CTF中的压缩包

原创

小常吃不下了 于 2021-07-26 16:05:37 发布 777 收藏 7

分类专栏： 密码箱 文章标签： unctf 信息压缩 密码学

本文链接：https://blog.csdn.net/weixin_52620919/article/details/119103460

文章目录

1. 压缩包隐写

```
实际上压缩包本身并不具备隐藏信息的功能，但由于在CTF竞赛中，经常出现压缩包与隐写术结合在一起的题目，
所以我们需要掌握在CTF竞赛中有关压缩包的题目的常见考察方向及分析手段。
```

2. CTF中常见的压缩包套路（这些套路也不一定是单独出现，大多数情况都是组合出现的）

（1） 利用进制转换隐藏信息

（2） 作为冗余信息或隐藏信息藏在其他文件中，一般是图片

（3） 简单密码爆破

（4） 字典爆破/掩码攻击

（5） 伪加密

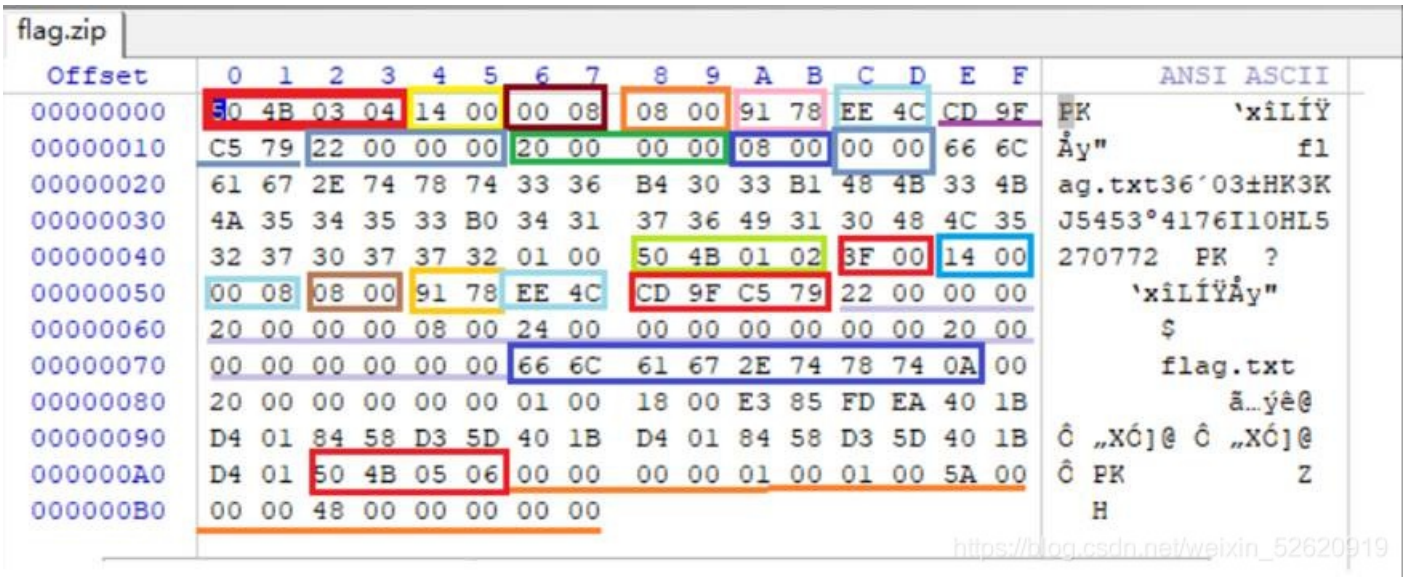
（6） 明文攻击

（7） CRC32碰撞

（8） 文件修复

（9） 冗余信息拼接

（10）注释隐藏密码4. ZIP文件格式



ZIP文件一般分为 **数据区** 与 **目录区**。

数据区的开头标识为 **504B0304**，而目录区开始的标识为 **504B0102**，

以下两图以上面图片显示的压缩包为例，对文件格式进行**解析**。

HEX值	在ZIP文件中代表的信息
数据区文件格式	
504B0304	ZIP文件的文件标识头，为固定值
1400	解压文件所需 pkware最低版本
0008	全局方式位标记，也叫通用比特标志位。（数据区加密标志位，一般从左往右前两个字节决定是否加密，若为奇数表示加密，若为偶数，表示未加密）
0800	表示加密的方式
9178	文件最后修改时间
EE4C	文件最后修改日期
CD9FC579	CRC32冗余校验码值（注意，读取应从右往左，即0X79C59FCD）

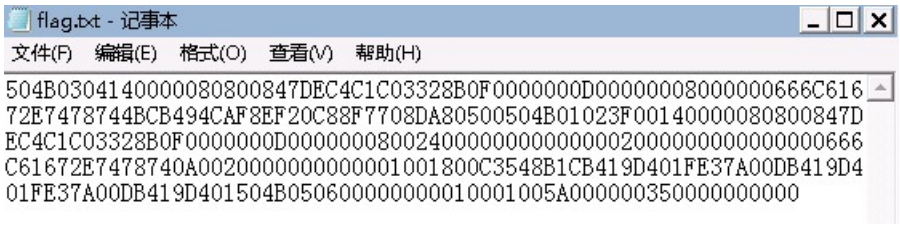
22000000	压缩后大小（值为0X00000022）
20000000	压缩前大小（值为0X00000020）
0800	文件名长度
0000	扩展记录长度

目录区文件格式

504B0102	目录区文件标识头，长度固定
3F00	压缩所用的 pkware 版本
1400	解压所需 pkware 的最低版本
0008	通用位标记，目录区加密标志位。
0800	压缩方法
9178	文件最后修改时间
EE4C	文件最后修改日期
CD9FC579	CRC32 冗余校验码值（注意，读取应从右往左，即 0X79C59FCD）
22000000	压缩后的大小
20000000	压缩前的大小
0800	文件名长度
0024	扩展域长度
0000	文件注释长度
0000	文件开始位置的磁盘编号
0000	内部文件属性
20000000	外部文件属性
00000000	本地文件头部的相对位移
666C61672E7478740A	被压缩的文件名
504B0506	目录结束标识，后面常有 18 位的冗余数据来存放一些目录有关的数据，总长度一般为 22 位

