:10086/.index.html.swp

得到了 swp 文件，用 vscode 打开，就看到了 flag



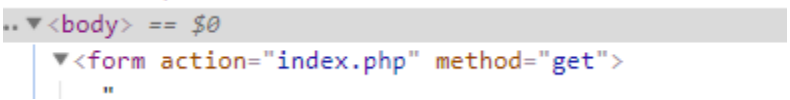
2.换头大作战

打开题目发现有一个表单，这时随意输入些什么如“want”

想要flag嘛： submit

request method is error.I think POST is better

发现题目提示我们需要使用 POST 的方式提交表单，于是我们去找十二姑娘，



将 method 改为 POST 然后重新提交表单即可，结果如下图：

想要flag嘛： submit

https://www.wikiwand.com/en/X-Forwarded-For
only localhost can get flag

这里提示我们使用 X-Forwarded-For 进行 ip 伪装成本地访问，故启动 burpsuite 抓包

将我们抓到的包发去 repeater 进行改包并重发：

```
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8
X-Forwarded-For:127.0.0.1
Cookie: admin=0
```

重发后得到：

```
<br/>https://www.wikiwand.com/en/User_agent<br/>please
use Waterfox/50.0
```

提示我们修改 User-agent 为水狐 2333，于是很显然：

```
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)  Waterfox/50.0
Safari/537.36
```

再重发，得到：

```
<br/>https://www.wikiwand.com/en/HTTP_referer<br/>the
requests should referer from www.bilibili.com
```

提示我们需要从 bilibili 来访问题目，于是：

```
mage/webp,image/apng,*/*;q=0.8
Referer: www.bilibili.com
Accept-Encoding: gzip, deflate
```

得到：

```
<br/>https://www.wikiwand.com/en/HTTP_cookie<br/>you
are not admin
```

然后直接改 cookie：

```
X-Forwarded-For:127.
Cookie: admin=1
```

拿到 flag

```
<br/>hgame{hTTp_HeaDeR_iS_Ez}
```

3.代码审计

```
<?php
error_reporting(0);
include("flag.php");

if(strpos("vidar",$_GET['id'])!==FALSE)
    die("<p>干巴爹</p>");

$_GET['id'] = urldecode($_GET['id']);
if($_GET['id'] === "vidar")
{
    echo $flag;
}
highlight_file(__FILE__);
?>
```

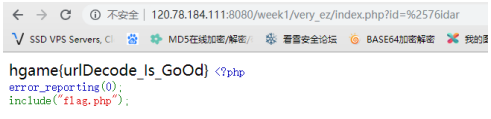
分析源码可知：

1. strpos 函数判断我们提交的变量 id 的值在字符串”vidar”中第一次出现的位置。
2. strops 的返回值要不为 0，即 id 在 vidar 中的位置不能为首位置。
3. 源码中对变量进行了一次 urldecode，而 url 实际上已经经过一次 decode，所以这里发生了 url 的二次解码。
4. id 最终的值必须严格等于 vidar 则可以获取 flag。

因此就想到，将字符‘v’进行二次 url 编码，首先‘v’十六进制为 0x76,故 url 一次编码为%76,然后利用工具将%76 二次编码为%2576。

于是构造 http://120.78.184.111:8080/week1/very_ez/index.php?id=%2576idar

即可得到 flag：



5.can u find me?

看到题目提示：

Description

为什么不问问神奇的十二姑娘和她的小伙伴呢

嗯。。十二姑娘，刚开始我还傻傻的去百度上搜了十二姑娘是啥 2333.然后突然想起来，不就是 F12 吗....

果断打开 F12，

```
  <p>the gate has been hidden</p>
  <p>can you find it? xixixi</p>
  <a href="f12.php"></a>
</body>
html>
```

进入/f12.php

让我 post 密码，粗粗地试了一下几个弱口令，发现都不行，于是便启动 burpsuite

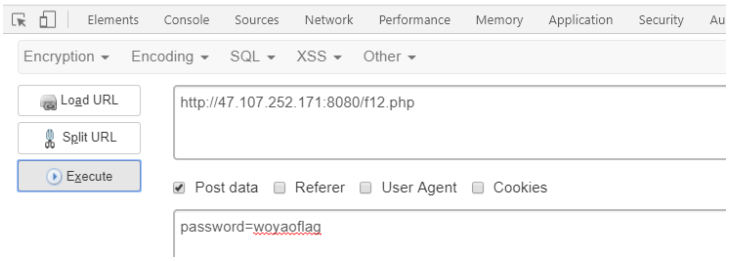
```
Server: nginx/1.15.8
Date: Sat, 26 Jan 2019 03:45:01 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
X-Powered-By: PHP/7.2.14
password: woyaoflag
Content-Length: 242
```

