

隐写术总结

转载

[weixin_30456039](#) 于 2017-08-20 01:21:00 发布 49 收藏

文章标签: [python 操作系统](#)

原文链接: <http://www.cnblogs.com/jiaxinguoguo/p/7398491.html>

版权

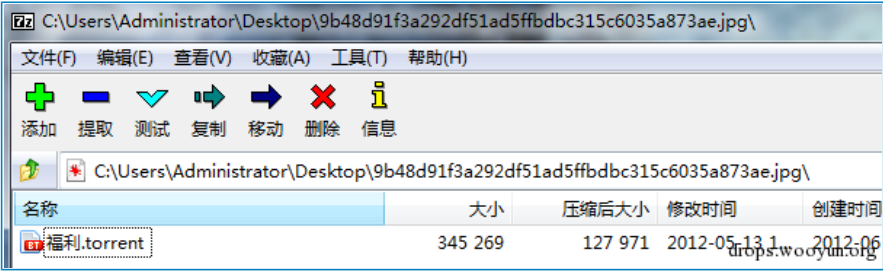
[隐写术总结](#)

0x00 前言

之前还没有见到drops上有关于隐写术的总结,我之前对于隐写术比较有兴趣,感觉隐写术比较的好玩。所以就打算总结总结一些隐写术方面的东西。写的时候,可能会有错误的地方,请不吝赐教,谢谢。

本篇章中用到的[隐写术的图片](#),都打包在了这里:隐写术图片,想去自己尝试一遍的话可以去下载。

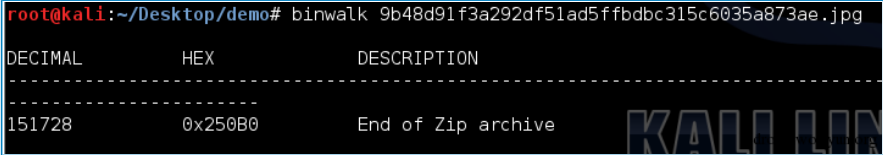
最开始接触到隐写术,是看到一种叫做图种的东西,当时不懂,只说要另存为zip,然后解压出来就可以了,当时觉得特别神奇,就像发现了新大陆,然后就尝试了一下,发现可以用另存为zip的方式,用7z或者是winzip等工具打开,然后就可以看到福利了。



后来才懂得了,先制作一个1.zip,把想要隐藏的东西放进去,再需要一张jpg图片2.jpg,然后就可以执行一个命令 `copy /b 2.jpg+1.zip output.jpg`。就可以得到一张图种,这是利用了copy命令,将两个文件已二进制方式连接起来,生成output.jpg的新文件。而在jpg中,是有结束符的,16进制是FF D9,利用winhex可以看到正常的jpg结尾都是FF D9的,图片查看器会忽视jpg结束符之后的内容,所以我们附加的zip,自然也就不会影响到图像的正常显示。

000036C0	B9 6D 82 ED 60 AD 8F 52 6A 2E 7F E7 A3 7F DF 34	算傢`?.Rj..??8?涕
000036D0	5A 31 92 D9 1D F0 59 BA 9C 75 A9 F7 9F 45 FC 85	Z1捺.?Y?渦 炆u...
000036E0	0D 59 D8 6D 1F FF D9	.Y穀.?? drops.wooyun.org

这种类型的隐写也是比较容易被发现的,如果发现是jpg图片的话,观察文件结束符之后的内容,查看是否附加的内容,正常图片都会是FF D9结尾的。还有一种方式来发现就是利用binwalk这个工具,在kali下自带的一个命令行工具。



利用binwalk可以自动化的分析图片中附加的其他的文件,其原理就是检索匹配文件头,常用的一些文件头都可以被发现,然后利用偏移可以配合winhex或者是dd分割出隐藏的部分。