一，利用进制转换隐藏信息

1. 打开下载的文件，进入demo_1,浏览flag.txt。



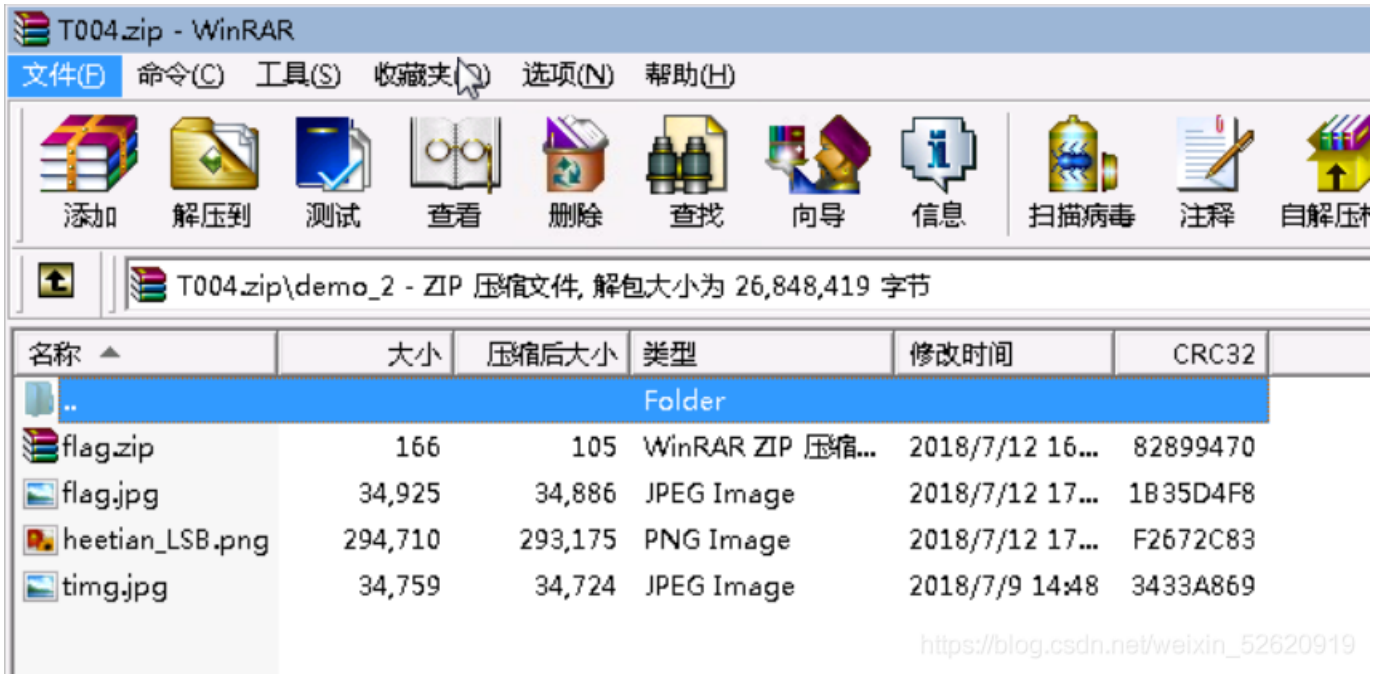
2. 分析字符串就会发现，这一长串字符串是十六进制字符串，尝试十六进制解码，这里我使用的是Python进行解码，虽然输出了一堆乱码，但是我们还是能看见存在敏感字符“flag.txt”。

3. 其实根据字符串的开头“504B0304”就可以判断这串符串是ZIP压缩文件的十六进制值。（这个涉及到ZIP文件格式的详解，会在后面的伪加密中详细解释，只需记住这是ZIP文件的文件标识头，为固定值。）
4. 判断为压缩包后，将其保存为ZIP文件。可以编写脚本进行十六进制解码后保存为ZIP文件，也可以直接使用winhex，这里我使用winhex进行保存。打开winhex后，新建一个空文件，将十六进制字符串复制，右键点击winhex的十六进制数据区，选择“Edit”->“Clipboard Data”->“Paste”->“ASCII hex”，保存为ZIP文件即可，如下图。

5. 保存后就可以正常解压了。

二，作为冗余信息隐藏在其他文件中，将压缩包提取出来。

1. 简单的隐藏在图片后面，即制作图种。（打开文件夹demo_2，其中flag.jpg为已经隐藏了压缩包的图片，timg.jpg为原图片，heetian_LSB.png为使用了LSB算法隐藏压缩包的图片）



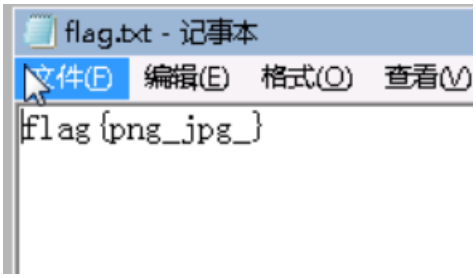
如何实现：使用windows的copy命令即可

```
C:\Users\Administrator\Desktop\T004\demo_2>copy /b timg.jpg + flag.zip flag.jpg
timg.jpg
flag.zip
已复制        1 个文件。
```

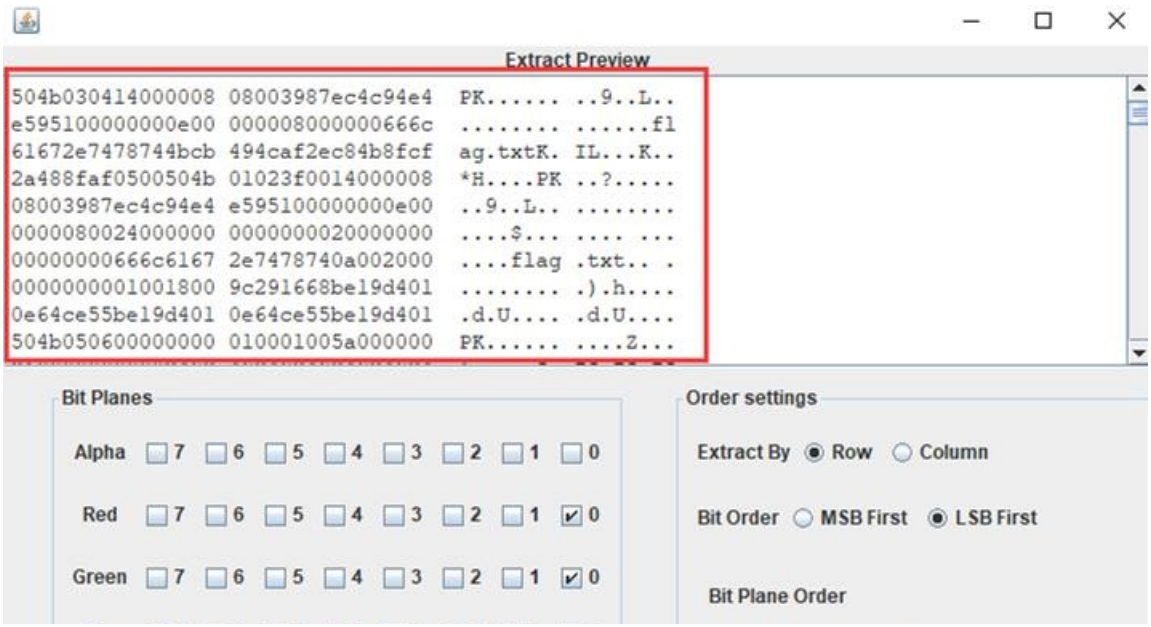
2. 使用winhex打开“flag.jpg”，就会发现在jpg图片的结束标识“FFD9”之后，就是ZIP文件的文件标识“504B0304”，而且可以看到flag.txt字样。

