【学习总结】ctf隐写初阶解题思路与方法

原创

Kelly_Young 于 2018-07-29 10:44:26 发布 6944 收藏 34
文章标签： ctf y

作为ctf中相对较为简单的题目，隐写题更适合初学者上手和提高初学者的兴趣。本人也对隐写术很感兴趣，于是尝试着对自己目前所学的隐写解题思路和工具的使用做一个总结。由于水平有限，总结可能会有错误的地方，希望大佬们不吝赐教，谢谢！

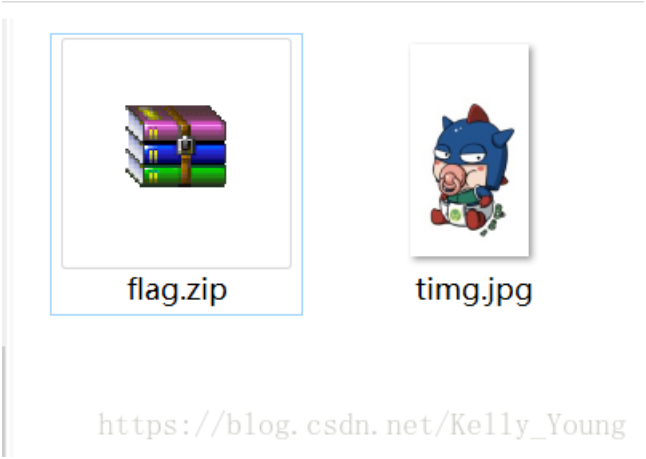
一.JPEG图片

这类图片在图片隐写中经常出现，其中，jpg图片是隐藏文件、信息的常见载体。关于jpg格式的更多信息见：https://blog.csdn.net/yangysng07/article/details/9025443

1.文件的合并与分离

在初阶的隐写题目中，常见的有在jpg图片中藏有压缩文件，需要我们把图片与压缩文件分离，从压缩文件里得到flag。现在，我们可以自己在电脑上操作一下，看看此类照片是怎么做出来的，以便于我们解题。

首先，找一张jpg格式的图片，并制作一个zip文件。



然后我们在当前文件夹下面打开cmd，执行copy /b timg.jpg+flag.zip outfile.jpg命令。就可以得到一张图种，也就是藏有文件的图片了。

outfile.jpg这种图经常就出现在一些简单的ctf隐写题里面了。

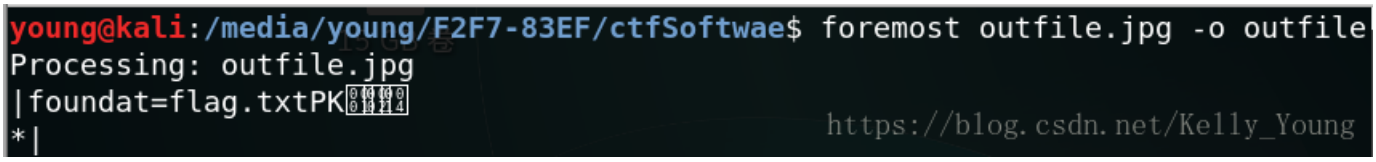
那么，我们该如何分离outfile.jpg这张图片呢？ 我们可以利用binwalk工具来识别文件的结构，发现图片里的隐藏文件。binwalk是kali里自带的一个命令行工具。在终端中输入binwalk <filename><格式>

我们可以发现在outfile.jpg中除了有一张图片外，还存在一个zip文件。

既然发现了图片里的隐藏图片，现在我们用两种方法将之分离。

1.使用foremost工具使之分离

foremost能够将多个合并文件进行分离。在kali中，我们在刚才那张图片的文件夹下，在终端执行foremost outfile.jpg -o outfile命令，将outfile.jpg与flag.zip分离到outfile这个文件夹中。



我们进入这个文件夹可以看到存在foremost自动生成的audit.txt文件、jpg和zip文件

我们解压zip文件就可以得到文件中的flag了。

2.直接用解压工具解压

如果我们在binwalk上面发现图片里存在zip等压缩文件，我们可以直接修改图片后缀名为压缩格式，然后用解压工具解压的到文件即可。

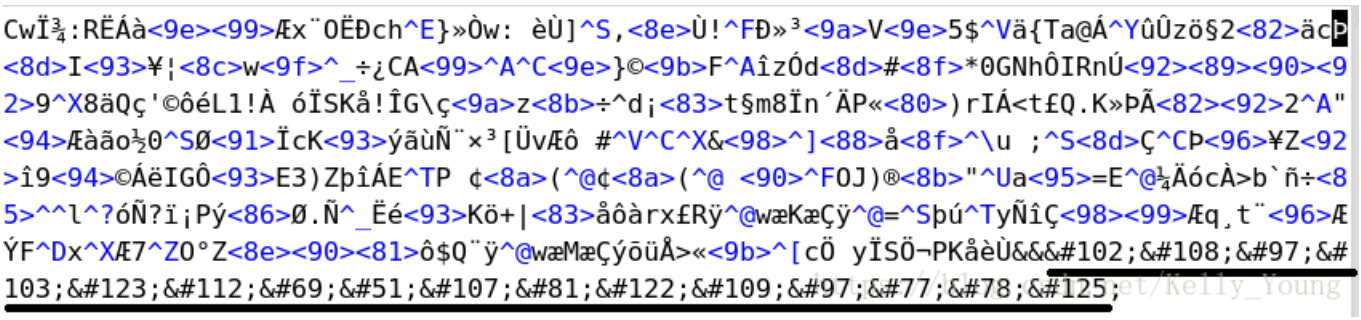
此外，在jpg图片还可以与jpg图片合并，较为简单的题目会把flag以图片的形式与另外一张图片合并，此时我们直接使用foremost分离出图片得到flag即可。