0x01 修改数据

上面说到的隐藏方式,是利用了增加数据的方式,把数据直接增加在了jpg后面。还有另一类隐藏的方法,就是利用了修改数据的方式来隐藏自己传递的信息。

一种常见的方式是利用LSB来进行隐写，LSB也就是最低有效位 (Least Significant Bit)。原理就是图片中的像数一般是由三种颜色组成，即三原色，由这三种原色可以组成其他各种颜色，例如在PNG图片的储存中，每个颜色会有8bit，LSB隐写就是修改了像数中的最低的1bit，在人眼看来是看不出来区别的，也把信息隐藏起来了。譬如我们想把'A'隐藏进来的话，如下图，就可以把A转成16进制的0x61再转成二进制的01100001，再修改为红色通道的最低位为这些二进制串。

颜色	二进制	十进制
红	11111110	254
绿	00000000	0
蓝	00000000	0

红

drops.wooyun.org

十进制	最低有效位 十六进制 ASCII码
11111110	0
00000001	1
00000001	1
11111110	0
00000000	0
00000000	0
00000000	0
00000000	0
00000001	1

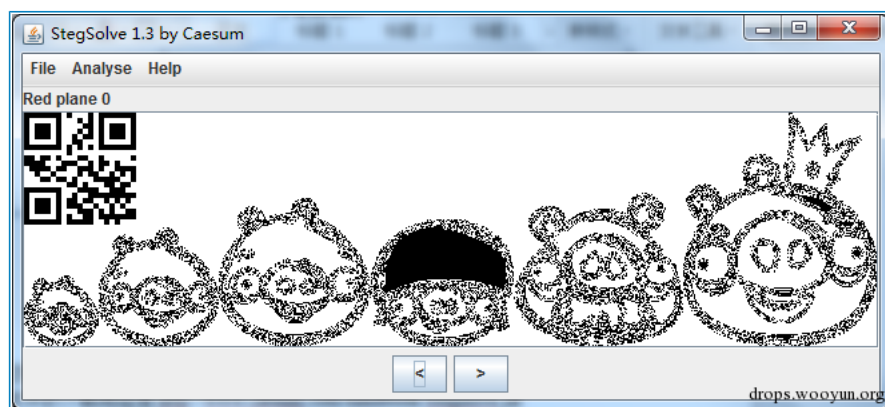
01100001 = 0x61 = A

drops.wooyun.org

如果是要寻找这种LSB隐藏痕迹的话，有一个工具是个神器，可以用来辅助我们进行分析。Stegsolve这个软件的下载地址是

<http://www.caesum.com/handbook/Stegsolve.jar>

打开之后，使用Stegsolve——Analyse——Frame Browser这个可以浏览三个颜色通道中的每一位，可以在红色通道的最低位，发现一个二维码，然后可以扫描得到结果。



再解一下qrcode，用在线的就可以<http://tool.chinaz.com/qrcode/>，得到了flag{AppLeU0}，如果是隐写的使用了ascii的话，可以使用Stegsolve——Analyse——Data Extract来查看ascii码。

在这个过程中，我们要注意到，隐写的载体是PNG的格式，如果是像之前的jpg图片的话就是不行的，原因是jpg图片对像数进行了有损的压缩，你修改的信息可能会被压缩的过程破坏。而PNG图片虽然也有压缩，但却是无损的压缩，这样子可以保持你修改的信息得到正确的表达，不至于丢失。BMP的图片也是一样的，是没有经过压缩的，可以发现BMP图片是特别的大的，因为BMP把所有的像数都按原样储存，没有压缩的过程。

0x02 隐写与加密

我们先要区分一个概念，隐写术和加解密的区别。其实说起来很简单，加解密的话，就是会出现一些神秘的，可疑的字符串或者是数据之类的。而隐写术的话，就是信息明明就在你的面前，你却对他视而不见。隐写术在CTF中出现时，常常会和解密结合起来一起出现，或者是一些编码方式一起出现，以提高题目的难度。

用一个ctf的题目作为例子吧，iscc2014中有一个题目，给了一个名为 此为gif图片.gif的文件，打开发现了报错。有的时候，会需要我们去修复图片，这对我们对于图片的文件结构要有了解。找到gif的文件格式，然后对照这个破损的文件。Gif的图片格式文档可以查看这个链