直接在响应头中发现了 password，于是便用 hackbar post 过去。

please post password to me! I will open the gate for you!

right!

click me to get flag



得到了一个链接，果断点进去。

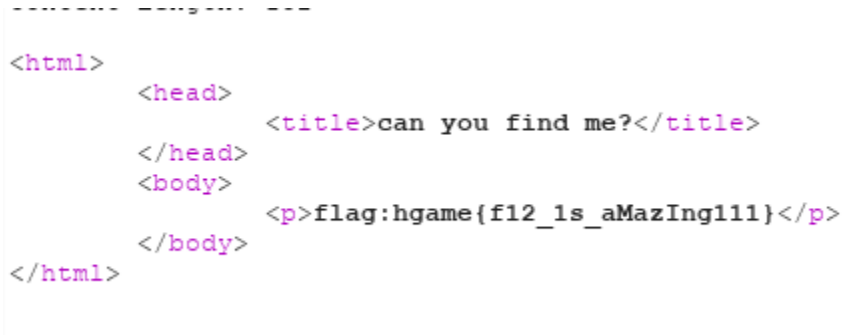
aoh,your speed is sososo fast,the flag must have been left in somewhere

太快了？想了一下，可能前面错过了什么吧。。

于是回到上一个页面，打开 F12 看了一眼，

发现源码中的链接和我点击后跳转的页面是不同一个链接，于是猜测这里被跳转了。

打开 burpsuite，打算将跳转前的 http 包截下来。



看一下源码，就得到了 flag。

RE

（RE 啥都不会= =，就现学现做，照着网上的某些类似题的 wp 混了点分）

1. brainfxxker

这题刚开始看的时候有点头晕，感觉那些代码太抽象了，后来搜到了这个：

https://www.jianshu.com/p/4abeda905abe

当读到`+`的时候，将所在车厢里的数字加一
当读到`-`的时候，将所在的车厢里的数字减一
当读到`>`的时候，跑到后一个车厢去
当读到`<`的时候，跑到前一个车厢去
当读到`[`的时候，如果该车厢里面的数字**为0**，则跳去执行下一个`]`之后的程序内容
当读到`]`的时候，如果该车想里面的数字**不为0**，则跳去执行上一个`[`之后的程序内容
当读到`.`的时候，将所在车厢里面的数字翻译成ASCII字符，显示在你的屏幕上
当读到`,`的时候，从等待使用者输入一个ASCII字符，转码成数字写进所在车厢里

可以说特别的形象了

然后就去一点点的分析代码，大概看懂代码的意思之后，注意到了题目给的说明：

补充说明:

判定答案是否正确的是 Notice 2，即“不执行 [+.] 这个部分”，不要单纯看有没有输出 orz

不执行[+.]部分，也就是说我们所输入并不会有相应的输出给我们，故只要让程序不满足执行[+.]的条件即可。

```
,           等待使用者输入ascii字符，转码成数字放入第一节车厢
>           去下一节车厢
++++++++++  第二节车厢+10

[           第二节车厢值为10，所以继续执行下面的
<           回到第一节车厢
----------  第一节车厢的值-10
>           到第二节车厢
-           第二节车厢的值-1，现在为9
]           总共循环执行十次，在第十次时第二节车厢为0，此时，第一节车厢的值已被-100
<           回到第一节车厢
++          第一节车厢的值+2
[           input:b
+
.
]
```