2.文件中插入信息

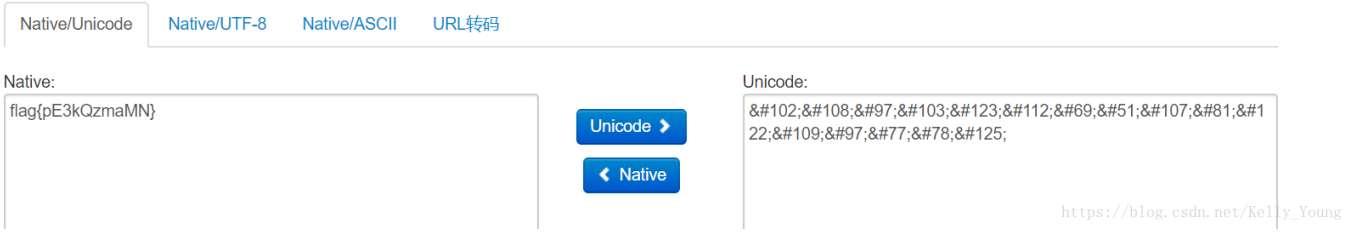
对于一些初阶的隐写题，flag信息也许会插入在文件的头部或者尾部，稍难一点的也许会插入到文件中间。而且flag形式一般会进行编码（如base64等），我们需要找到flag的位置并解码，最终得到flag。

我以一道ctf题为例来总结：

这张jpg图片我们使用binwalk检查，发现没有任何异常。这时候怎么办呢？我们尝试用文本查看器打开这张图片查看该图片的源码，先在文件首尾查看有没有flag或者不一样的编码。我在文件尾部发现了与图片源码不一样的编码：