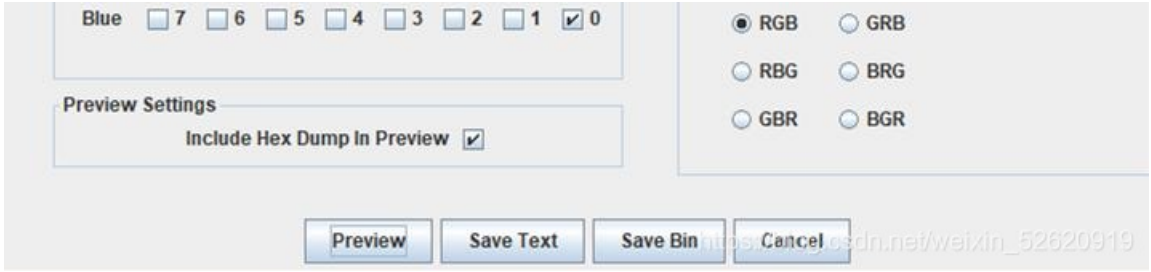
```
00008760  08 B2 50 91 47 E1 D3 DF  88 CC 8E D4 9A 5E A6 C3   ²P'GáÓß^ÌŽÔš^¦Ã
00008770  37 A7 FD 4C 71 80 63 A1  29 54 F8 6B C5 95 F7 7D   7§ýLq€c¡)TøkÅ•÷}
00008780  B5 E6 59 3E 39 D3 8E 49  BB 19 C5 93 5D 7D B4 87   µæY>9ÓŽI» Å"]}´‡
00008790  FC EA DC 35 F1 12 1A 4B  75 25 BC EA B3 91 E1 6F   üêÜ5ñ  Ku%¼ê³'áo
000087A0  5C AD 1F 7D 54 FF 00 3F  E7 56 09 40 90 BA 98 2D   \- }Tÿ ?çV @ °˜-
000087B0  F0 5F FD F6 D5 5E FF 00  9D 49 39 63 09 42 4C 57   ð_ýöÕ^ÿ  I9c BLW
000087C0  C9 A6 08 72 7F FF D9 50  4B 03 04 14 00 00 08 08   É¦ r ÿÙPK
000087D0  00 39 87 EC 4C 94 E4 E5  95 10 00 00 00 0E 00 00    9‡ìL"äå•
000087E0  00 08 00 00 00 66 6C 61  67 2E 74 78 74 4B CB 49        flag.txtKËI
000087F0  4C AF 2E C8 4B 8F CF 2A  48 8F AF 05 00 50 4B 01   L¯.ÈK Ï*H ¯  PK
00008800  02 3F 00 14 00 00 08 08  00 39 87 EC 4C 94 E4 E5    ?        9‡ìL"äå
00008810  95 10 00 00 00 0E 00 00  00 08 00 24 00 00 00 00   •           $
00008820  00 00 00 20 00 00 00 00  00 00 00 66 6C 61 67 2E                flag.
00008830  74 78 74 0A 00 20 00 00  00 00 00 01 00 18 00 9C   txt             œ
00008840  29 16 68 BE 19 D4 01 0E  64 CE 55 BE 19 D4 01 0E   ) h¾ Ô  dÎU¾ Ô
00008850  64 CE 55 BE 19 D4 01 50  4B 05 06 00 00 00 00 01   dÎU¾ Ô PK
00008860  00 01 00 5A 00 00 00 36  00 00 00 00 00            Z   6
```

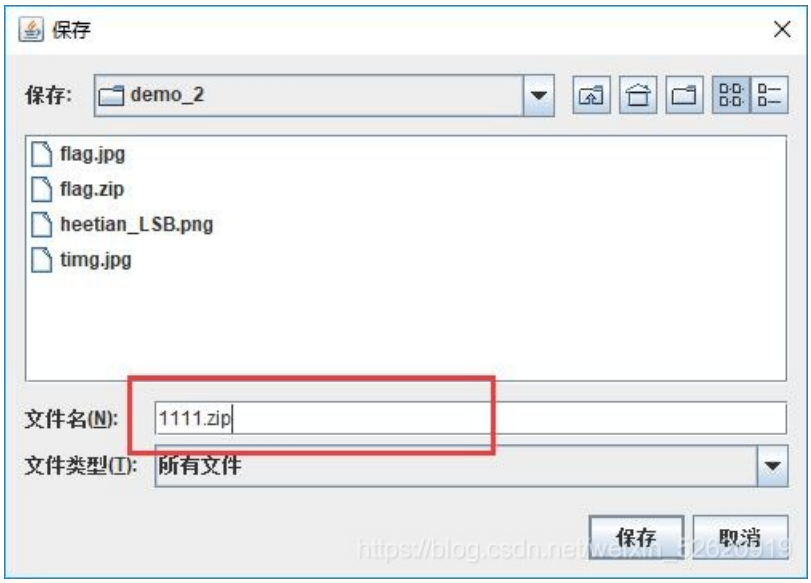

3. 解决方式：直接将图片后缀名改为zip，即可解压。



4. 若是使用隐写算法隐藏在图片中，则需要借助相应的工具，这里以LSB隐写算法为例。
使用StegSolve找到隐藏的信息。



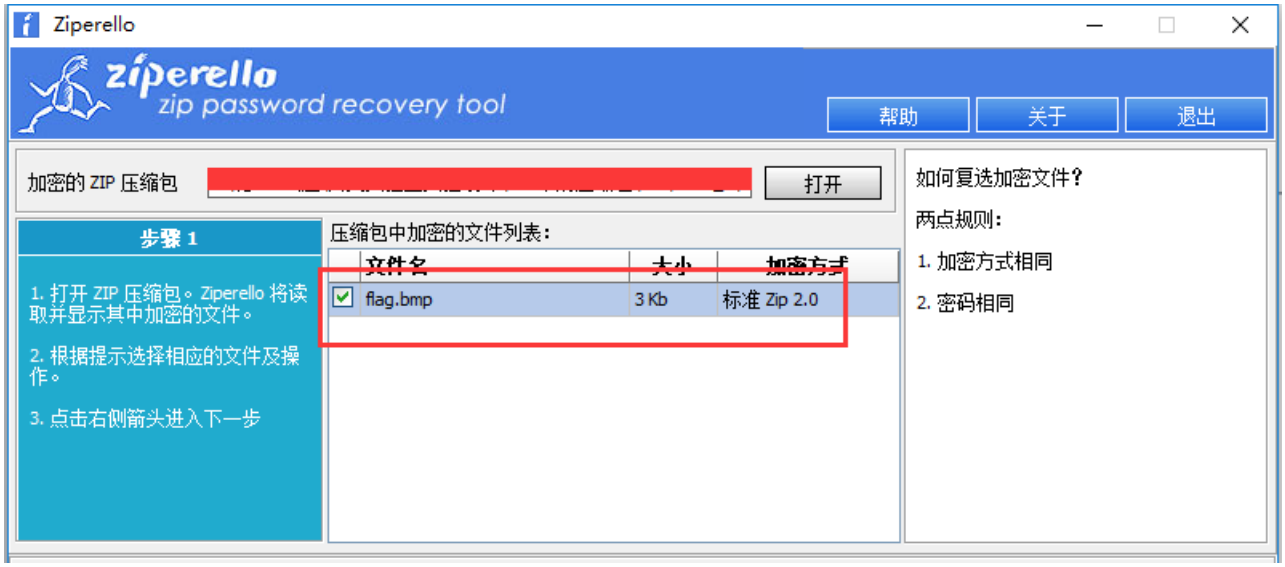




这样就可以解压出来了

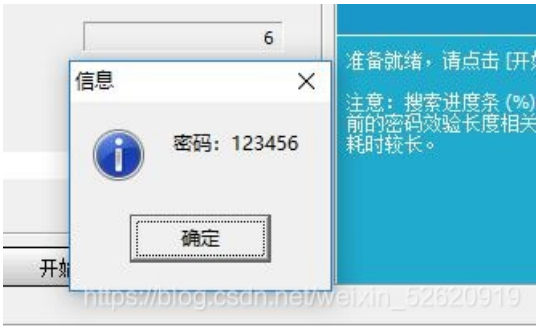
三，简单密码爆破

5. 使用工具进行爆破，一般需要爆破的压缩包密码不会复杂，大部分情况都是纯数字，这里使用ziperello对压缩包进行爆破。
6. 打开ziperello，选择要解密的压缩包。



