



2020

图片隐写

chendong1@venustech.com.cn

启明星辰网络安全学院-培训教学部

陈栋



什么是隐写术？

隐写术（Steg）就是一种保密通信技术，将秘密信息隐藏于公开的普通载体中进行传送，实现隐蔽通信，可以让计划者之外的人即使得到传递的信息，也不知道隐秘的数据，从而达到安全传递秘密消息的目的，保证通信安全。

讲到保证通信安全，还能想到什么？





加解密与隐写术的区别：

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，第三方是知道数据被加密的，但是却不知道是什么加密方式。通俗点就是看得见摸得着，但是就是吃不到；

而隐写术的话，则着重于让第三方无法察觉数据的存在性。就是信息明明就在你的面前，你却对它视而不见。





CTF隐写术常用的信息载体类型：

1. 图片文件（jpeg/png/gif...）
2. 文档文档（txt/pdf/doc..）
3. 音频文件（mp3/wav...）

既然隐写术是以文件为载体，那我们通常如何判断一个文件的类型呢？





认识常见文件头标识

FFD8FF	-----	JPEG (jpg)
89504E47	-----	PNG (png)
47494638	-----	GIF (gif)
424D	-----	Windows Bitmap (bmp)
504B0304	-----	ZIP Archive (zip)
52617221	-----	RAR Archive (rar)
57415645	-----	Wave (wav)
41564920	-----	AVI (avi)
D0CF11E0	-----	MS Word/Excel (xls.or.doc)
255044462D312E	---	Adobe Acrobat (pdf)





- Binwalk 帮助我们识别文件结构
- Foremost 将多个合并文件分离
- Exiftool 读取jpeg图片的exif信息。
- Pngcheck 查看png图片模块信息。
- MP3Stego 可以将wav文件和需要隐藏的文件合并成一个新的MP3文件
- Stegosolve 神器，通常使用frame browser功能来查看图片不同通道，不同色块来分析图片隐藏信息。注意：运行需要Java环境



目录

常见隐写类型



启明星辰
网络空间安全学院



文件尾部插入



文件中部插入



插入压缩包



双图隐写



搜索文件尾



LSB隐写



JPG的Exif隐写



PNG图片隐写



GIF图片隐写



特殊工具隐写





常见隐写手法



- 图片隐写常见有两种：
- 插入
 - 插入往往利用文件格式的无关数据或者空白区域，放置需要的数据，不会改变原始数据，只是增加了隐写的内容
- 替换
 - 替换的经典例子就是LSB替换方法，把每个字节最低有效位变换，不会改变文件大小，但是源文件发生了变化





插入类型



- 图片尾部插入特殊字符串
- 这种类型的隐写一般最简单也最容易发现，所以一般会配合编码来增加难度。
- 方法：winhex打开图片，在末尾添加特殊字符串





图片尾部插入



- 例如我们知道正常JPG图片都会是FF D9结尾的

000041E0	8B 7D FB B0 AD 17 8F E6 7A D2 22 BF 2F C3 CB E3	< }û°- æzÒ"¿/ÃĖă
000041F0	6E C5 1B 58 DA 6A BA 22 DE 7A 67 45 07 92 22 5F	nĂ XÚj°"ĤzgE ' "
00004200	48 8D 57 12 4A D6 71 2E 7D 13 D5 11 73 64 E9 C1	H W JŎq.} Ŏ sdéĂ
00004210	3D CE FF 00 99 5D A0 34 50 44 4C 4C 90 E2 6E 95	=îÿ ð] 4PDLL ân•
00004220	D8 05 5A 22 D3 0F 6C F2 F4 92 AA 88 B6 64 AA AA	Ø Z"Ó lðô' ã^qđ==
00004230	22 0F FF D9	" ÿÙ

- 在结尾插入一段base64编码的字符串。

00016912	3D CE FF 00 99 5D A0 34 50 44 4C 4C 90 E2 6E 95	=îÿ ð] 4PDLL ân•
00016928	D8 05 5A 22 D3 0F 6C F2 F4 92 AA 88 B6 64 AA AA	Ø Z"Ó lðô' ã^qđ==
00016944	22 0F FF D9 64 32 56 73 59 32 39 74 5A 58 52 76	" ÿÙd2VsY29tZXRV
00016960	64 6D 56 75 64 58 4D 3D	dmVudXM=