熟悉HTML等语言的同学们应该一眼就能发现这是HTML的实体编码，那么，我们直接复制着一串编码，直接在网上对应的在线解码工具中解码即可。

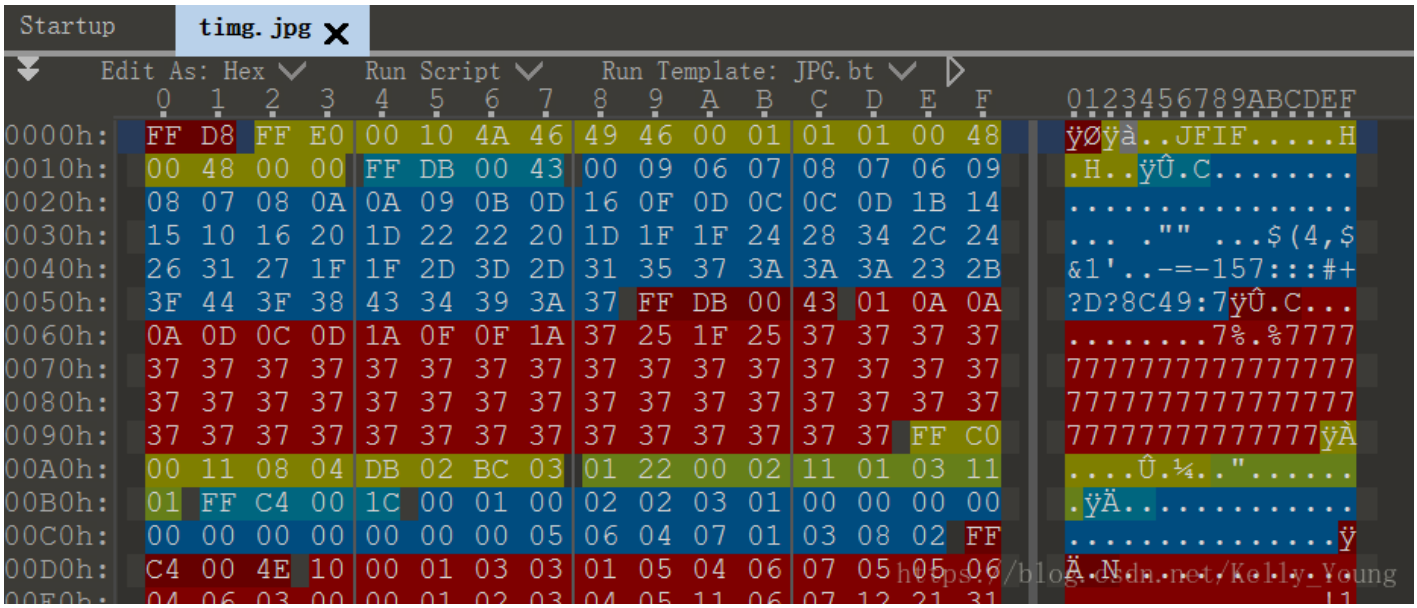


3.JPEG格式隐写

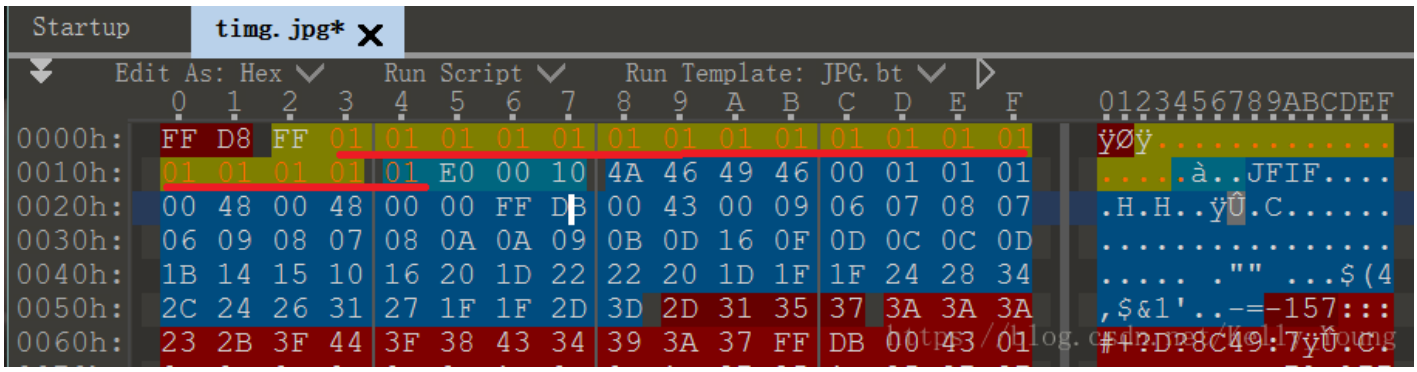
如果你仔细阅读了JPEG格式的文档，我们再来看JPEG格式的图片会怎么隐藏flag。