第一段如图所示，只要我们所输入的字符的 ASCII 码为 98 即可，以此类推，得到所有的正确输入

连起来即为 flag：hgame{bR4!NfUcK}

2. HelloRe

签到题吧。。啥也不会的我也能做。丢进 IDA 里到 main 函数 F5 一下就出来了。。

```
{
  char s[8]; // [rsp+0h] [rbp-30h]
  __int64 v5; // [rsp+8h] [rbp-28h]
  __int64 v6; // [rsp+10h] [rbp-20h]
  __int64 v7; // [rsp+18h] [rbp-18h]
  unsigned __int64 v8; // [rsp+28h] [rbp-8h]

  v8 = __readfsqword(0x28u);
  *(_QWORD *)s = 0LL;
  v5 = 0LL;
  v6 = 0LL;
  v7 = 0LL;
  puts("Please input your key:");
  fgets(s, 32, stdin);
  s[strlen(s) - 1] = 0;
  if ( !strcmp(s, "hgame{Welc0m3_t0_R3_World!}") )
    puts("success");
  else
    puts("failed..");
  return 0LL;
}
```

5. Pro 的 Python 教室(一)

题目就是比较我们的输入和 base64 以及 base32 加解密后的值是否相等。

```
enc1 = 'hgame{'
enc2 = 'SGVyZV8xc18zYXN5Xw=='
enc3 = 'Pyth0n}'
```

```
first = raw_input()
if first == enc1:
    pass
```

```
secend = raw_input()
secend = base64.b64encode(secend)
if secend == enc2:
    pass
else:
```

```
third = raw_input()
third = base64.b32decode(third)
if third == enc3:
    pass
```

enc1 显然没有任何的处理，于是 first==hgame{

enc2 显然是经过 base64 加密过后的字符串，于是我们只要将 enc2 解密即可。

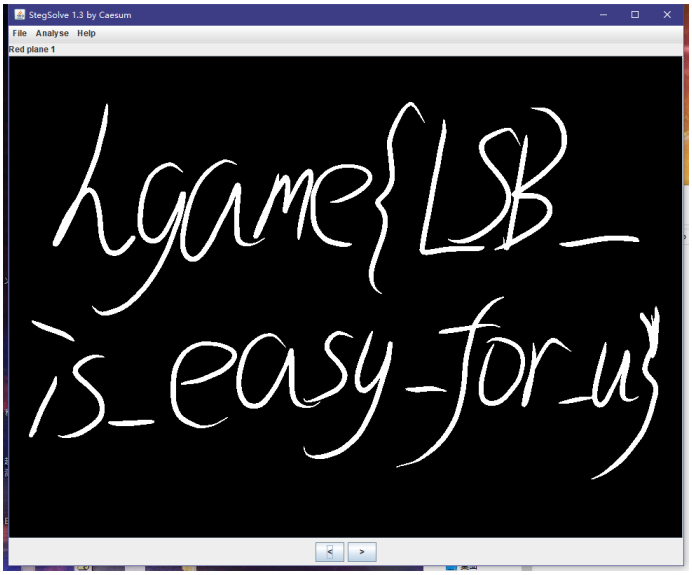
enc3 不清楚为啥。。本来应该是经过 base32 解密的值，third 应该是一串 base32 编码吧。但是 flag 却是 enc3 的值。。

连起来就得到了 flag　hgame{Here_1s_3asy_Pyth0n}

MISC

1. Hidden Image in LSB

压缩包打开是一张图和一个 python 代码，看到这张图，于是就果断的用隐写神器 Stegsolve 看了一下。



直接得到了 flag。

2. 打字机

全靠猜吧。。。首先前面五个一定是 hgame，后面相应的对照出来，不管对错先写出所有，然后看了一下大概是，

My_vi0let_tyPewRiter，我的紫罗兰打字机？？

得到 flag：`hgame{My_vi0let_tyPewRiter}`

3. Broken Chest

下载题给压缩包，解压发现损坏报错，于是上网搜索相关信息。

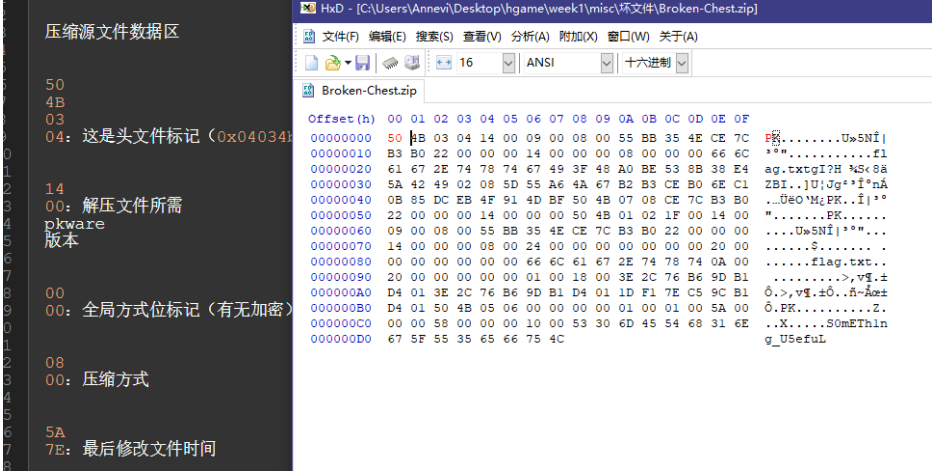
Google 到了这篇文章：https://www.anquanke.com/post/id/86211