● 图片中间插入特殊字符串

这类隐写一般以JPG格式图片为载体，先认识JPEG格式图片组成：标记码和压缩数据。标记码高字节固定为0xFF，标记码之间有冗余字节。意味着如果在标记码之间插入隐秘数据，不会影响图片正常打开。

```
SOI : FF D8 // 图片起始
APP0 : 0xFF E0 // 标记号
        APP0 SIZE : 1D 23 //当前标记的长度
        JFIF Flag : JFIF // JFIF 标识
        VERSION : // 版本号
        ATTRIBUTION: // 长宽、DPI等信息
DQT : 0xFFDB //Define Quantization Table, 定义量化表
SOF0 : 0xFFC0 //Start of Frame, 帧图像开始
DHT : 0xFFC4 //Difine Huffman Table, 定义哈夫曼表
SOS : 0xFFDA // Start of Scan, 扫描开始 12字节
压缩数据
EOI : FF D9 // 图片结束
```





图片中间插入



- 标记码之间插入数据

00000000	FF D8 FF E0 00 10 4A 46 49 46 00 01 01 01 00 48	ÿøÿà JFIF H
00000016	00 48 00 00 FF DB 00 43 00 0C 08 09 0A 09 07 0C	H ÿÜ C
00000032	0A 09 0A 0D 0C 0C 0E 11 1D 13 11 10 10 11 23 19	#
00000048	1B 15 1D 2A 25 2C 2B 29 25 28 28 2E 34 42 38 2E	*%,(+)%((.4B8.
00000064	31 3F 32 28 28 3A 4E 3A 3F 44 47 4A 4B 4A 2D 37	1?2 ((:N:?DGJKJ-7
00000080	51 57 51 48 56 42 49 4A 47 FF DB 00 43 01 0C 0D	QWQHVB1JGÿÜ C
00000096	0D 11 0F 11 22 13 13 22 47 30 28 30 47 47 47 47	" "G0 (OGGGG
00000112	47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47	GGGGGGGGGGGGGGGGGG
00000128	66 6C 61 67 7B 77 65 6C 63 6F 6D 65 74 6F 76 65	flag{welcometove
00000144	6E 75 73 7D 47 47 47 47 47 47 47 47 47 47 FF C0	nus}GGGGGGGGGGÿà
00000160	00 11 08 01 75 02 30 03 01 22 00 02 11 01 03 11	u o "
00000176	01 FF C4 00 1C 00 01 00 02 03 01 01 01 00 00 00	ÿÄ

- 不影响图片打开，达到隐写目的





插入压缩包



- 插入不同文件类型
- 这类隐写题目一般会在图片中插入zip等形式的压缩包，需要我们从图片中分离出压缩包，找到flag。
- 这种隐写手法命令行就能制作隐写图种
- 命令行： `copy /b 1.jpg+1.zip new.jpg`

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





插入压缩包



- 得到new.jpg，打开正常 但是winhex查看16进制

00016928	D8 05 5A 22 D3 0F 6C F2 F4 92 AA 88 B6 64 AA AA	0 Z"Ó l0ó' a~gd a a
00016944	22 0F FF D9 50 4B 03 04 14 00 00 08 08 00 A0 85	" yÜPK ...
00016960	8A 4C 98 5F 82 FE 10 00 00 00 0E 00 00 00 05 00	ŠL~_,p
00016976	00 00 31 2E 74 78 74 2B 4F CD 49 CE CF 4D 2D C9	1.txt+0íiîïM-É
00016992	2F 4B CD 2B 2D 06 00 50 4B 01 02 3F 00 14 00 00	/Kí+- PK ?
00017008	08 08 00 A0 85 8A 4C 98 5F 82 FE 10 00 00 00 0E	...ŠL~_,p
00017024	00 00 00 05 00 24 00 00 00 00 00 00 00 20 00 00	\$
00017040	00 00 00 00 00 31 2E 74 78 74 0A 00 20 00 00 00	1.txt
00017056	00 00 01 00 18 00 56 50 7A 35 A8 D0 D3 01 1A 1C	VPz5"ĐÓ
00017072	68 2C A8 D0 D3 01 1A 1C 68 2C A8 D0 D3 01 50 4B	h,"ĐÓ h,"ĐÓ PK
00017088	05 06 00 00 00 00 01 00 01 00 57 00 00 00 33 00	W 3
00017104	00 00 00 00	