接，<http://dev.gameres.com/Program/Visual/Other/GIFDoc.htm>

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	39	61	A2	06	6B	04	F7	FF	00	20	20	20	02	02	02	23	9a? k. ?#
00000010	23	23	04	04	04	2B	2B	2B	21	21	21	06	06	06	33	33	##...+++!!!...33
00000020	33	05	05	05	FE	FE	FE	28	28	28	27	27	27	2D	2D	2D	3... ? (''' ---
00000030	3C	3C	3C	51	51	51	30	2D	2E	CD	CD	CD	D3	D3	D3	46	<<<QQQ...中估明.org

用winhex打开，我们会发现他和普通的GIF图片不一样，头部缺少了东西，在对比一些文档，会发现是少了GIF8。

BYTE	7	6	5	4	3	2	1	0	BIT	
1									'G'	
2									'T'	GIF文件标识
3									'F'	
4									'8'	
5									'7'或'9'	GIF文件版本号：87a - 1987年5月 89a - 1989年7月
6									'a'	

drops.wooyun.org

我们手动修复一下，增加GIF8。

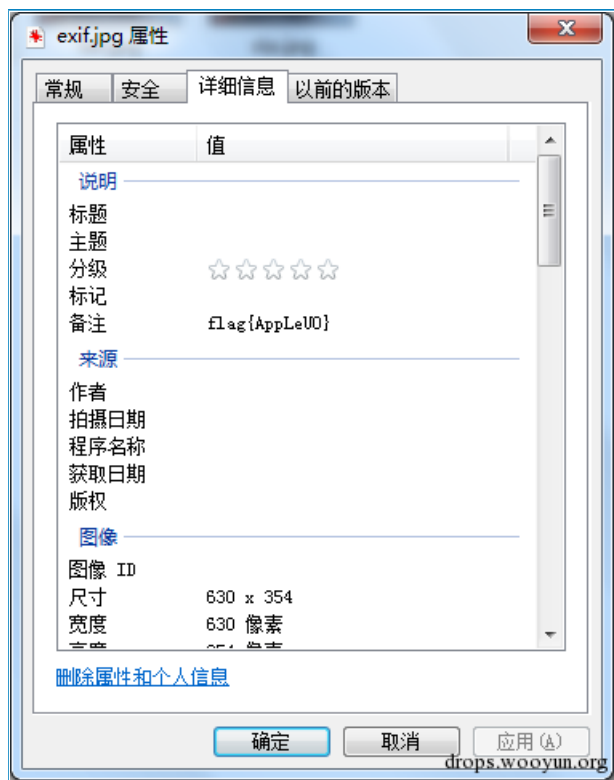
Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	47	49	46	38	39	61	A2	06	6B	04	F7	FF	00	20	20	20	GIF89a?kk. ? .
00000010	02	02	02	23	23	23	04	04	04	2B	2B	2B	21	21	21	06	...##...+++!!!
00000020	06	06	33	33	33	05	05	05	FE	FE	FE	28	28	28	27	27	..333.drops.wooyun.org

然后浏览图片后会发现，有个PASSWORD一闪而过，gif和别的图片最大的区别就是gif是动态图，它是可以由多帧组成的可以顺序播放的，有的题就是把播放的时间弄得特别慢，几乎就不会动的，所以我们可以用工具一帧一帧的观察图片。Stegsolve就带有这种功能。

Stegsolve——Analyse——Frame Brower就可以看到是有8帧的图片，有点重叠不太好观察，也可以用Namo_GIF_gr这个工具。得到了PASSWORD is Y2F0Y2hfdGhIX2R5bmFtaWNfZmxhZ19pc19xdW10ZV9zaW1wbGU=。很明显，这个时候PASSWORD是经过的编码的，我们可以看到字符范围是0-9a-Z结尾还有=，所以判断是base64编码，解码得到了catch_the_dynamic_flag_is_qumte_simple。这个就是和编码方式结合，传递一些可疑的数据，隐写术常常会与加解密或编码结合在一起，对一些常见的编码和加密方法也要了解，得到密文的字符范围和长度能发现这是什么加密或者是编码。