第七条讲的就是 zip 文件缺少文件头或文件尾造成损坏，于是便在 linux 下用 binwalk 分析了一下压缩包文件，

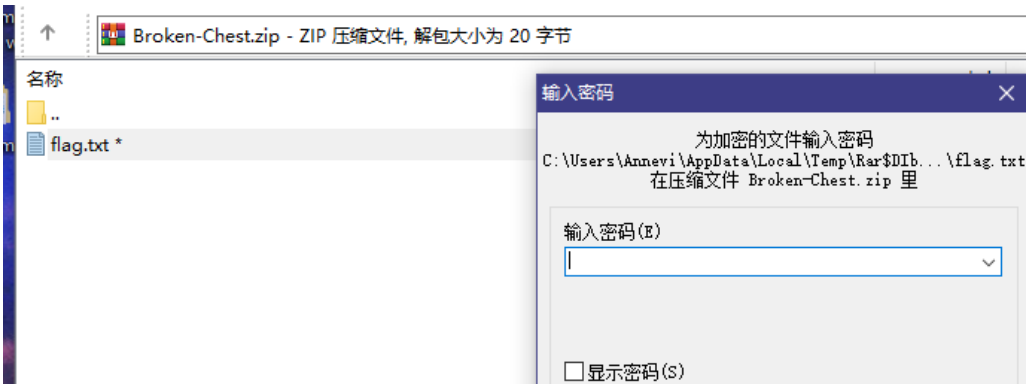
发现确实缺失了文件头。于是又查询相关资料：

https://blog.csdn.net/fox_wayen/article/details/78155064

用 HDX 打开 zip 文件，修改文件头：

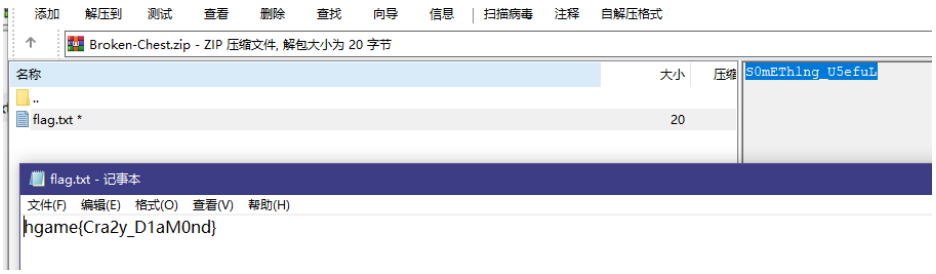


即可成功打开 zip 文件，但是又发现 zip 文件被加密了：



这时候发现压缩包的描述中有 S0mEThlng_U5efuL

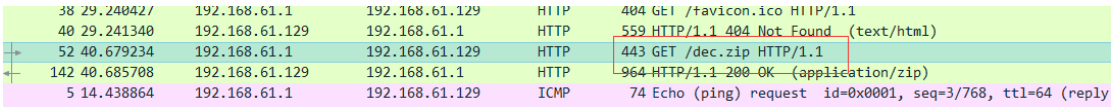
果断猜测这可能就是压缩包密码吧。于是输入成功得到 flag

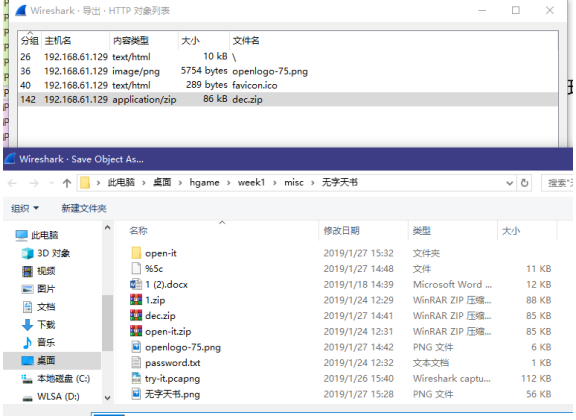


4.无字天书

这道题有一个之前没有见过的文件格式：*.pcapng，于是又搜了一下，发现这是数据包的文件格式，可以用 wireshark 进行分析。

找到了一个很可疑的文件，于是将其保存下来。

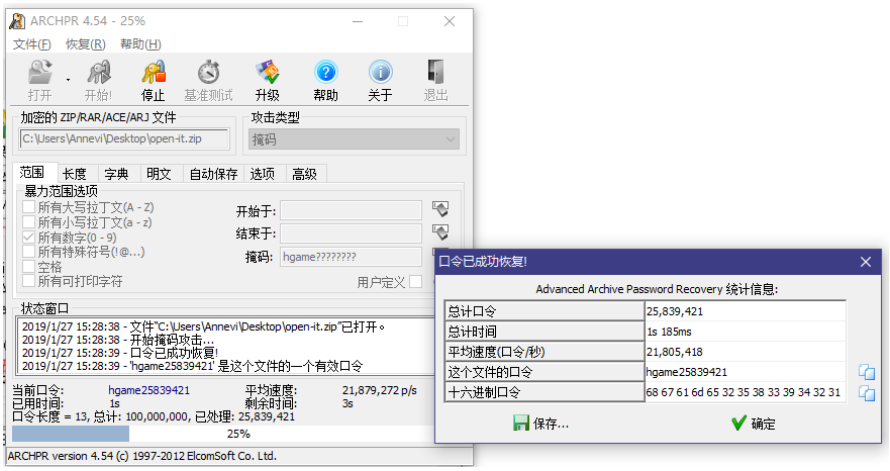




打开压缩包，发现里面是一个 password 的文本文件，内容为：hgame********

然后还有一个压缩包，是加密的。

于是便猜想，压缩包密码肯定和这个 hgame 有关，google 了一下，发现了压缩包的掩码攻击，用到的工具是 APR，