7. 点击右下角的“NEXT”，然后选择暴力破解。

8. 选择字符集，一般比赛中使用到爆破的，都是纯数字的密码，且长度较短。

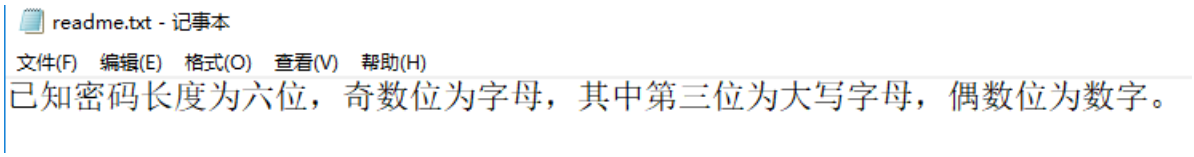


9. 再次“NEXT”，开始爆破，很快密码就爆破出来了。

四，掩码攻击。

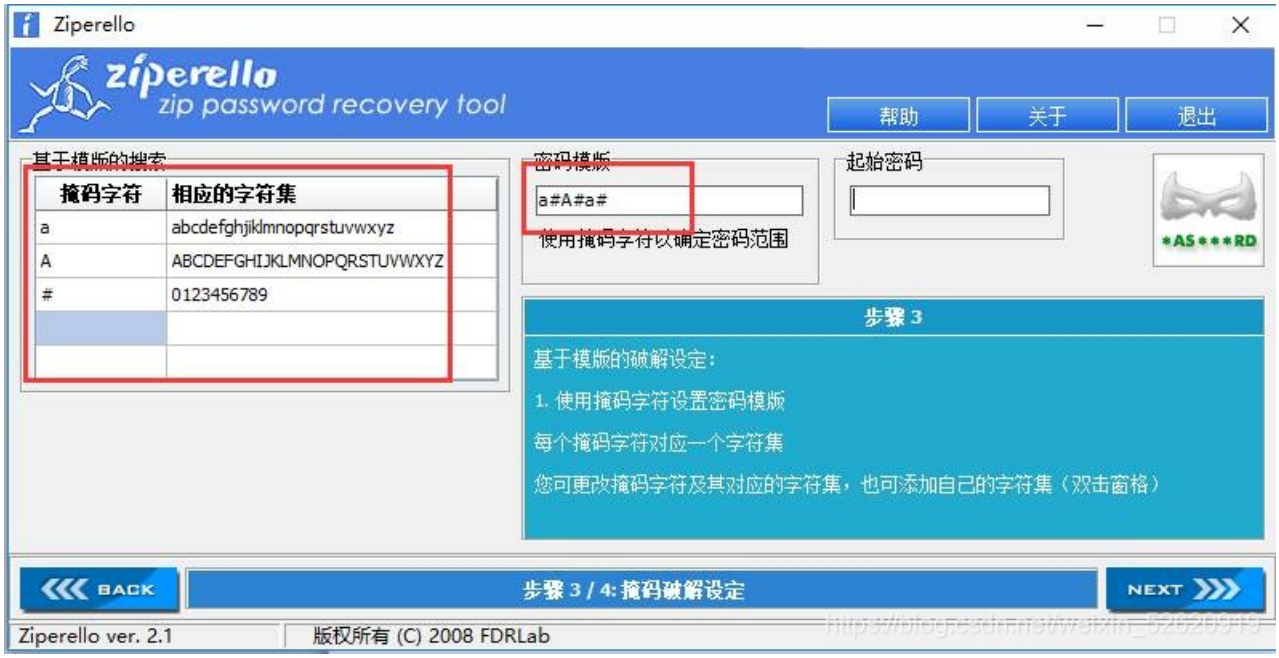
掩码攻击的意思，就是已知这个压缩包的密码格式，比如已知它的密码长度为六位，前三位为小写字母，后三位为数字，又或者已经知道这个六位长的密码第三位为“a”，其它位置有数字也有小写字母，根据这些已知的条件，来构造一个符合已知条件的字典，将所有可能的结果列出来，进行爆破。

进入“demo_4”，先查看readme.txt。得到如下信息



打开ziperello，选择要解密的文件，第二部选择“基于模板的破解”。

根据所得的信息构造符合条件的模板，如图所示，“a”，“A”，“#”分别代表一个字符集，也可以双击左边的字符集窗格构建自定义的字符集。密码模板中的“a#A#a#”就是符合已知条件的密码格式。