1.JPEG图片格式分为：标记码和压缩数据；如果在标记码之间隐写数据，不会影响图片的正常打开。

我们照一张JPEG格式的图片，用010Editor打开，在标记码之间加入一串数据，保存后发现图像依旧可以正常打开。

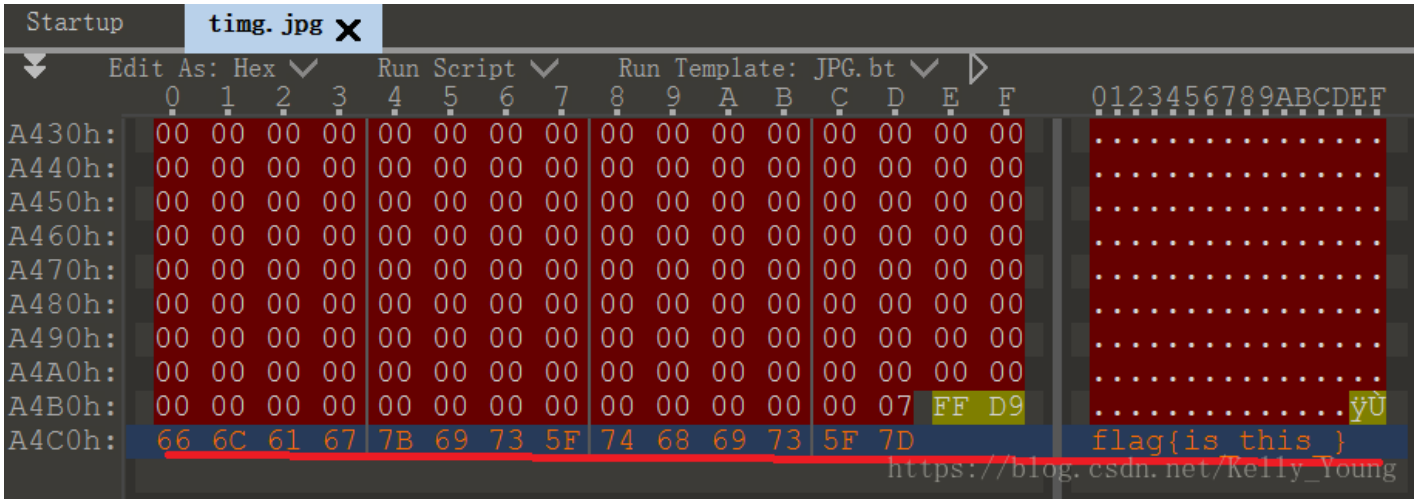


未修改的图片



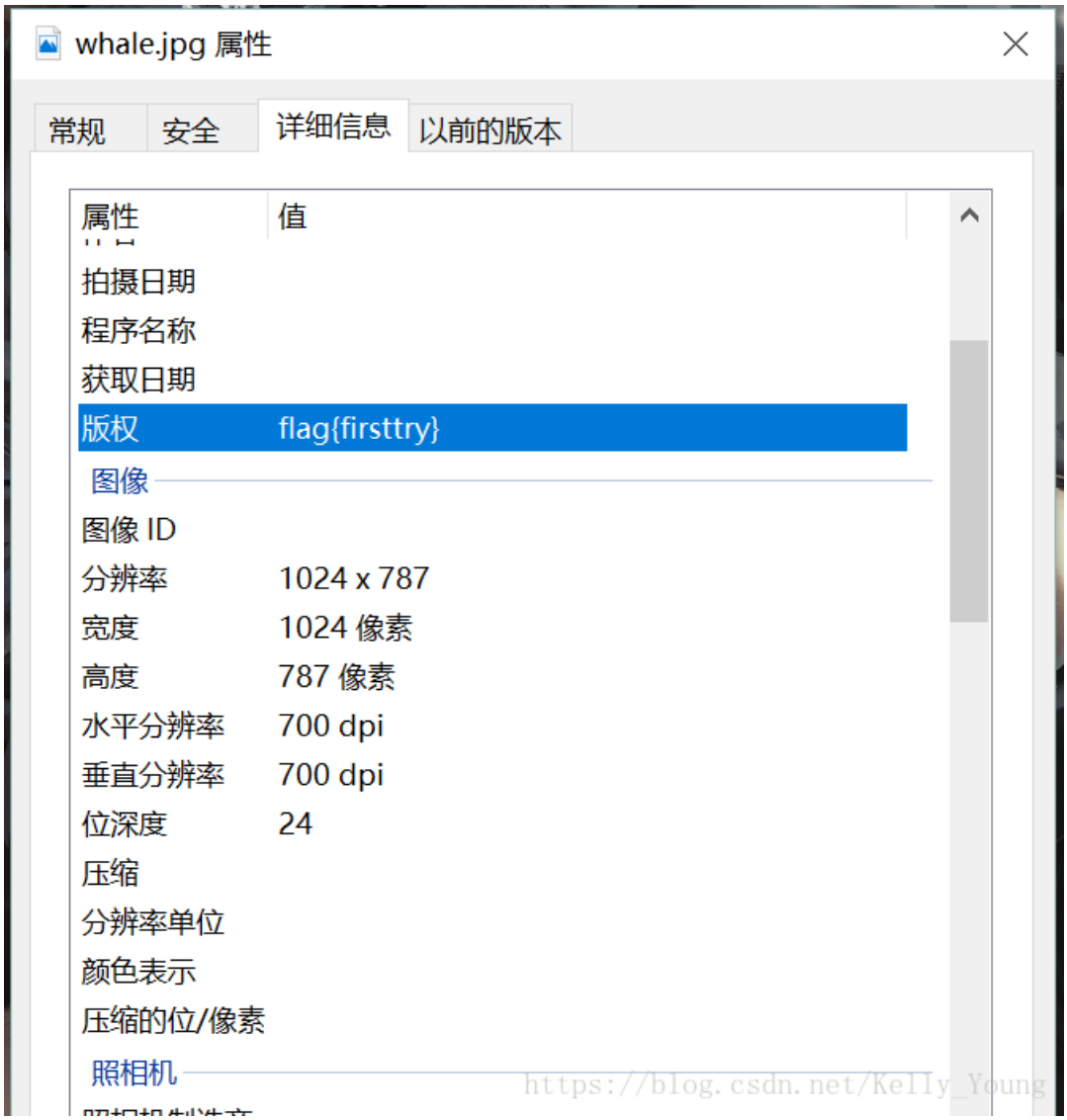
在标记码之间随意加上一些数据

2.JPEG文件以0xFF 0XD9结束；如果在文件尾发现还有信息，若不能直接识别，可以用010Editor或者winhex把信息复制出来，保存为新的文件，继续查找flag。



在文件尾加入信息，不影响图片打开

3. JPEG文件还有一个特殊的地方：具备exif文件描述信息。我们用Windows自带的属性查看JPEG图片，可以发现JPEG图片所附带的信息比其他类型的图片多很多，这些附带信息就储存在exif头中。因此我们查找JPEG图片的flag还可以用属性查看。



可以直接看到flag

但是，并不是所有信息都能够从Windows自带属性查看。这里我们使用exiftool工具，这款工具能够查看完整的exif头信息。当图片是JPEG格式时，可以用exiftool查看JPEG图片是否藏有flag信息。

```
young@kali:~/图片$ exiftool Question.jpg
ExifTool Version Number         : 11.03
File Name                       : Question.jpg
Directory                       : .
File Size                       : 2.7 MB
File Modification Date/Time     : 2018:07:17 10:34:49-04:00
File Access Date/Time           : 2018:07:28 03:31:38-04:00
File Inode Change Date/Time     : 2018:07:17 10:34:49-04:00
File Permissions                : rw-r--r--
File Type                       : JPEG
File Type Extension             : jpg
MIME Type                       : image/jpeg
JFIF Version                    : 1.01
Resolution Unit                 : None
X Resolution                    : 300
Y Resolution                    : 300
Comment                         : ^[[A^[[D^[[C^[[A^[[C^[[C^[[B^[[D^[[A^
[[A^[[A^[[A^[[D^[[D^[[D^[[C^[[B^[[A^[[B^[[B^[[A^[[C^[[C^[[D^[[C^[[B^[[C
^[[D^[[A^[[B^[[A^[[D^[[B^[[C^[[C^[[A^[[B^[[C^[[A^[[D^[[B^[[C^[[C^[[C^[[
```

Comment异常

小结

以上就是我对JPEG格式的隐写解题思路做的总结，当然JPEG图片的隐写远不止这些，flag经常会与密码学、RGB值以及其它脑洞思路相结合，需要多拓展才行。

二.PNG图片

在做PNG的隐写题之前，我们需要了解PNG图片格式的相关信息：http://www.cnblogs.com/fengyv/archive/2006/04/30/2423964.html。