0x03 载体

数据在隐藏的时候，我们常常是需要先分析是数据隐藏在哪里，也就是他在利用是什么做载体，之后才可以进一步的分析是加密或编码的。这也就是说我们要对一个图片的格式要有了解，才能知道哪些地方是可疑的，哪些是可以隐藏起信息的，会有冗余的成分在。举个例子吧，比如给了一个jpg的图片。除了我们之前说到的隐藏在结束符之后的信息，jpg图片还可以把信息隐藏的exif的部分。exif的信息是jpg的头部插入了数码照片的信息，比如是用什么相机拍摄的。这些信息我们也是可以控制的，用查看属性的方式可以修改一部分的信息，还可以用exif编辑器来进行编辑。Power_exif这个可以用来编辑。



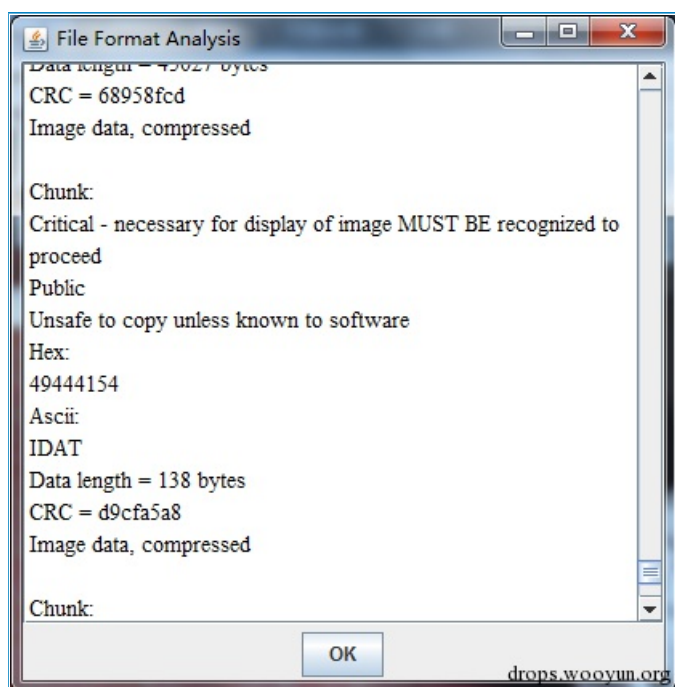
可以看到flag{AppLeU0}，就是需要了解隐藏信息的地方，隐写术有的时候难，就是难在了一张图片有太多的地方可以隐藏信息了，有的时候根本连隐藏的载体都找不到，在你的眼里他就是一张正常的图片。

0x04 编程辅助

有一些情况下，我们也是没有现成的工具来完成的，可以自己写一些简单的程序来辅助我们进行分析，或者是加解密。比如sctf的misc400的题目，就需要用到一些简单的编程。题目给出了一个png图片，需要我们找到有SCTF{}标志的flag。

这个题需要我们对于png图片的格式有一些了解，先用stegsolve查看一下，其他的LSB之类的并没有发现什么问题，然后看了一下结构发现，有一些异常的IDAT块。IDAT是png图片中储存图像像数数据的块。Png图片格式的扩展阅读可以看看这篇

<http://www.cnblogs.com/fengyv/archive/2006/04/30/2423964.html>



可以用pngcheck来辅助我们观察，可以看得更加清晰。pngcheck.exe -v scf.png

```
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
```

可以看到，正常的块的length是在65524的时候就满了，而倒数第二个IDAT块长度是45027，最后一个长度是138，很明显最后一个IDAT块是有问题的，因为他本来应该并入到倒数第二个未满足的块里。

1421296	95 8F CD 00 00 00 8A 49 44 41 54 78 9C 5D 91 01	??[...?IDATx?]?.?
1421312	12 80 40 08 02 BF 04 FF FF 5C 75 29 4B 55 37 73	.@...?. u)KU7s
1421328	8A 21 A2 7D 1E 49 CF D1 7D B3 93 7A 92 E7 E6 03	? .I[?] ?換擊?
1421344	88 0A 6D 48 51 00 90 1F B0 41 01 53 35 0D E8 31	?mHQ...癆.S5.?.
1421360	12 EA 2D 51 C5 4C E2 E5 85 B1 5A 2F C7 8E 88 72	.?-Q嚮伎觀Z/莢垠
1421376	F5 1C 6F C1 88 18 82 F9 3D 37 2D EF 78 E6 65 B0	?o??賃=7-?x?e?
1421392	C3 6C 52 96 22 A0 A4 55 88 13 88 33 A1 70 A2 07	肱R?"? ? ? ?
1421408	1D DC D1 82 19 DB 8C 0D 46 5D 8B 69 89 71 96 45	.?禪.?? F]嬬%q脛
1421424	ED 9C 11 C3 6A E3 AB DA EF CF C0 AC F0 23 E7 7C	頊.??j?«陸俠 #?
1421440	17 C7 89 76 67 D9 CF A5 A8 00 00 00 00 49 45 4E	.?塲g?膝'....IEN
1421456	44 AE 42 60 82	D?B`

