CTF题目记录2（图片隐写）

原创

kkzz1x 于 2020-05-15 00:40:40 发布 1553 收藏 3

分类专栏： CTF题目记录

本文链接：https://blog.csdn.net/qq_39679772/article/details/105974659

网上找了一个图片隐写的练习，故记录一下

题目1

png图片–LSB隐写

习惯性要先查看属性–没收获

然后stegsolve

我当然是用data extract查看了各个通道的最后两位情况，并保存了几个文件下来试试。。。但是也没有收获

没想到直接浏览最后一位的图像即可找到一些东西

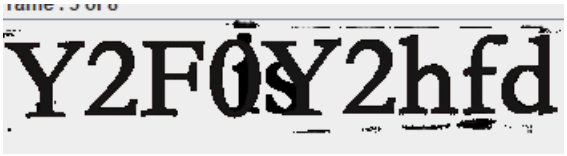
扫了一下获得flag{AppLeU0}

题目2

一张打不开的gif
考察文件格式吧
winhex打开，发现文件头出缺损，把文件头补全即可修复

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	ANSI ASCII
47	49	46	38	39	61	A2	06	6B	04	F7	FF	00	20	20	20	GIF89a¢ k ÷ÿ
02	02	02	23	23	23	04	04	04	2B	2B	2B	21	21	21	06	### +++!!!
06	06	33	33	33	05	05	05	FE	FE	FE	28	28	28	27	27	333 þþþ(((''
27	2D	2D	2D	3C	3C	3C	51	51	51	30	2D	2E	CD	CD	CD	'---<<<QQQ0-.ÍÍÍ

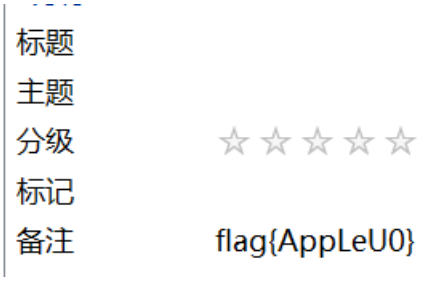
修复以后放入stegsolve
gif的最大考察点就是图片空间、时间的分离
先看下空间上的：按帧查看，发现得到flag



…不截全了…

题目3

jpg图片



论先查看属性的重要性
其实他是吧信息藏在了jpg头部exif部分

还有一些可能：
藏在尾部（属性里面看不到）
藏在里面的压缩包里

题目4双图 isg2014-misc200

经典题目~ png类型
双图的总结：两张图片相同或者有关联，可能考察运算 + - 异或之类
两张无联系：拼接

step1：先审查一下详细信息 winhex，属性 看看
step2：binwalk跑一下

```
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             PNG image, 1440 x 900, 8-bit/color RGB, non-interlaced
41            0x29            Zlib compressed data, default compression
1922524       0x1D55DC        PNG image, 1440 x 900, 8-bit/color RGB, non-interlaced
1922565       0x1D5605        Zlib compressed data, default compression
```


发现偏移处1D55DC处有图片，分离出来
offset 1D55DC 然后把一大块保存为新图片 2.png

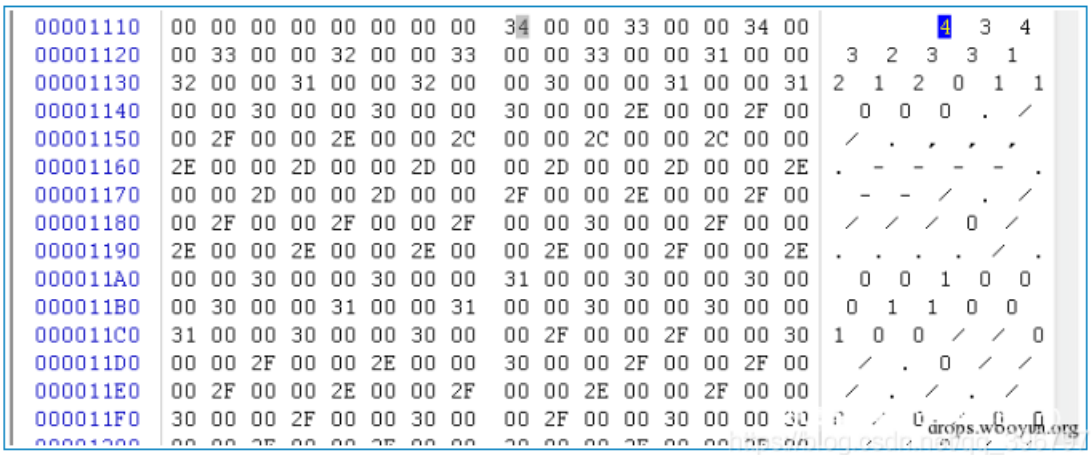
这两张图片没有什么太大的区别
利用linux下compare命令（啊我咋不行。。假linux吧5555555）
`compare 1.png 2.png diff.png` 观察一下发现了左下角有异常，png图片像数保存是从左到右，从下往上排列的。