构造好掩码，然后选择好类型，很快密码就出来了。 Hgame25839421

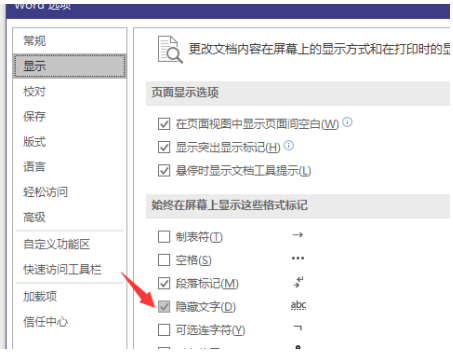
用密码打开加密的压缩包，是一张小姐姐的图片。之前做过不少隐写，于是就按套路来，拖进 HDX 看了一眼

```
24 00 00   ....øt2NÝÄÍ"ç$..
00 20 00   '/....$.......  .
00 20 00   ......1.docx.. .
AE D4 01   ........Æ,Yšø®Ô.
AE D4 01   ·;"fœ³Ô.J7Ý.ø®Ô.
00 00 00   PK..........X...
```

不小心就看到了什么不得了的东西，于是就将小姐姐变成了压缩包，1.jpg=>1.zip

里面是一个 word 文档，打开它，发现里面啥也没有。。这时突然想起了之前在 bugku 上做过的一道 word 里面隐藏文字的题，于是就去选项里面勾选了显示隐藏文字，拿到 flag。

hgame{59d28413e36019861498e823f3f41406}



Crypto

1. Mix

描述

--.../....-/....-/-.../--.../...--/...../..-./-..../-../-..../..-./--.../----./....-/....-/--.../.----/-..../-.../--.../-.../-..../..---/...../.----/-..../-..../...--/....-/...--/-----/-..../...../--.../-.. So easy