drops.wooyun.org

我们用winhex把这一部分异常的IDAT块给扣出来。然后就是要研究研究这个块是什么情况，发现了载体之后就是要想办法找出他的规律。观察那一部分的数据，可以看到是16进制的78 9C开头的，百度一下分析是zlib压缩的标志。在png的百度百科里也可以查到PNG的IDAT是使用从LZ77派生的无损数据压缩算法，可以用zlib解压。那么就尝试用zlib来解一下这段数据。Zlib的扩展阅读<http://zlib.net/>

我们使用python来编程，先把那段数据处理一下，保存成16进制的。

80	28 4A 80 00 00 00 8A 49 44 41 54 78 9C 5D 91 01	(J€...\$IDATx?]?.
96	12 80 40 08 02 BF 04 FF FF 5C 75 29 4B 55 37 73	.@...?. u)KU7s
112	8A 21 A2 7D 1E 49 CF D1 7D B3 93 7A 92 E7 E6 03	? .I[?] ?換擊?
128	88 0A 6D 48 51 00 90 1F B0 41 01 53 35 0D E8 31	? mHQ...癆.S5.?.
144	12 EA 2D 51 C5 4C E2 E5 85 B1 5A 2F C7 8E 88 72	.?-Q嚮伎觀Z/莢垠
160	F5 1C 6F C1 88 18 82 F9 3D 37 2D EF 78 E6 65 B0	? o?? 賃=7-?x?e?
176	C3 6C 52 96 22 A0 A4 55 88 13 88 33 A1 70 A2 07	肱R?"? ? ? ?
192	1D DC D1 82 19 DB 8C 0D 46 5D 8B 69 89 71 96 45	.?禪.?? F]嬬%q脛
208	ED 9C 11 C3 6A E3 AB DA EF CF C0 AC F0 23 E7 7C	頊.??j?«陸俠 #?
224	17 C7 89 76 67 D9 CF A5 A8 00 00 00 00 49 45 4E	.?塲g?膝'....IEN
240	44 AE 42 60 82	D?B`

drops.wooyun.org

得到16进制的以方便python处理，前面的4字节是长度 然后是标志位IDAT 然后开始是数据，直到 D9 CF A5 A8 是crc32校验位。 所以实际的数据是：

```
789C5D91011280400802BF04FFFF5C75294B5537738A21A27D1E49CFD17DB3937A92E7E603880A6D485100901FB0410153350DE83112
EA2D51C54CE2E585B15A2FC78E8872F51C6FC1881882F93D372DEF78E665B0C36C529622A0A45588138833A170A2071DDCD18219DB8C
0D465D8B6989719645ED9C11C36AE3ABDAEFCFC0ACF023E77C17C7897667
```

然后用python来写zlib解压



```
#!/usr/bin/env python
import zlib
import binascii IDAT =
"789C5D91011280400802BF04FFFF5C75294B5537738A21A27D1E49CFD17DB3937A92E7E603880A6D485100901FB0410153350DE8311
2EA2D51C54CE2E585B15A2FC78E8872F51C6FC1881882F93D372DEF78E665B0C36C529622A0A45588138833A170A2071DDCD18219DB8
C0D465D8B6989719645ED9C11C36AE3ABDAEFCFC0ACF023E77C17C7897667".decode('hex') #print IDAT result =
binascii.hexlify(zlib.decompress(IDAT)) print result #print result.decode('hex')
```