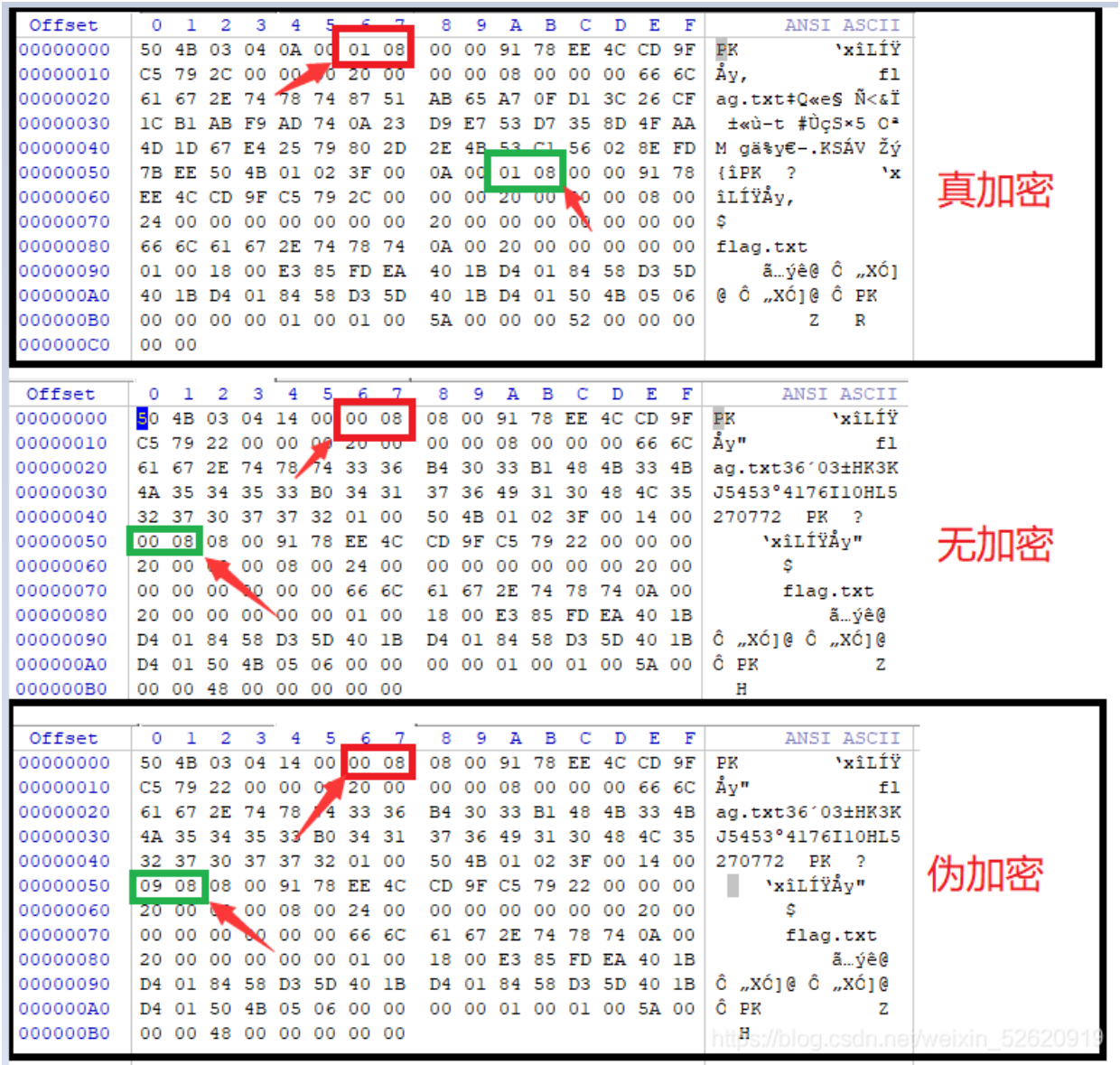


开始爆破后，很快便得到解压密码。

五，ZIP伪加密。

根据实验开始前的预备知识，我们已经知道ZIP文件格式中存在两个加密标志位，前面的加密标志位为数据区的加密标志位，后面的加密标志位为目录区的加密标志位。所谓伪加密就是修改目录区的加密标志位，使本来没有加密的ZIP文件，在解压的时候，需要用户输入密码来进行解压，但是这个密码又是不存在的。

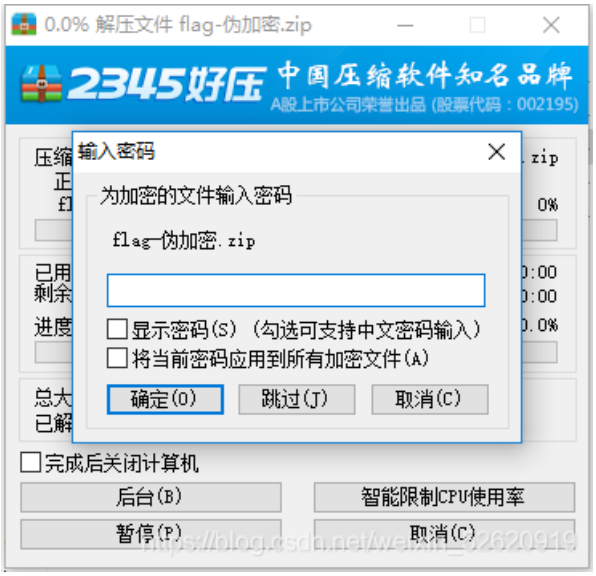
我们以同一个压缩文件的无加密、伪加密、真加密三种形式来做比较，如图，红色框为数据区加密标志位，绿色框为目录区加密标志位。从图中的比较我们可以知道，真加密的ZIP文件，两个加密标志位应该都表示加密（一般为前面一位数为奇数即可表示为加密），无加密的ZIP文件，两个标志位均表示没有加密，而伪加密则只有目录区的加密标志位表示为加密（为什么只有目录区呢，我们可以测试，只将数据区的加密标志位改为奇数，选择解压文件时，会发现他还是没有加密



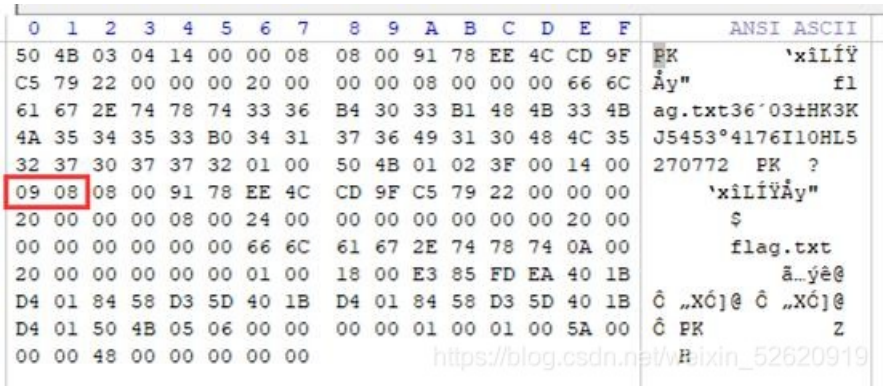
的）。

所以，在判断一个压缩文件为伪加密之后，只需将其目录区的加密标志位前面一位改为奇数即可。

进入“demo_5”文件夹，解压“flag-伪加密.zip”会发现需要我们输入密码才能解压，使用winhex打开“flag-伪加密.zip”，将图示



的0908改为0808（只要将9改为偶数即可）。



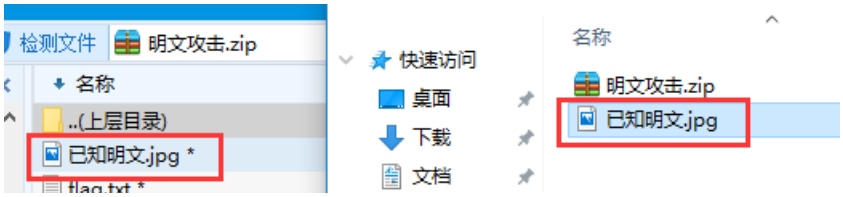
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ANSI ASCII
50	4B	03	04	14	00	00	08	08	00	91	78	EE	4C	CD	9F	PK 'xîLÍŸ
C5	79	22	00	00	00	20	00	00	00	08	00	00	00	66	6C	Åy" fl
61	67	2E	74	78	74	33	36	B4	30	33	B1	48	4B	33	4B	ag.txt36´03±HK3K
4A	35	34	35	33	B0	34	31	37	36	49	31	30	48	4C	35	J5453°4176I10HL5
32	37	30	37	37	32	01	00	50	4B	01	02	3F	00	14	00	270772 PK ?
09	08	08	00	91	78	EE	4C	CD	9F	C5	79	22	00	00	00	'xîLÍŸÅy"
20	00	00	00	08	00	24	00	00	00	00	00	00	00	20	00	$
00	00	00	00	00	00	66	6C	61	67	2E	74	78	74	0A	00	flag.txt
20	00	00	00	00	00	01	00	18	00	E3	85	FD	EA	40	1B	ã…ýê@
D4	01	84	58	D3	5D	40	1B	D4	01	84	58	D3	5D	40	1B	Ô „XÓ]@ Ô „XÓ]@
D4	01	50	4B	05	06	00	00	00	00	01	00	01	00	5A	00	Ô PK Z
00	00	48	00	00	00	00	00									H