- 在JPG图片结尾标志FFD9后面能够发现有504B0304是zip压缩包的文件头标识。

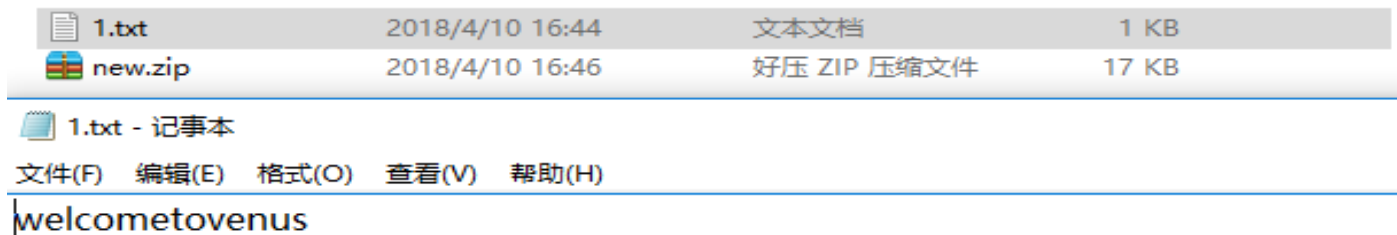




插入压缩包



- 直接将文件名后缀改成.zip解压缩就能得到压缩数据，系统会根据需求来匹配文件。



- 但是这种方法提取数据有一定的局限性，如果文件中插入的不是压缩文件，而是图片文件呢？





双图隐写



- 双图隐写的隐写方法与插入压缩包的手法类似
- 命令行: `copy /b 1.jpg+2.jpg 12.jpg`

```
C:\Users\Administrator\Desktop>copy /b 1.jpg+2.jpg 12.jpg
1.jpg
2.jpg
已复制          1 个文件。
```





双图隐写



- 如果是这类双图隐写，我们就需要使用kali中的工具来辅助解题了。
- binwalk:帮助我们识别文件结构
- foremost文件:得到隐秘数据
- 所以binwalk分离图片，发现是由两张JPEG格式图片组成

```
root@kali2017-64:~/桌面# binwalk 12.jpg
```

DECIMAL	HEXADECIMAL	DESCRIPTION
0	0x0	JPEG image data, JFIF standard 1.01
16948	0x4234	JPEG image data, JFIF standard 1.01

- foremost将图片分离，输出到new文件夹中去

```
root@kali2017-64:~/桌面# foremost 12.jpg -o new
Processing: 12.jpg
|*|
```





双图隐写



- 因为我们使用binwalk对文件进行分析一般是基于文件头标识的，所以当我们插入的双图头部表示被破坏，就只能手动去分析，抽离隐秘数据。
- 当我们的binwalk不能自动分析时

```
root@kali2017-64:~/桌面# binwalk 12.jpg
```

DECIMAL	HEXADECIMAL	DESCRIPTION
0	0x0	JPEG image data, JFIF standard 1.01





搜索文件尾



- 我们搜索文件尾看是否存在未补全的文件头

48	8D	57	12	4A	D6	71	2E	7D	13	D5	11	73	64	E9	C1	H W JÖq.) Ö sdéÄ
3D	CE	FF	00	99	5D	A0	34	50	44	4C	4C	90	E2	6E	95	=îÿ ð] 4PDLL ân•
D8	05	5A	22	D3	0F	6C	F2	F4	92	AA	88	B6	64	AA	AA	Ø Z"Ó lðð'•^gd**
22	0F	FF	D9	D8	FF	E0	00	10	4A	46	49	46	00	01	01	" yÜ@yà JFIF
01	00	60	00	60	00	00	FF	DB	00	43	00	08	06	06	07	· · yÜ C
06	05	08	07	07	07	09	09	08	0A	0C	14	0D	0C	0B	0B	
0C	19	12	13	0F	14	1D	1A	1F	1E	1D	1A	1C	1C	20	24	\$
2E	27	20	22	2C	23	1C	1C	28	37	29	2C	30	31	34	34	. ' ",# (7),0144
34	1F	27	39	3D	38	32	3C	2E	33	34	32	FF	DB	00	43	4 '9=82<.342yÜ C