发现解出来了一些3031的字符串，30和31是hex的 0和1的编码，再解一次hex得到一串625长度的01字符串。



```
1111111000100001101111111000001011100101101000001101110101000000000101110110111010010000000010111011011010
1110110100101110110000010101011011010000011111110101010101011111100000000101110111000000001101001100000101
00111011011110101010010000111000000000010100000000100100110100010011100111101110011110000111011111000110010
10001100111000010101000110100011110101100000101000101100000110111011001000011100111001000010111111101000000
01101010010001111011111101110000110101101110000010000110011000111101011101000110100111110000101110101100011
1010011100101110100100111011011000110000010110001101000110001111111011010110111011011
```



得到的01 串的长度是625，除以8 除以7 都无法整除，也就是说没法直接转换成ascii码。

```
>>> 625.0/8.0
78.125
>>> 625.0/7.0
89.28571428571429
```

然后发现 $625 = 25 \times 25$ ，刚好是个正方形的形状，那么尝试一下 把这些01 组成一个正方形 看看是什么，可以用python的PIL编程可以很方便的画图，在kali自带就可以有，win的环境需要安装PIL的第三方库。



```
#!/usr/bin/env python
import Image
MAX = 25
pic = Image.new("RGB", (MAX, MAX)) str =
"1111111000100001101111111000001011100101101000001101110101000000000101110110111010010000000010111011011101
0111011010010111011000001010101101101000001111111010101010101111110000000010111011100000000110100110000010
10011101101111010101000011100000000001010000000010010011010001001110011110111001111000011101111100011001
010001100111000010101000111010110000010100010110000011011101100100001110011100100001011111110100000
0011010100100011101111110111000011010110111000001000011001100011110101110100011010011111000010111010110001
11010011100101110100100111011011000110000010110001101000110001111111011010110111011011" 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")
```



发现是一个二维码 可以编码来画出 0代表了是白色 而1代表了黑色，然后可能会需要旋转来调整一下，才能扫描出来。处理一下得到了一个二维码。然后扫描得到了flag。



操作的类型: ☐ 生成 ☒ 解码

二维码图片:

解码数据为: `SCTF{(121.518549,25.040854)}`

drops.wooyun.org

SCTF{(121.518549,25.040854)}, 成功得到了flag。

在有的情况下，是没法用现成的工具来处理的，所以就要我们用编程来设法解决。Python的PIL是个好东西。批量处理图片的时候可能会需要它。

0x05 双图

还有一种情况是比较特殊的，有的时候会给出两张图片，或者是需要你去寻找原来的图片来进行对比寻找隐藏的信息。这个一般是因为一张图片给出来的隐藏信息太过于隐蔽，无法找不到具体的位置，具体的信息。这个时候就要用到一些对比的技巧来查找了。比如ISG2014的misc200就是用到的这种给出了两张图的。有的情况下，第二张图是需要你自己去找到的。

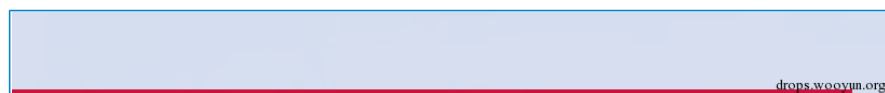
我们来看isg2014-misc200的题，题目给了一张png图片，png的图片，就怕里面插个什么rar之类的，所以先用linux下的binwalk命令跑一跑。

```
root@kali:~/Desktop/isg# binwalk final.png
DECIMAL      HEX      DESCRIPTION
-----
0            0x0      PNG image, 1440 x 900, 8-bit/color RGB, non-interlaced
1922524      0x1D55DC PNG image, 1440 x 900, 8-bit/color RGB, non-interlaced
```

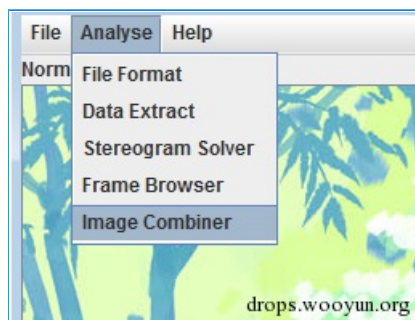
跑一跑，发现了有两个PNG图片，binwalk会给出偏移，确定了偏移是0x1D55DC之后，用winhex把图片扣出来，保存成2.png。原来的图final.png删除后面那的一部分，保存成1.png。肉眼查看了一下，发现两张图片没有太大的区别，我们用软件来帮助我们区分他。