此时便可以正常解压

六，明文攻击。

明文攻击是一种较为高效的攻击手段，大致原理是当你不知道一个zip的密码，但是你有zip中的一个已知文件（文件大小要大于12Byte）时，因为同一个zip压缩包里的所有文件都是使用同一个加密密钥来加密的，所以可以用已知文件来找加密密钥，利用密钥来解锁其他加密文件。在压缩文件时输入的密码，首先被转换成3个32bit的key，所以可能的key的组合是2^96，如果用暴力穷举的方式是不太可能的，除非密码比较短或者有个厉害的字典。压缩软件用这3个key加密所有包中的文件，所有文件的key是一样的，如果我们能够找到这3个key，就能解开所有的文件。如果我们找到加密压缩包中的任意一个文件，这个文件和压缩包里的文件是一样的，我们把这个文件用同样的压缩软件同样的压缩方式进行无密码的压缩包（这里可以通过比较压缩后文件的CRC32值来判断，如果一样，则使用的是同样的压缩方式和软件），得到的文件就是我们的Known plaintext（已知明文）。用这个无密码的压缩包和有密码的压缩包进行比较，分析两个包中相同的那个文件，抽取出两个文件的不同点，就是那3个key了，如此就能得到key。两个相同文件在压缩包中的字节数应该相差12个byte，就是那3个key了。虽然我们还是无法通过这个key还原出密码，但是我们已经可以用这个key解开所有的文件，所以已经满足我的要求了，得到其中加密的其他文件。更详细的原理请读者自行谷歌。

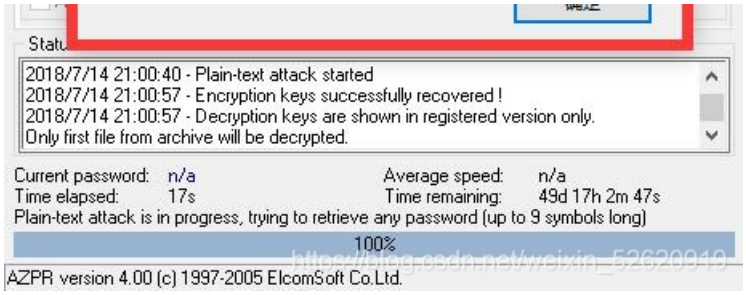
进入“demo_6”文件夹，文件夹中存在一个加密的压缩包“明文攻击.zip”和一个“已知明文.jpg”，双击点开压缩包，即可看见

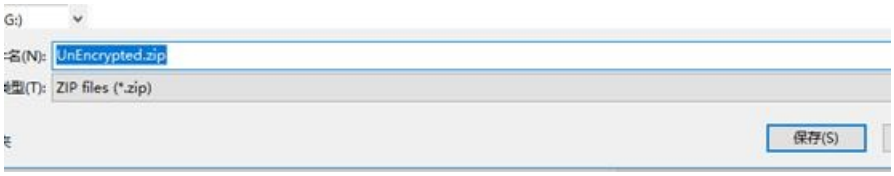


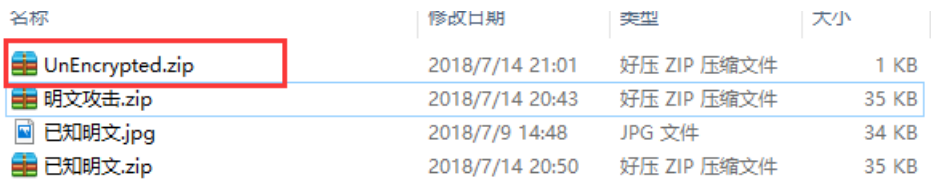
压缩包中也存在一个“已知明文.zip”。

我们对“已知明文.jpg”进行无加密的压缩，然后对比两个压缩文件中“已知明文.jpg”的CRC32值，如果一致，就可以进行明文攻击，如果不一致，则换一种压缩方式继续比较。

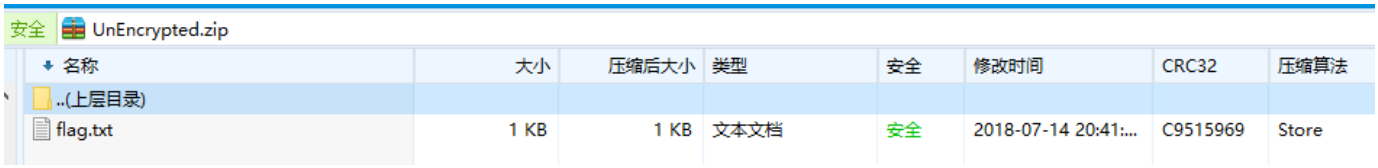
使用工具进行明文攻击，这里使用AZPR进行明文工具。







攻击成功，得到其他的加密文件。



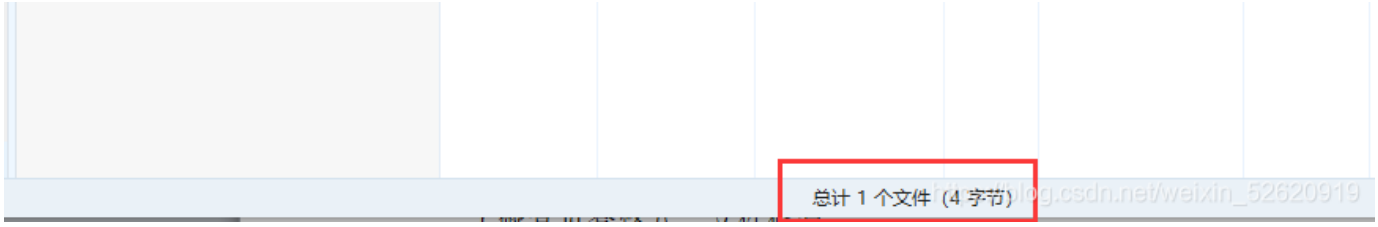
七，CRC32碰撞。

1. CRC32:CRC本身是“冗余校验码”的意思，CRC32则表示会产生一个32bit（8位十六进制数）的校验值。在产生CRC32时，源数据块的每一位都参与了运算，因此即使数据块中只有一位发生改变也会得到不同的CRC32值，利用这个原理我们可以直接爆破出加密文件的内容，但是CRC32值也存在被碰撞的可能，也就是会出现内容不一样但是CRC32值一样的情况，所以利用CRC32碰撞的方法得知压缩文件的内容，一般是在被压缩的文件很小的情况下，在CTF中一般为4个字节。

CRC简介

```
在数据传输过程中，无论传输系统的设计再怎么完美，差错总会存在
这种差错可能会导致在链路上传输的一个或者多个帧被破坏(出现比特差错，0变为1，或者1变为0)，
从而接受方接收到错误的数据。
为尽量提高接受方收到数据的正确率，在接收方接收数据之前需要对数据进行差错检测，
当且仅当检测的结果为正确时接收方才真正收下数据。
检测的方式有多种，常见的有奇偶校验、因特网校验和循环冗余校验等。
循环冗余校验是一种用于校验通信链路上数字传输准确性的计算方法（通过某种数学运算来建立数据位和校验位的约定关系的 [1]  ）。
发送方计算机使用某公式计算出被传送数据所含信息的一个值，并将此值 附在被传送数据后，
接收方计算机则对同一数据进行 相同的计算，应该得到相同的结果。
如果这两个 CRC结果不一致，则说明发送中出现了差错，接收方计算机可要求发送方计算机重新发送该数据。

在计算机网络通信中运用CRC校验时相对于其他校验方法就有一定的优势。
CRC可以高比例的纠正信息传输过程中的错误，可以在极短的时间内完成数据校验码的计算，
并迅速完成纠错过程，通过数据包自动重发的方式使得计算机的通信速度大幅提高，对通信效率和安全提供了保障。
由于 CRC 算法检验的检错能力极强，且检测成本较低，因此在对于编码器和电路的检测中使用较为广泛。
从检错的正确率与速度、成本等方面，都比奇偶校验等校验方式具有优势。
因而，CRC 成为计算机信息通信领域最为普遍的校验方式。
```