其中比较重要的是：

PNG图片通过zlib压缩编码后分为IDAT块储存，每个IDAT块能够储存65524大小的数据。

IHDR标识能够控制图片显示的长和宽。

下面，我就这两个点分别以两个例题讲解：

1.IDAT块异常

我们看这样一张图：

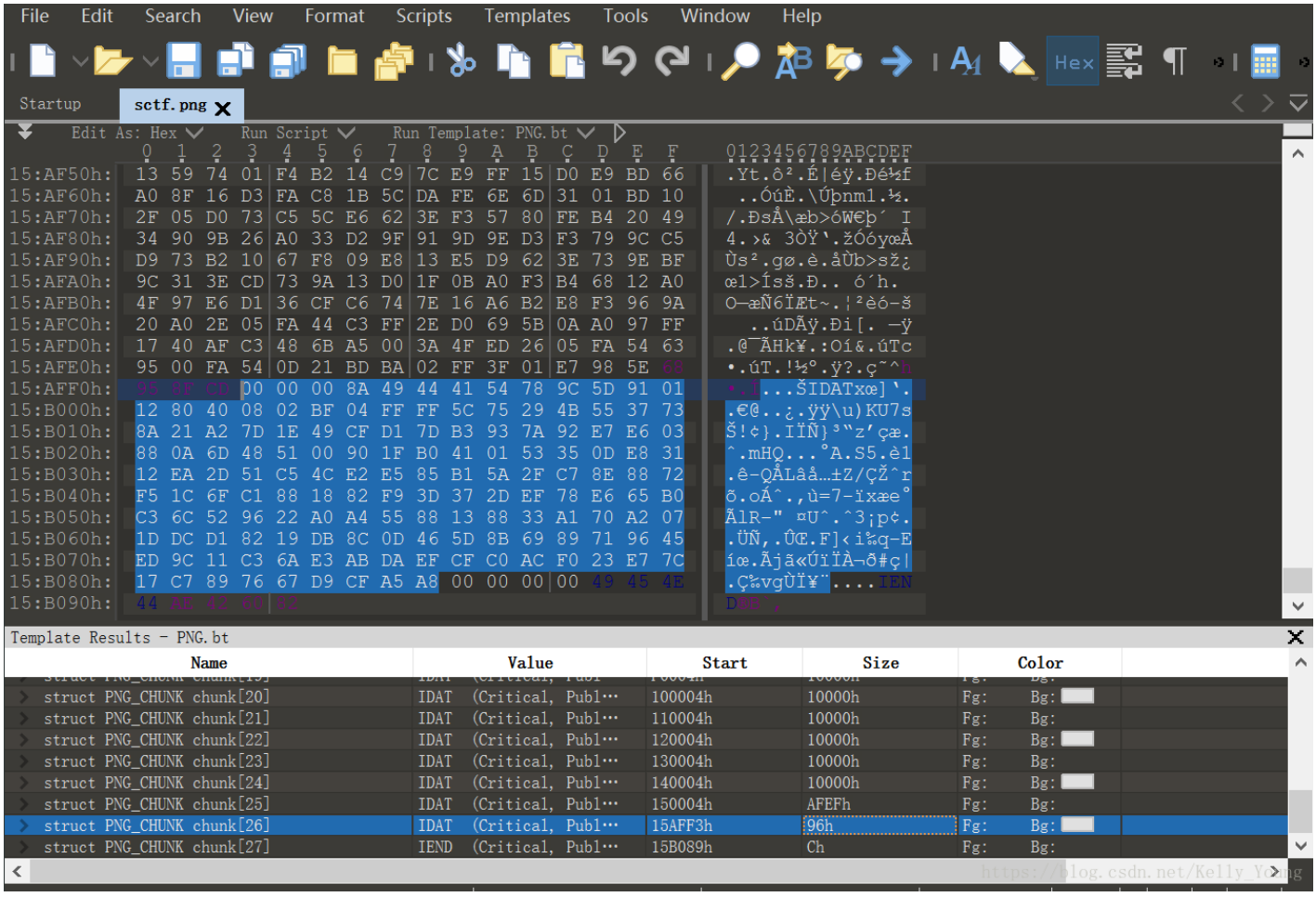
我们知道，PNG图片每个IDAT块能储存65524大小的数据，而且必须一个块填满才会把数据写入下一个数据块。那么，我们拿到一张PNG图片，首先使用工具pngcheck来检查该图片。

在cmd下输入如下命令：pngcheck.exe -v <图片名>.png (如果是在Linux系统下，则为：pngcheck -v <图片名>.png)

```
E:\study&life\ctf\CtfSoftwave\CTF工具合集\隐写\图像隐写\pngcheck-2.3.0-win32>pngcheck.exe -v sctf.png
File: sctf.png (1421461 bytes)
  chunk IHDR at offset 0x0000c, length 13
    1000 x 562 image, 32-bit RGB+alpha, non-interlaced
  chunk sRGB at offset 0x00025, length 1
    rendering intent = perceptual
  chunk gAMA at offset 0x00032, length 4: 0.45455
  chunk pHYs at offset 0x00042, length 9: 3780x3780 pixels/meter (96 dpi)
  chunk IDAT at offset 0x00057, length 65445
    zlib: deflated, 32K window, fast compression
  chunk IDAT at offset 0x10008, length 65524
  chunk IDAT at offset 0x20008, length 65524
  chunk IDAT at offset 0x30008, length 65524
  chunk IDAT at offset 0x40008, length 65524
  chunk IDAT at offset 0x50008, length 65524
  chunk IDAT at offset 0x60008, length 65524
  chunk IDAT at offset 0x70008, length 65524
  chunk IDAT at offset 0x80008, length 65524
  chunk IDAT at offset 0x90008, length 65524
  chunk IDAT at offset 0xa0008, length 65524
  chunk IDAT at offset 0xb0008, length 65524
  chunk IDAT at offset 0xc0008, length 65524
  chunk IDAT at offset 0xd0008, length 65524
  chunk IDAT at offset 0xe0008, length 65524
  chunk IDAT at offset 0xf0008, length 65524
  chunk IDAT at offset 0x100008, length 65524
  chunk IDAT at offset 0x110008, length 65524
  chunk IDAT at offset 0x120008, length 65524
  chunk IDAT at offset 0x130008, length 65524
  chunk IDAT at offset 0x140008, length 65524
  chunk IDAT at offset 0x150008, length 45027
  chunk IDAT at offset 0x15aff7, length 138
  chunk IEND at offset 0x15b08d, length 0
No errors detected in sctf.png (28 chunks, 36.8% compression).
```