- 补全，分离得到双图

00004220	D8	05	5A	22	D3	0F	6C	F2	F4	92	AA	88	B6	64	AA	AA	Ø Z"Ó lðð'•^gd**
00004230	22	0F	FF	D9	FF	D8	FF	E0	00	10	4A	46	49	46	00	01	" yÜ@yà JFIF
00004240	01	01	00	60	00	60	00	00	FF	DB	00	43	00	08	06	06	· · yÜ C
00004250	07	06	05	08	07	07	07	09	09	08	0A	0C	14	0D	0C	0B	
00004260	0B	0C	19	12	13	0F	14	1D	1A	1F	1E	1D	1A	1C	1C	20	

```
root@kali2017-64:~/桌面# binwalk 12.jpg
```

DECIMAL	HEXADECIMAL	DESCRIPTION
0	0x0	JPEG image data, JFIF standard 1.01
16948	0x4234	JPEG image data, JFIF standard 1.01





LSB隐写



- LSB也就是最低有效位 (Least Significant Bit)。
- LSB隐写原理就是图片中的像素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隐写就是修改了像素中的最低的1bit，在人眼看来是看不出来区别的，也把信息隐藏起来了。
- 如果是要寻找这种LSB隐藏痕迹的话，有一个工具是个神器——Stegsolve，可以用来辅助我们进行分析。









LSB隐写二维码



- 基于LSB隐写的手法一般由两种：
- 1.将最低有效位替换为0,1，然后组成二维码

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

红

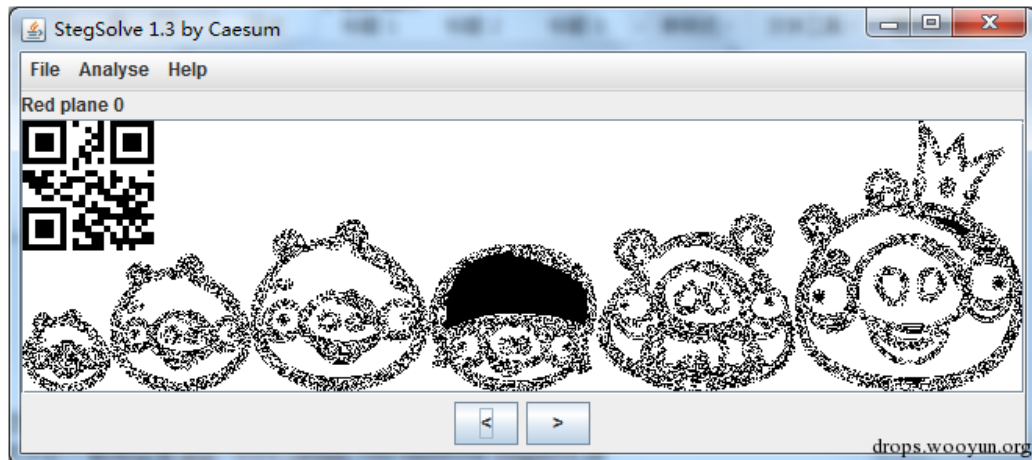




LSB隐写二维码



- 例题：ste.png
- Stegsolve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。





LSB隐写ascii码



- 2.将最低有效位替换为0和1，然后7或者8位一组组成新的ascii码

十进制

11111111	0
00000000	1
00000000	1
11111111	0
00000000	0
00000000	0
00000000	0
00000000	1