2. 进入“demo_7”文件夹，文件夹中存在四个加密的压缩包，并且有一定的命名顺序，readme.txt文件中表明被压缩的文件中的内容为base64编码的字符串。可以看见每个压缩文件内的文件大小均为四个字节，即文件内容为四个字节长度的字符串。



3. 所以使用CRC碰撞的方法循环碰撞出所有压缩文件的内容并按顺序拼接即为最终的答案，运行目录下的CRC32.py脚本，最后会生成一个flag.txt文件，文件的内容即为最终的答案。

```
[redacted]demo_7>python CRC32.py
Cracking........1.zip
Cracking Successfuly
Cracking........2.zip
Cracking Successfuly
Cracking........3.zip
Cracking Successfuly
Cracking........4.zip
Cracking Successfuly
```

4. 关于对脚本的解释，都在代码的注释中。

CRC32.py

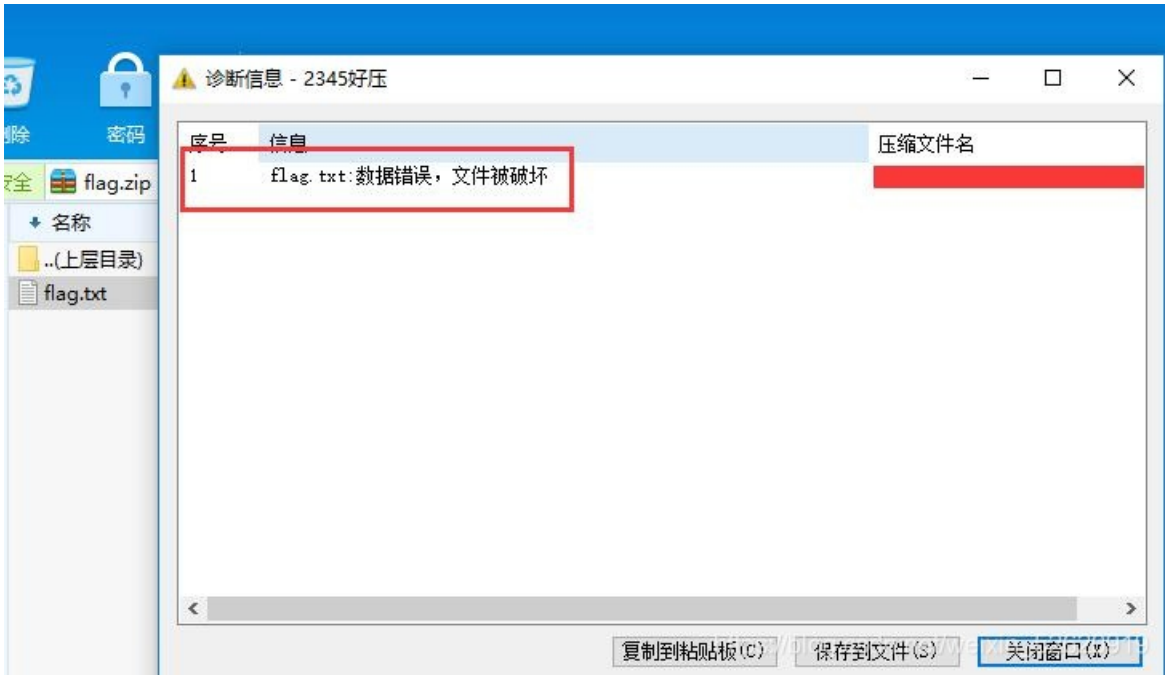
```python
# -*- coding:utf-8 -*-
import zipfile
import string
import binascii
#构造字符集，因为题目提及是base64字符串，所以使用大小写字母+数字+‘+/=’
dic = string.ascii_letters + string.digits + '+/='
#具体碰撞CRC的函数
def CrackCRC32(crc_str):
    #从字符集中依次取出字符组成四位字符串进行碰撞
    for i in dic:
        for j in dic:
            for p in dic:
                for q in dic:
                    s = i + j + p + q
                    '''
                    在 Python 2.x 的版本中，binascii.crc32 所计算出来的
                    CRC 值域为[-2^31,2^31-1] 之间的有符号整数，为了要与
                    一般CRC 结果作比对，需要将其转为无符号整数，所以加上
                    & 0xffffffff来进行转换。如果是 Python 3.x 的版本，其
                    计算结果为 [0, 2^32-1] 间的无符号整数，因此不需额外加
                    上& 0xffffffff 。
                    '''
                    if crc_str == (binascii.crc32(s) & 0xffffffff):
                        print "Cracking Successfuly"
                        #碰撞成功后将碰撞出的内容写入flag.txt文件中
                        flag.write(s)
                        return
#选择ZIP文件的函数
def CrackZIP():
    for i in range(1,5):
        file = str(i) + ".zip"
        #获取压缩包中文件的CRC32值
        f = zipfile.ZipFile(file,'r')
        GetCRC32 = f.getinfo(str(i) + ".txt")
        crc_str = GetCRC32.CRC
        print "Cracking........" + file
        CrackCRC32(crc_str)

flag = open('flag.txt','wb')
```

```
flag = open("flag.txt","wb")
CrackZIP()
flag.close
```