从这里我们就可以看到异常了，倒数第二个IDAT块在数据没有填满的情况下，还多生成了一个IDAT数据块，这说明最后一个IDAT块存在异常。

那么我们使用010Editor或者Winhex查看图片：



我们找到存在问题的IDAT块，49 44 41 54是IDAT块头部信息，D9 CF A5 A8是CRC校检位，我们又知道中间的信息是zlib压缩格式，于是我们把中间的数据复制出来，尝试用一个python脚本来解压缩：

```
File  Edit  Format  Run  Options  Windows  Help
import zlib
import binascii
IDAT = "789C5D91011280400802BF04FFFF5C75294B5537738A21A27D1E49CFD17DB3937A92E7E6
#print IDAT
result = binascii.hexlify(zlib.decompress(IDAT))
#print result
bin = result.decode('hex')
print bin
print '\r\n'
print len(bin)
```



```
Python 2.7.5 (default, May 15 2013, 22:43:36) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
11111110001000011011111111000001011100101101000001101110101000000000101110110111
01001000000001011101101110101110111010010111011000001010101101101000001111111110101
10101010111111100000000101110111000000000110100110000010100111011011110101010010000
00111000000000001010000000010010011010001001110011111011100111100001110111111000110
00101000110011100001010100011010001111010110000010100010110000011011101100100001
11001110010000101111111010000000011010100100011110111111101110000110101101110000
01000011001100011110101110100011010011111000010111010110001110100111001011101001
00111011011000110000010110001101000110001111111011010110111011011
```



```
625
>>>
```

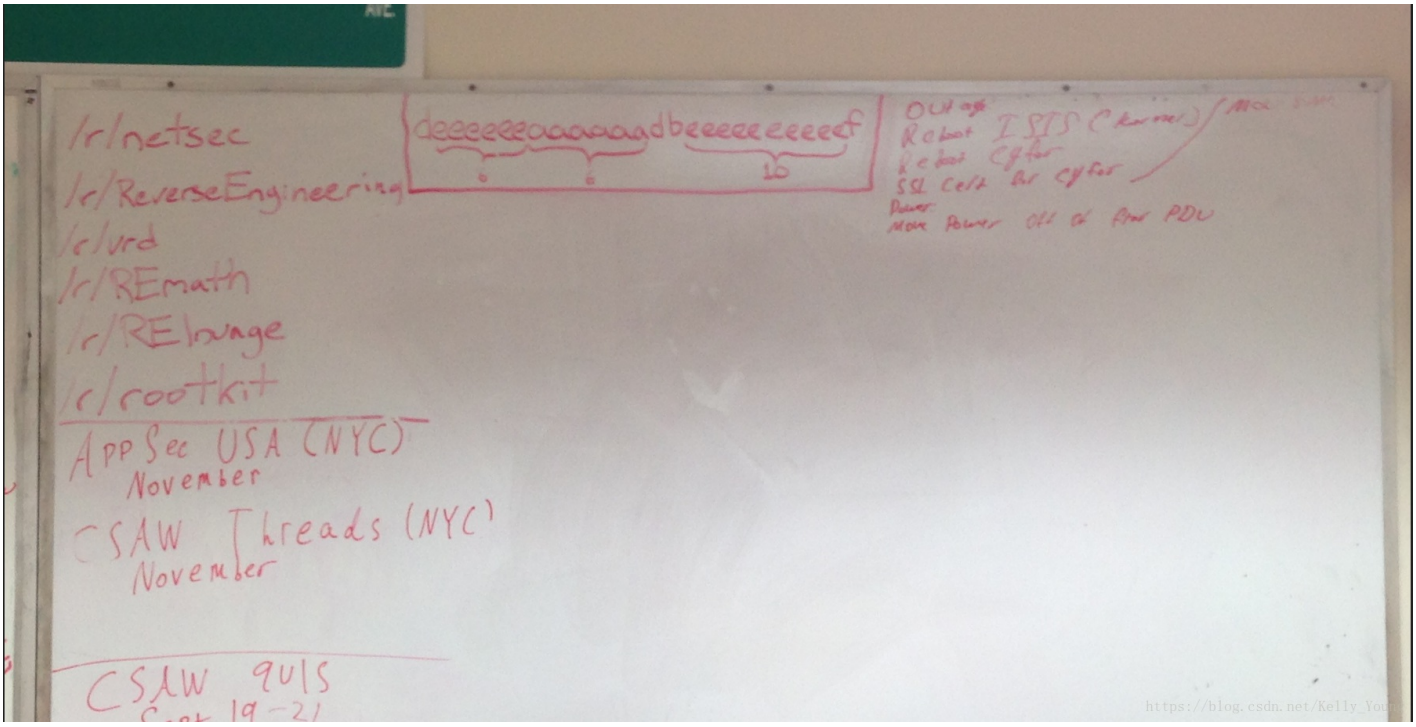
我们可以看到解出来的是二进制串，有625位，因为625不是7或者8的倍数，因此我们排除转为ASCLL码的情况。我们又发现625=25x25，是一个正方形的形状，那么，我们很自然的可以想到这是一幅二维码，我们用python的image库画图：

```
#!/usr/bin/env python
import PIL.Image
MAX = 25
pic = PIL.Image.new("RGB",(MAX, MAX))
str = "11111110001000011011111111000001011100101101000001101110101000000000101111
i=0
for y in range (0,MAX):
    for x in range (0,MAX):
        if(str[i] == '1'):
            pic.putpixel([x,y],(0, 0, 0))
        else:
            pic.putpixel([x,y],(255,255,255))
        i = i+1

pic.show()
pic.save("flag.png")
```