001D55D0	00 00 00 00 49 45 4E 44 AE 42 60 82 89 50 4E 47	IEND0B` PNG
001D55E0	0D 0A 1A 0A 00 00 00 0D 49 48 44 52 00 00 05 A0	IHDR
001D55F0	00 00 03 84 08 02 00 00 00 D8 2F 01 85 00 01 00	0/1
001D5600	00 49 44 41 54 78 9C BC FD 57 B3 6C 4B 92 1E 88	IDATx myw1K'
001D5610	B9 88 88 25 52 6D 71 F4 B9 AA EA 56 75 55 0B 34	%mgô'âëVüU 4
001D5620	80 86 26 7B 0C 63 E0 D0 86 0F 63 9C 67 3E F0 F7	&{ càDl c g>ð÷
001D5630	F1 85 C6 17 92 83 01 8D 30 62 80 69 88 6E 4C 8B	ñ Æ 'l Ob l nLl
001D5640	EA AE EA 52 57 1F BD 65 8A A5 22 C2 DD F9 B0 32	é0ëBw ke W"áý.º2
001D5650	73 F7 D9 67 9F DB 55 40 01 61 D7 CF CD BD 73 ED	drops.wooyun.org

用linux下的命令可以进行对比，生成一个有差异的图片diff.png。compare 1.png 2.png diff.png 观察一下发现了左下角有异常，png图像像素保存是从左到右，从下往上排列的。



发现了左下的第二条像素有异常，对比一下1.png 2.png发现了2.png有问题 那么我们可以用神器stegsolve来辅助，stegsolve——Analyse——Image Combiner对比两个文件。查看Sub或Xor，可以发现左下角，第二条像素条是有异常的，有红色的出现。



把1.png和2.png进行一下sub方法 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，直接保存各个像素点的数据。

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

00001110	00 00 00 00 00 00 00 00 34 00 00 33 00 00 34 00	3 3 4
00001120	00 33 00 00 32 00 00 33 00 00 33 00 00 31 00 00	3 2 3 3 1
00001130	32 00 00 31 00 00 32 00 00 30 00 00 31 00 00 31	2 1 2 0 1 1
00001140	00 00 30 00 00 30 00 00 30 00 00 2E 00 00 2F 00	0 0 0 . /
00001150	00 2F 00 00 2E 00 00 2C 00 00 2C 00 00 2C 00 00	/ . / . /
00001160	2E 00 00 2D 00 00 2D 00 00 2D 00 00 2D 00 00 2E	. - - - - .
00001170	00 00 2D 00 00 2D 00 00 2F 00 00 2E 00 00 2F 00	- - / . /
00001180	00 2F 00 00 2F 00 00 2F 00 00 30 00 00 2F 00 00	/ / / 0 /
00001190	2E 00 00 2E 00 00 2E 00 00 2E 00 00 2F 00 00 2E	 / .
000011A0	00 00 30 00 00 30 00 00 31 00 00 30 00 00 30 00	0 0 1 0 0
000011B0	00 30 00 00 31 00 00 31 00 00 30 00 00 30 00 00	0 1 1 0 0
000011C0	31 00 00 30 00 00 30 00 00 2F 00 00 2F 00 00 30	1 0 0 / / 0
000011D0	00 00 2F 00 00 2E 00 00 30 00 00 2F 00 00 2F 00	/ . 0 / /
000011E0	00 2F 00 00 2E 00 00 2F 00 00 2E 00 00 2F 00 00	/ / . / .
000011F0	30 00 00 2F 00 00 30 00 00 2F 00 00 30 00 00 30	0 / 0 0 0 0 0
00001200	00 00 2F 00 00 2E 00 00 30 00 00 2F 00 00 2F 00	/ / . / .