最低有效位 十六进制 ASCII码
01100001 = 0x61 = A

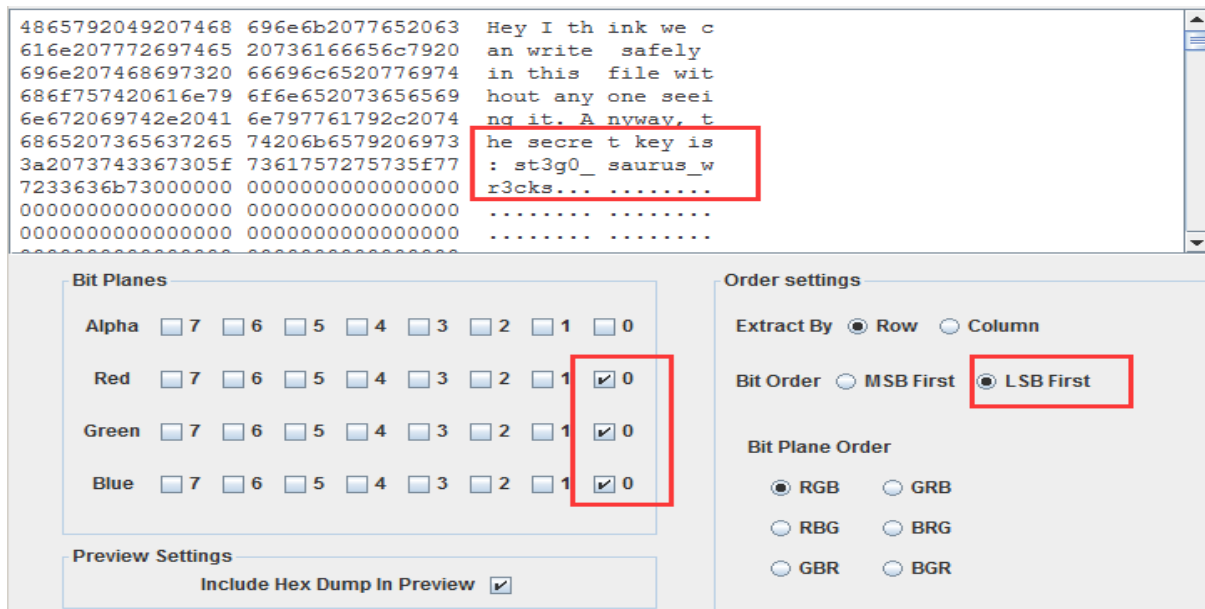




LSB隐写ascii码



- 例题：steg.png
- 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





LSB双图对比



- 双图对比
- 前面讲到的LSB都是单图隐写，但是有时候，得到的两张图看似一样。
- 例如：

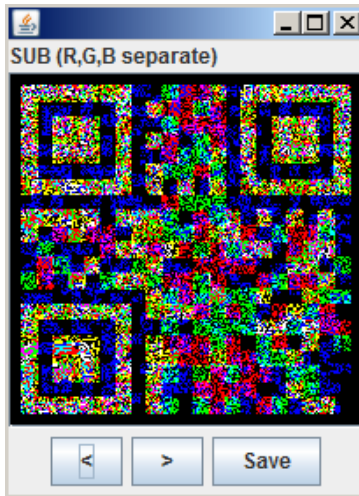
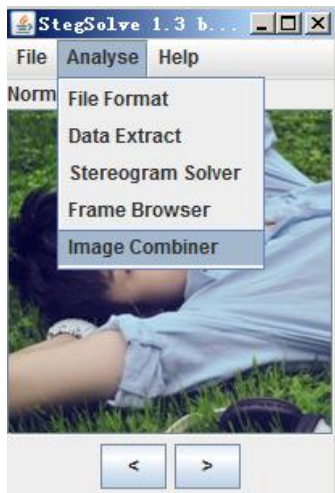