运行后得到一个二维码，直接在网上的二维码扫描上扫描，就可以得到flag了。

2.修改IHDR显示的高度

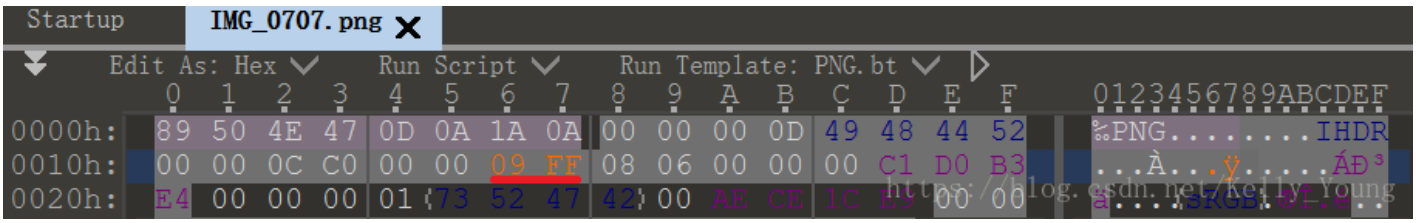
IHDR标记头后紧跟着长度和宽度限制位（各4位，宽度限制位在前）。ctf中有些PNG图片就是修改了长度的标记位，从而达到隐藏flag的目的。我们来看这样的一张图：



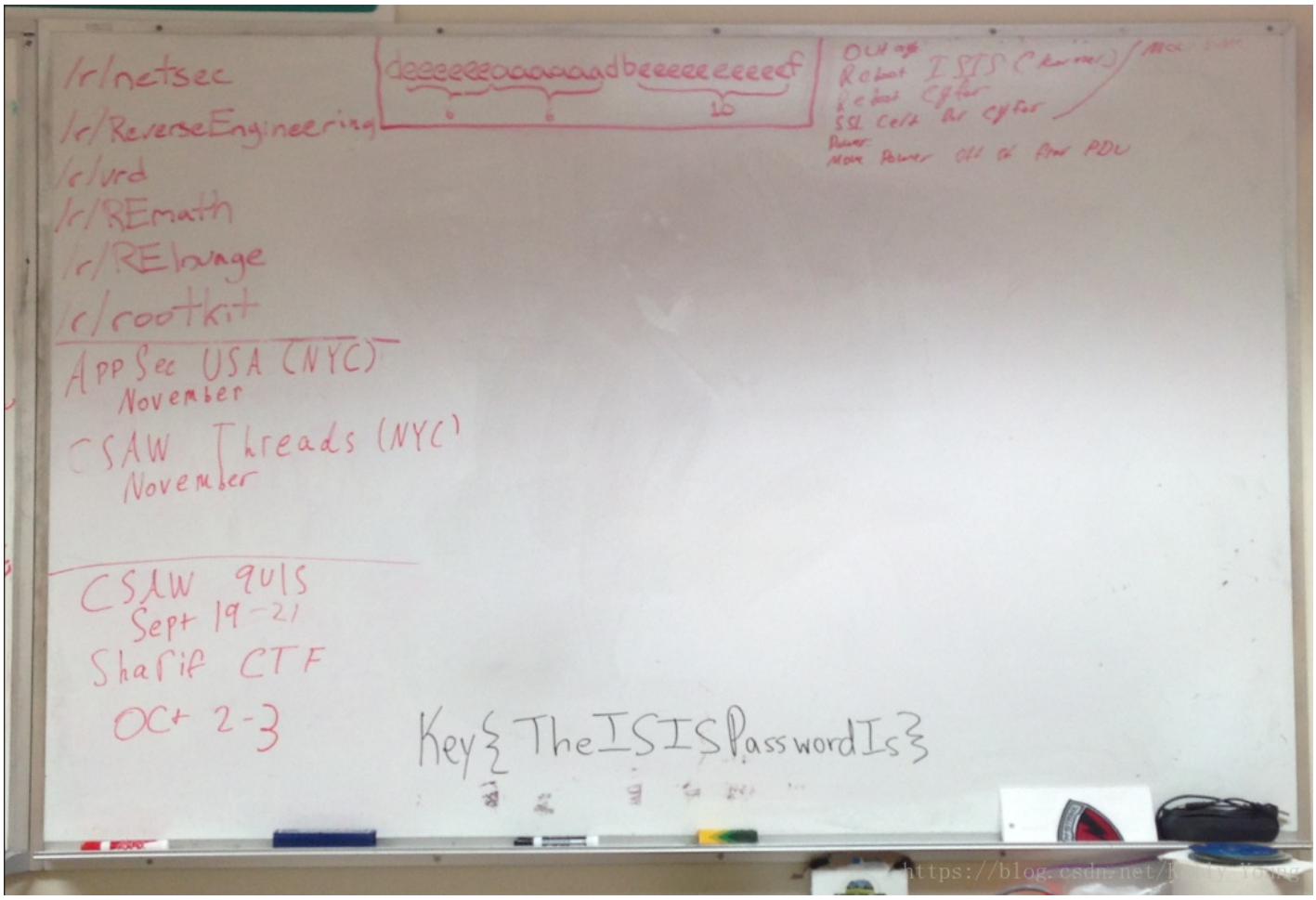
从这张图上我们不能发现任何有用信息，于是我们继续使用pngcheck来检查一下图片是否存在问题：

```
E:\study&life\ctf\CtfSoftwave\CTF工具合集\隐写\图像隐写\pngcheck-2.3.0-win32>cmd.exe
Microsoft Windows [版本 10.0.17134.165]
(c) 2018 Microsoft Corporation。保留所有权利。

E:\study&life\ctf\CtfSoftwave\CTF工具合集\隐写\图像隐写\pngcheck-2.3.0-win32>pngcheck.exe -v IMG_0707.png
File: IMG_0707.png (6214620 bytes)
  chunk IHDR at offset 0x0000c, length 13
    3264 x 1681 image, 32-bit RGB+alpha, non-interlaced
  CRC error in chunk IHDR (computed fcc410a8, expected c1d0b3e4)
ERRORS DETECTED in IMG_0707.png

E:\study&life\ctf\CtfSoftwave\CTF工具合集\隐写\图像隐写\pngcheck-2.3.0-win32>
```