URL http://example.com

看到是摩斯电码，于是上网找在线工具转换成字符。

```
744B735F6D6F7944716B7B62516634306 57D
```

观察发现，这里面的字符都限制在了 F 以内，于是猜测这是 16 进制的编码，先手动转换了一下，得到：

```
tKs_moyDqk{bQf40e}
```

（后来才知道，原来这就是 base16 加密。。）发现这串东西很不符合 flag 的格式，而且还有两个大括号的存在，于是便猜测运用了栅栏加密。神器启动，CTFCrakTools，解码得：

```
结果:
得到因数(排除1和字符串长度):
 2 3 6 9

第1栏: tsmyq{Q4eK_oDkbf0}
第2栏: t_ykQ0KmD{fesoqb4}
第3栏: tyQKDfsq4_k0m{eob}
第4栏: tkK{sb_Qmfo4y0Deq}
```

观察发现第一栏最为符合 flag 的格式，于是将第一栏保存下来，发现格式一致，而且花括号没变，所以猜测是凯撒加密

```
结果:
utnzr{R4fL_pElcgO}
vuoas{S4gM_qFmdhO}
wvpbt{T4hN_rGneiO}
xwqcu{U4iO_sHofjO}
yxrdv{V4jP_tIpgkO}
zysew{W4kQ_uJqhlO}
aztfx{X4lR_vKrimO}
baugy{Y4mS_wLsjnO}
cbvhz{Z4nT_xMtkoO}
dcwia{A4oU_yNulpO}
edxjb{B4pV_zOvmqO}
feykc{C4qW_aPwnrO}
gfzld{D4rX_bQxosO}
hgame{E4sY_cRyptO}
ihbnf{F4tZ_dSzquO}
```

得到了很多结果，一眼就看见了 flag。

2. perfect_secrecy!