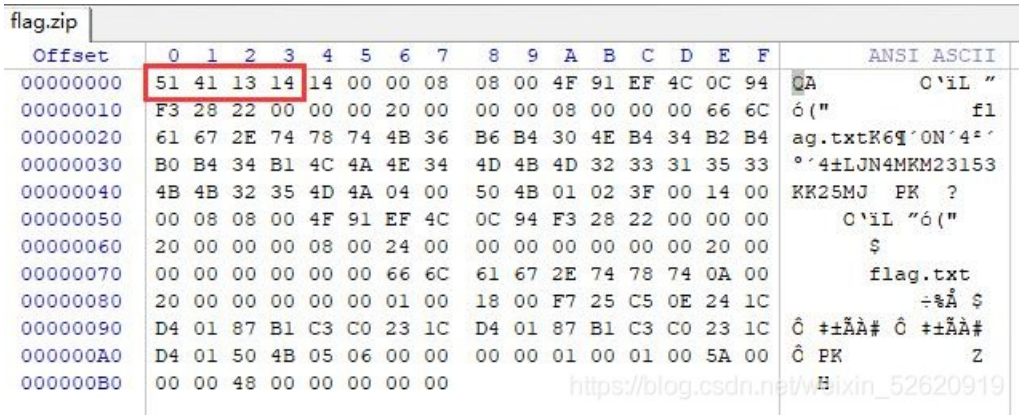

八，文件修复

这个常见的套路也是跟文件的格式有关，一般都是修改ZIP文件的开始标识，使得在解压文件时产生错误，无法解压。

进入“demo_8”文件夹下，尝试解压 “flag.zip”，会报错。



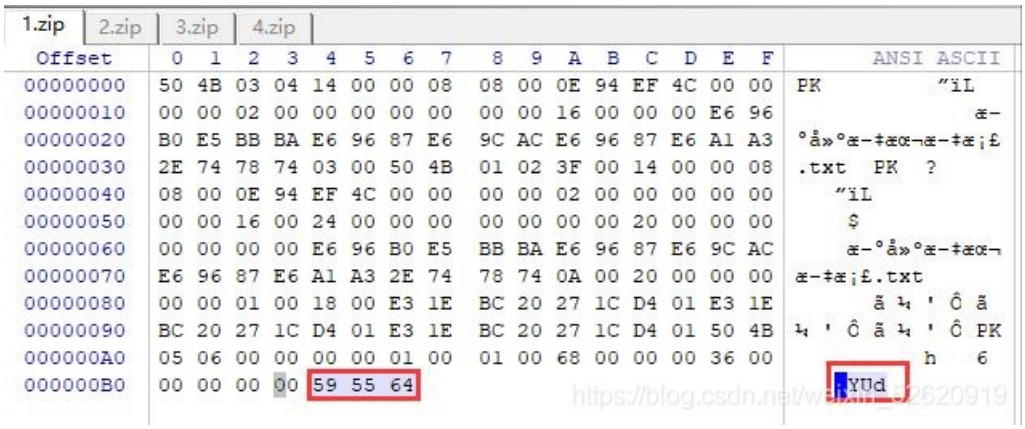
使用winhex打开这个压缩文件进行查看，会发现文件头有些异常，从预备知识中我们知道ZIP文件的头部标识为“504B0304”，且为固定值，所以我们将其修改回来，即可进行正常解压。



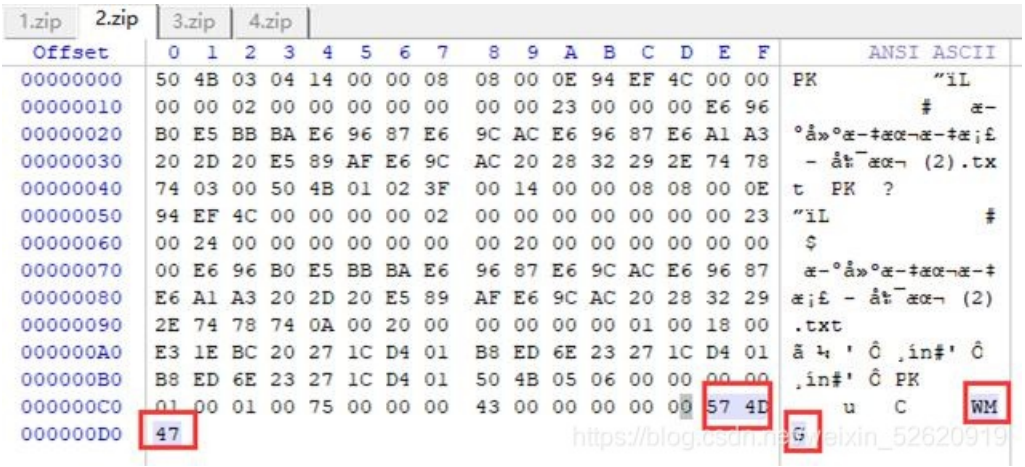
九，冗余信息拼接。

从预备知识中我们知道，ZIP压缩文件目录结束标识位为“504B0506”，且通常带有18字节（在预备知识中我们将每个偏移量视作一位，也是一个字节）的冗余数据，总共长度一般为22个字节，所以这个套路就是将隐藏信息分为多片隐藏在多个压缩包的结尾。

进入“demo_9”文件夹下，有四个压缩文件，解压后文件内容为空。使用winhex打开这些压缩包，发现每个压缩包的最后都



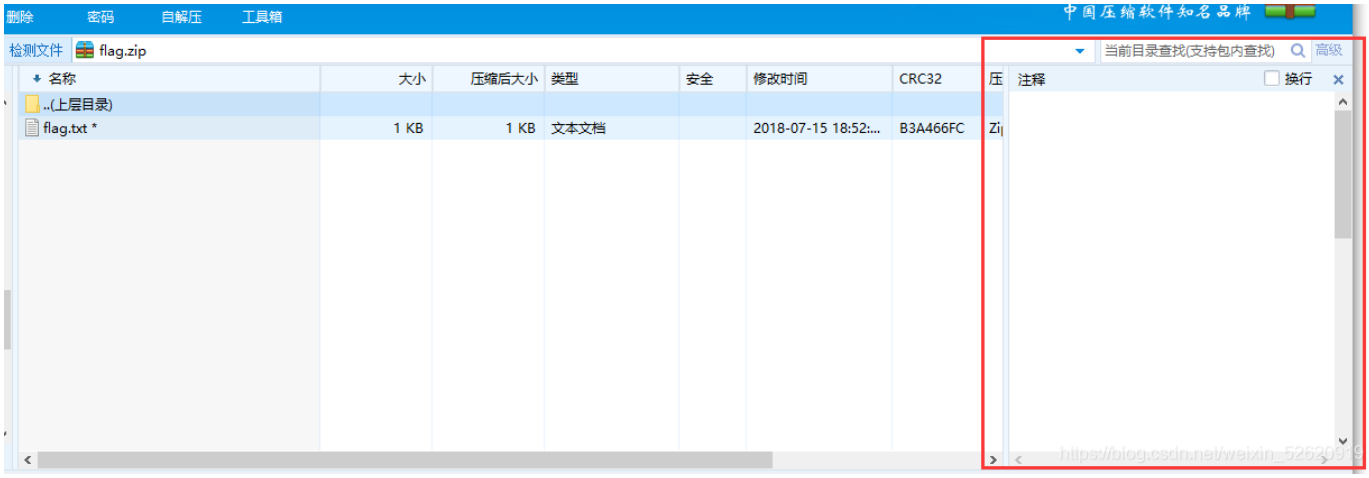
有三个字节的冗余数据。



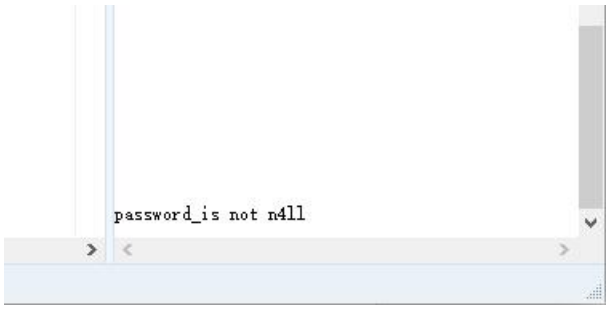
将这些冗余数据复制取出（选中多出来的数据，右键选择Edit -> Copy Block -> Normally），拼接后得到base64字符串，解码后得到答案。

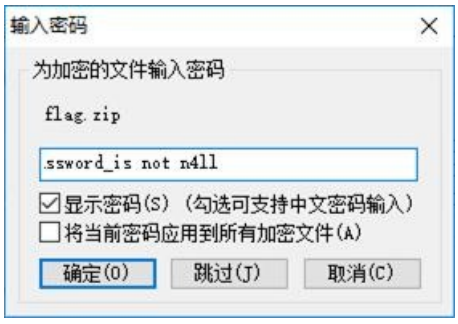
十，注释隐藏密码

1. 进入“demo_10”文件夹下，双击打开flag.zip，会发现旁边多出了一个注释框，而这个注释框只有在这个压缩文件存在注释的情况下才会显示。



2. 但是并没有看见注释框中有什么内容，尝试下拉右边的下滑条，发现先一个字符串。





3. 猜测为压缩文件的解压密码，复制输入到密码框中，成功解压。