LSB双图对比



- 这种情况下，人眼是无法进行判断的，需要利用工具stegsolve，利用image combiner功能对比两张图片，翻看通道，得到一张类似二维码的图片





LSB双图对比



- 经过反色，stegsolve查看通道，得到三个二维码图



- 根据扫描二维码得到的提示解题即可

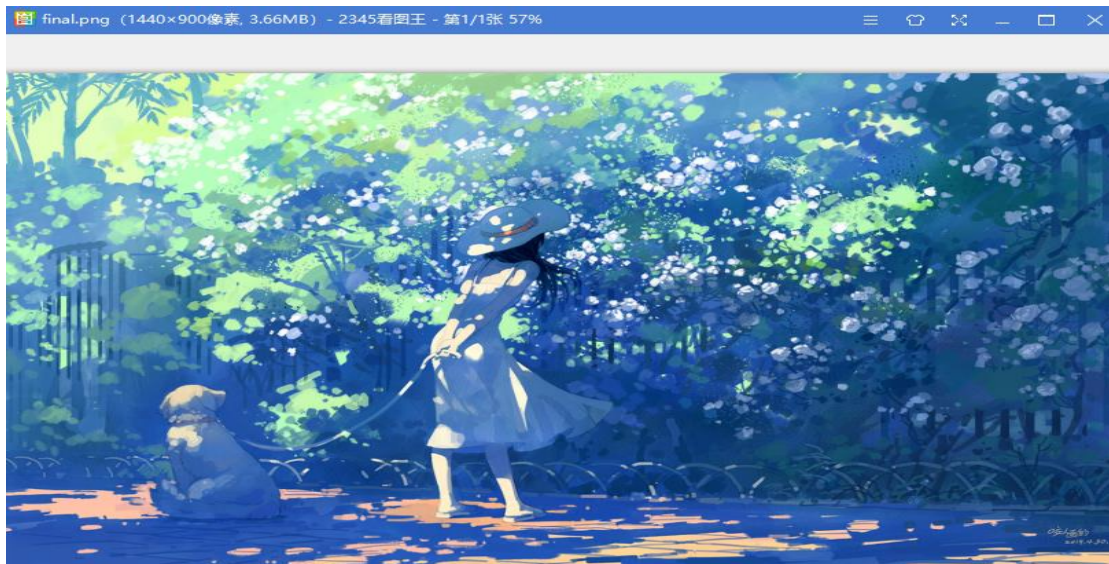




LSB双图对比



- LSB终极双图对比，提取不同点像素
 - 例题，给出一张图片如下：





LSB双图对比



- 首先binwalk分析图片结构

```
root@kali2017-64:~/桌面# binwalk final.png
```

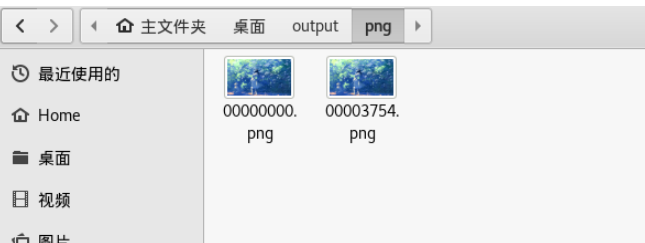
DECIMAL	HEXADECIMAL	DESCRIPTION
0	0x0	PNG image, 1440 x 900, 8-bit/color RGB, non-interlaced
41	0x29	Zlib compressed data, default compression
1922524	0x1D55DC	PNG image, 1440 x 900, 8-bit/color RGB, non-interlaced
1922565	0x1D5605	Zlib compressed data, default compression

- 可以看出图种是由两张PNG格式图片合并的，使用foremost工具分析出两张图片

```
root@kali2017-64:~/桌面# binwalk final.png
```

DECIMAL	HEXADECIMAL	DESCRIPTION
0	0x0	PNG image, 1440
41	0x29	Zlib compressed
1922524	0x1D55DC	PNG image, 1440
1922565	0x1D5605	Zlib compressed

```
root@kali2017-64:~/桌面# foremost final.png
Processing: final.png
[*]
```





LSB双图对比



- 得到两种看似相同的图片，很显然是做了LSB的替换。这种情况比较特殊，因为有的时候主办方会给出两张图片，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，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。linux比较像个文件不同的命令是diff。

```
root@kali2017-64:~/桌面/output/png# diff 00000000.png 00003754.png
二进制文件 00000000.png 和 00003754.png 不同
```