看到题目提示我们 OTP 这种加密方式，于是便 google 了一下，在 github 上找到了一个恢复的脚本，

```
#!/usr/bin/env python
import string
import collections
import sets

# XORs two string
def strxor(a, b):     # xor two strings (trims the longer input)
    return "".join([chr(ord(x) ^ ord(y)) for (x, y) in zip(a, b)])

# 10 unknown ciphertexts (in hex format), all encrpyted with the same key
c1 = "daaa4b4e8c996dc786889cd63bc4df4d1e7dc6f3f0b7a0b61ad48811f6f7c9bfabd7083c53ba54"
c2 = "c5a342468c8c7a88999a9dd623c0cc4b0f7c829acaf8f3ac13c78300b3b1c7a3ef8e193840bb"
c3 = "dda342458c897a8285df879e3285ce511e7c8d9afff9b7ff15de8a16b394c7bdab920e7946a05e9941d8308e"
c4 = "d9b05b4cd5ce7c8f938bd39e24d0df191d7694dfeaf8bfbb56e28900e1b8dff1bb985c2d5aa154"
c5 = "d9aa4b00c88b7fc79d99d38223c08d54146b88d3f0f0f38c03df8d52f0bfc1bda3d7133712a55e9948c32c8a"
c6 = "c4b60e46c9827cc79e9698936bd1c55c5b6e87c8f0febdb856fe8052e4bfc9a5efbe5c3f57ad4b9944de34"
c7 = "d9aa5700da817f94d29e81936bc4c1555b7b94d5f5f2bdff37df8252ffbecfb9bbd7152a12bc4fc00ad7229090"
c8 = "c4e24645cd9c28939a86d3982ac8c819086989d1fbf9f39e18d5c601fbb6dab4ef9e12795bbc549959d9229090"
c9 = "d9aa4b598c80698a97df879e2ec08d5b1e7f89c8fbb7beba56f0c619fdb2c4bdef8313795fa149dc0ad4228f"
c10 = "cce25d48d98a6c8280df909926c0de19143983c8befab6ff21d99f52e4b2daa5ef83143647e854d60ad5269c87"
c11 = "d9aa4b598c85668885df9d993f85e419107783cdbee3bbba1391b11afcf7c3bfaa805c2d5aad42995ede2cdd82977244"
c12 = "e1ad40478c82678995df809e2ac9c119323994cffbb7a7b713d4c626fcb888b5aa920c354be853d60ac5269199"
c13 = "c4ac0e53c98d7a8286df84936bc8c84d5b50889aedfebfba18d28352daf7cfa3a6920a3c"
c14 = "d9aa4f548c9a609ed297969739d18d5a146c8adebef1bcad11d49252c7bfd1f1bc87152b5bbc07dd4fd226948397"
c15 = "c4a40e698c9d6088879397d626c0c84d5b6d8edffbb792b902d49452ffbec6b6ef8e193840"
c16 = "c5ad5900df8667929e9bd3bf6bc2df5c1e6dc6cef6f2b6ff21d8921ab3a4c1bdaa991f3c12a949dd0ac5269c"
c17 = "c2967e7fc59d57899d8bac852ac3c866127fb9d7f1e5b68002d9871cccb8c6b2aa"
ciphers = [c1, c2, c3, c4, c5, c6, c7, c8, c9, c10,c11,c12,c13,c14,c15,c16,c17]
# The target ciphertext we want to crack
target_cipher = "32510ba9babebbbefd001547a810e67149caee11d945cd7fc81a05e9f85aac650e9052ba6a8cd8257bf14d13e6f0a803b54fd
e9e77472dbff89d71b57bddef121336cb85ccb8f3315f4b52e301d16e9f52f904"

# To store the final key
final_key = [None]*150
# To store the positions we know are broken
known_key_positions = set()

# For each ciphertext
for current_index, ciphertext in enumerate(ciphers):

    counter = collections.Counter()
    # for each other ciphertext
    for index, ciphertext2 in enumerate(ciphers):
```

```
        if current_index != index: # don't xor a ciphertext with itself
            for indexOfChar, char in enumerate(strxor(ciphertext.decode('hex'), ciphertext2.decode('hex'))): # Xor the two ciphertexts
                # If a character in the xored result is a alphanumeric character, it means there was probably a space character in one of the plaintexts (we don't know which one)
                if char in string.printable and char.isalpha(): counter[indexOfChar] += 1 # Increment the counter at this index
    knownSpaceIndexes = []

    # Loop through all positions where a space character was possible in the current_index cipher
    for ind, val in counter.items():
        # If a space was found at least 7 times at this index out of the 9 possible XORS, then the space character was likely from the current_index cipher!
        if val >= 7: knownSpaceIndexes.append(ind)
    #print knownSpaceIndexes # Shows all the positions where we now know the key!

    # Now Xor the current_index with spaces, and at the knownSpaceIndexes positions we get the key back!
    xor_with_spaces = strxor(ciphertext.decode('hex'),' '*150)
    for index in knownSpaceIndexes:
        # Store the key's value at the correct position
        final_key[index] = xor_with_spaces[index].encode('hex')
        # Record that we known the key at this position
        known_key_positions.add(index)

# Construct a hex key from the currently known key, adding in '00' hex chars where we do not know (to make a complete hex string)
final_key_hex = ''.join([val if val is not None else '00' for val in final_key])
# Xor the currently known key with the target cipher
output = strxor(target_cipher.decode('hex'),final_key_hex.decode('hex'))
# Print the output, printing a * if that character is not known yet
print ''.join([char if index in known_key_positions else '*' for index, char in enumerate(output)])

'''''
Manual step
'''
# From the output this prints, we can manually complete the target plaintext from:
# The secuet-mes*age*is: Wh** usi|g **str*am cipher, nev***use th* k*y *ore than onc*
# to:
# The secret message is: When using a stream cipher, never use the key more than once

# We then confirm this is correct by producing the key from this, and decrpyting all the other messages to ensure they make grammatical sense
target_plaintext = "The secret message is: When using a stream cipher, never use the key more than once"
print target_plaintext
key = strxor(target_cipher.decode('hex'),target_plaintext)
for cipher in ciphers:
    print strxor(cipher.decode('hex'),key)
```

将密文导入恢复明文，最终分析得到 flag。

3.Base 全家

要瞎了。。

既然题目告诉我们是 base 全家，那么就是对那一堆东西进行各种 base 编码的尝试，base64 和 base32 长得挺像，但是 base16 一眼便可看出区别(大写字母与数字组成)，看见末尾有“=”的，就在 base64 与 32 里面尝试，最终经过 20 次的反复解码.

(64->64->16->16->16->32->16->32->64->16->64->16->16->16->16->32->64->64->64->32)

得到了一个叫 base58 的东西（之前还真没听说过。。）

```
base58 : 2BAja2VqXoHi9Lo5kfQZBPjq1EmZHGEudM5JyDPREPmS3CxrpB8BnC
```

上百度搜不到相关的 decoder，上 Google 上搜到一个：

https://www.browserling.com/tools/base58-decode 解开，便得到了 flag。(弄得我头晕