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。

000012E0	30 00 00 2F 00 00 2F 00 00 2F 00 00 30 00 00 30	0 / / / 0 0
000012F0	00 00 30 00 00 30 00 00 30 00 00 2F 00 00 30 00	0 0 0 / 0
00001300	00 30 00 00 2F 00 00 30 00 00 30 00 00 30 00 00	0 / 0 0 0
00001310	31 00 00 30 00 00 30 00 00 30 00 00 30 00 00 31	1 0 0 0 0 1
00001320	00 00 31 00 00 30 00 00 31 00 00 30 00 00 30 00	1 0 1 0 0
00001330	00 30 00 00 30 00 00 30 00 00 31 00 00 30 00 00	0 0 0 1 0
00001340	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
000010E0	9A	CE	FC	9A	CE	FC	9A	CE	FC	9A	CE	FC	9A	CE	FC	9A	iiiiiiiiiiiiiiii
000010F0	CE	FC	9A	CE	FC	9A	CE	FC	9A	CE	FC	9A	CE	FC	9A	CE	iiiiiiiiiiiiiiii
00001100	FC	9A	CE	FC	9A	CE	FC	9A	CE	FC	9A	CE	FC	9A	CE	FC	iiiiiiiiiiiiiiii
00001110	9A	CE	FC	9A	CE	FC	B6	61	00	B6	61	01	B6	61	00	B5	iiiiii9a 9a 9a μ
00001120	60	00	B5	60	01	B5	60	00	B5	60	00	B4	5F	01	B4	5F	μ μ μ _ _
00001130	00	B4	5F	01	B4	5F	00	B3	5E	01	B3	5E	00	B3	5E	00	_ _ 3^ 3^ 3^
00001140	B3	5E	01	B3	5E	01	B1	5F	00	B0	5E	01	B2	5D	00	B2	3^ 3^ ± 3^ 2] 2
00001150	5D	00	B2	5C	00	B3	5B	01	B3	5B	01	B3	5B	01	B3	5A	] 2\ 3[3[3[3Z
00001160	00	B3	5A	01	B3	5A	01	B3	5A	01	B3	5A	01	B3	5A	00	3Z 3Z 3Z 3Z 3Z
00001170	B3	5A	01	B3	5A	01	B2	5B	00	B2	5B	01	B2	5B	00	B2	3Z 3Z 2[2[2[2
00001180	5B	00	B2	5B	00	B3	5C	01	B3	5C	00	B3	5C	01	B2	5C	[2[3\ 3\ 3\ 2\
00001190	00	B2	5C	00	B3	5D	01	B3	5D	01	B3	5D	00	B3	5D	01	2\ 3[3[3[3[

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。

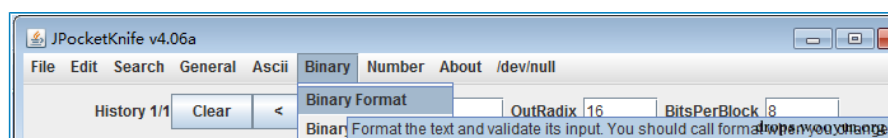
00010000010000010001000100000101000100000001010100010101010001010001000000010001000001010001000000101010000
010100010001010000010001000101010101000100000101000101010001000000010000000100010001010000010100000101
000100000001010001010100000001010000000000101000001010100010000010000000101000100000001010100000000001
000001000000000101010100000100010101010001

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。

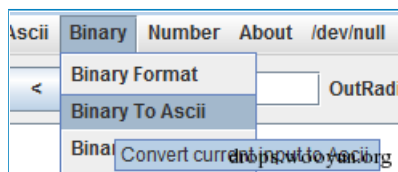
010010010101001110100011101111011010001010011010001110011010110010101111010011011101001000101011001110011
010001101110001100000110011101010010001101000111000001001000011110010111101

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。

0x06 后记

这个总结其实还是缺很多的，因为隐写术能写的东西太多了，比如jpg的冗余信息的压缩也可以隐藏进信息，还有其他的多媒体文件也可以进行隐写，例如音频文件，视频文件等等，有很多东西可以研究。一开始是觉得隐写术特别的有趣才接触到的，就像是在藏宝寻宝一样，特别好玩，希望你们也可以感受到这种快乐。

posted on 2017-08-20 01:21 [夹心果果](#) 阅读(...) 评论(...) [编辑](#) [收藏](#)

转载于:<https://www.cnblogs.com/jiaxinguoguo/p/7398491.html>