或者下一个beyondcompare 虽然整体没什么不一样，但是仔细看边缘是有不一样之处的。。。

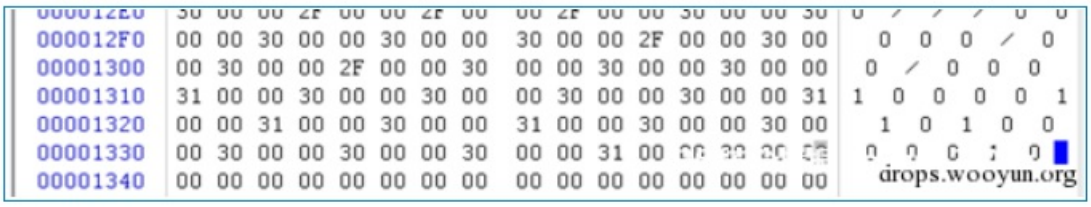
用stegsolve进行xor或者sub运算
把结果保存成solved.bmp。
然后把2.png保存成2.bmp 24位位图的格式，这个是因为png图片经过了压缩，不好直接对比每个字节，而bmp图片是没有压缩的，直接保存各个像数点的数据。

- 到这步我就陷入迷茫QAQ，以下是转载了师傅的wp
- 总的来说我觉得这题有很多可以学习的地方 大概都是一些我不会的常规操作

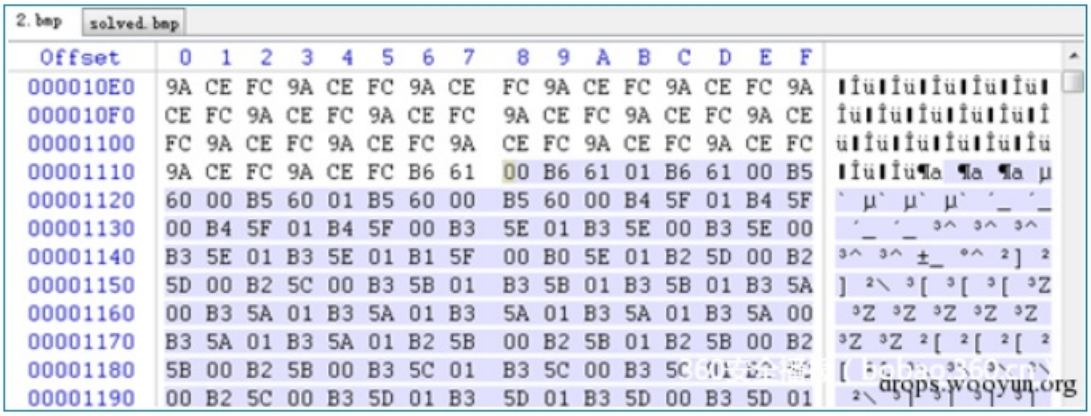
这个题还有一个坑点就是偏移的问题 png图片的扫描是从左向右，从下往上来的。而坑的是这个图的信息隐藏并没有在一开头的像数，而是是第二行像数，所以就需要利用bmp的优势，储存无压 缩，方便寻找到偏移，从而找到信息隐藏的地方。利用winhex打开，黑色的像数的在bmp中的hex的00保存的，那么我们就寻找不是00的地方。在偏 移0x1110的地方可以发现



有不是00的字节，一开始还以为这些就是flag的信息了，后来才发现是因为两个图片sub影响到了效果，真正的信息是隐藏在2.png中的，所以 打开由2.png转换的2.bmp来对，通过之前diff得到的偏移，寻找到0x1110的地方，直到0x1330结束，这是隐藏的信息。



图片24.png



图片25.png

只保留00 01，这个是因为RGB的关系，只隐藏在R通道里面了，其他通道都是图片的正常像数信息，过滤掉就可以了。

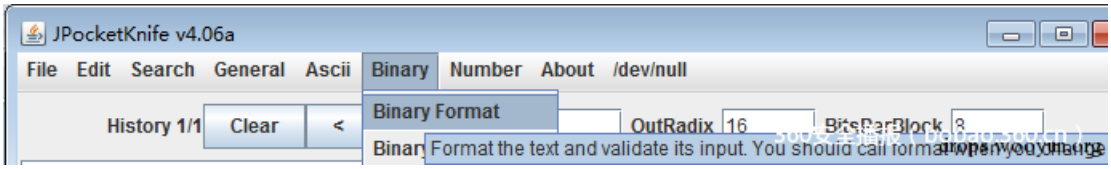
```
00010000010000010001000100000101000100000000101010001010101000101000100000001 0
00100000101000100000000101010000010100010001010000010001000101010101000100010 0
00010100010101000100000001000000010001000101000001010100000101000100000001010
00101010000000101000000000001010000010101000100010000010000000101000100000001
01010000000000010000010000000001010101000001000101010101000 1
```

观察一下可以发现，而奇数位都是0，是多余的，把这些去除掉。直接把00 替换成0，01替换成1就可以了。

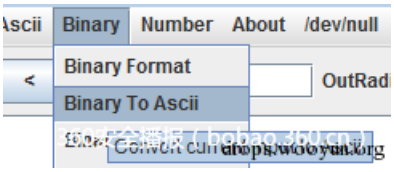
```
01001001010100110100011101111011010001010011010001110011010110010101111101010
01101110100010001010110011100110100011011100011000001100111010100100011010001
11000001001000011110010111110 1
```

得到了这个之后，可以发现他的长度是184，是8的倍数，把他转换成ascii码就可以了。可以使用JPK工具来进行转换，工具的下载的链接是www.wechall.net/applet/JPK_406.jar。

对比2.bmp可以发现隐藏了一些00 01这些信息，把这一部分扣出来。



JPK——binary——binary to ascii



就得到了flag，ISG{E4sY_StEg4n0gR4pHy}

这种就是利用的两张图片对比来寻找差异，从而找到信息隐藏的地方，这样子出题往往是因为一张图片能提供的信息太少。