我们可以发现检查出IHDR块出现问题，那么我们就尝试更改IHDR块中图片的长度。同样，我们使用010Editor或者winhex打开图片，找到长度限制位，修改其长度后保存：



这时候我们再次打开图片，就可以看到隐藏的flag了：



小结：

这些就是PNG图片的一些常规隐写，只要利用好PNG图片的工具，我们可以轻松解题，当然PNG图片还可以进行LSB隐写，我将在最后和BMP图片格式一起讨论LSB隐写的一些简单的情况。

三.其他类型的图片隐写

1.GIF图片的隐写

关于gif图片的隐写一般比较简单，只要用好工具，解题不会太难。以下是我总结的一些解题思路：

1.用binwalk查看图片发现格式是gif时，若图片后缀不是gif格式，请改为.gif的后缀，再做题。

2.flag往往是GIF图片的一帧或者几帧，这时候我们需要用到一个GIF查看器：Namo Gif Animator，这个工具可以一帧一帧的查看GIF图片。

3.GIF图片格式可以参考这篇博客：http://dev.gameres.com/Program/Visual/Other/GIFDoc.htm，有时候ctf会给一个.gif后缀的文件，但是却我们打不开它或者显示图片损坏，这时候我们就需要从格式入手，查看格式是否有损坏，比如GIF的头部标识。这时候我们需要修复头部信息，再根据修复成功的GIF图片解题。

2.使用PS、Matlab等工具锐化

用图片处理工具调整处理图片，有一些flag也许就在图片中，但是需要我们调好亮度，锐化等之后，才会显现出flag。

四.LSB隐写（初阶）

LSB隐写就是修改RGB颜色分量的最低二进制位（LSB），每个颜色都会有8bit，LSB隐写就是修改了像数中的最低的1Bit，这是对图片影响十分小，而人类的眼睛不会注意到这前后的区别，每个像素可以携带3Bit的信息，这样就把信息隐藏起来了。LSB隐写常见于：BMP图片和PNG图片。

关于lsb隐写，本人由于学艺不精，无法做到像众大佬一般，随手写一个python脚本把最低位取出来解出flag，所以我只能当个脚本小子，用别人写好的脚本来解一些初阶的题。

下面推荐一款看lsb痕迹的神器，Stegsolve。这款软件能够帮助我们辅助分析图片的各个通道位。

我们来看一道简单的LSB隐写题来结束此次的初阶隐写之旅：

这是一张PNG图片的LSB隐写，我们直接使用神器，使用Stegsolve打开图片：

往左或者往右查看各个通道位的情况，在Gray bits中就可以发现图片中隐藏着一个二维码，扫描这个二维码，我们就可以直接得到flag了：

总结：

这些是我在隐写学习中学到的一些图片解题思路和技巧，隐写术不仅仅止于图片隐写，还有音频、视频、可执行文件等的隐写加密。如果想要更深入的了解隐写术，这里推荐一本书籍：《数据隐藏技术揭秘：破解多媒体、操作系统、移动设备和网络协议中的隐秘数据》，这本书籍能够让你的隐写术更上很多层楼。这里分享这本书的云盘链接：https://pan.baidu.com/s/16PfOjijmmMOAE--wj5SHog 密码：8nd2。