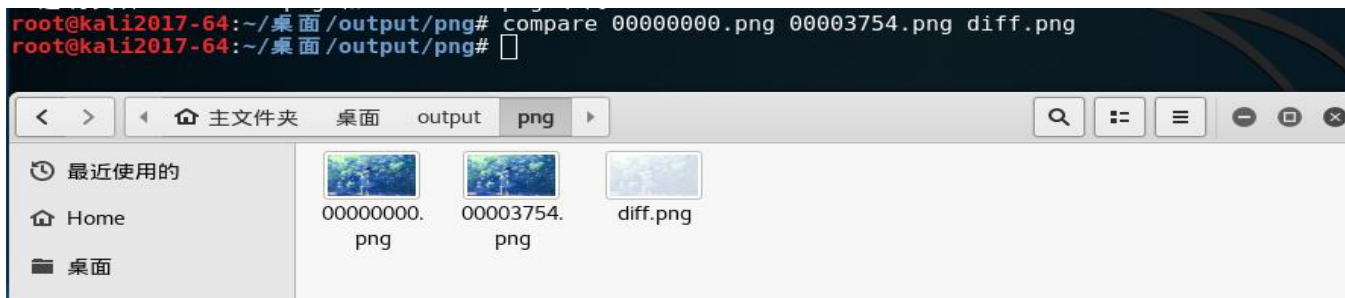




LSB双图对比



- 利用compare命令将不同的地方输出出来



- 可以清楚看到两张图片compare结果，不同的地方在于左下角位置





LSB双图对比



- 我们可以写个python脚本，将两张图片进行比较，将不同的像素位置输出，然后每8位一组组成ascii码

```
from PIL import Image
import random
img1 = Image.open('00000000.png')
im1 = img1.load()
img2 = Image.open('00003754.png')
im2 = img2.load()
a = 0
i = 0
s = ''
for x in range(img1.size[0]):
    for y in range(img1.size[1]):
        if(im1[x,y]!= im2[x,y]):
            print im1[x,y],im2[x,y]
            if i==8:
                s = s + chr(a)
                a= 0
                i= 0
            a= im2[x,y][0] + a*2
            i= i + 1
s = s + '}'
print s
```

```
(49, 90, 178) (1, 90, 178) 音乐
(49, 90, 178) (1, 90, 178)
(49, 90, 178) (1, 90, 178) 回收站
(49, 90, 178) (1, 90, 178)
(49, 90, 178) (0, 90, 178)
(49, 90, 178) (0, 90, 178) 其他位置
(49, 90, 178) (1, 90, 178)
(49, 90, 178) (0, 90, 178)
(49, 90, 178) (1, 90, 178)
(49, 90, 178) (1, 90, 178)
(49, 90, 178) (1, 90, 178)
(49, 90, 178) (1, 90, 178)
(49, 90, 178) (1, 90, 178)
(49, 90, 178) (1, 90, 178)
(49, 90, 178) (1, 90, 178)
(49, 90, 178) (0, 90, 178)
(49, 90, 178) (1, 90, 178)
ISG{E4sY StEg4n0gR4pHy} png
```

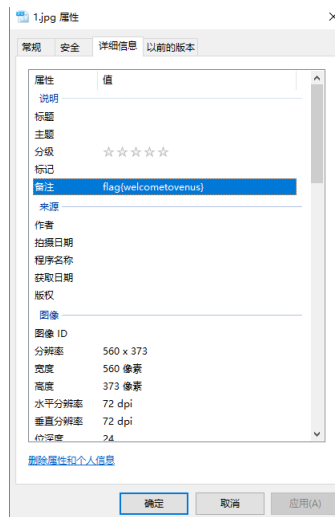
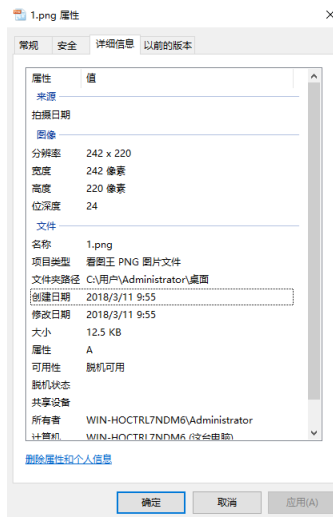




Exif隐写



- JPEG图片隐写还有一种比较常见的就是exif隐写
- JPEG图片的特点就是比其他类型图片多了exif描述，比如日期，器材，图像描述及版权等，这些位置就可以进行一些信息的隐写，比如最简单的：





Exif隐写



- 插入版权之类的可视信息，我们属性查看器就能看到，但是如果隐写在一些非可视信息中呢？
- 针对这类隐写，我们可以用前面讲到的exiftool来查看更多属性，检查是否隐写数据





Exif隐写



- 下这种图片隐写题的制作方法，依然是使用exiftool
- 我们将flag写入非可视的comment注释中

```
H:\【渗透测试工具包AI0201711】\8CTF\隐写>"exiftool (-k).exe" -copyright="ZmxhZ2lzbm90aGVyZQ==" -comment="ZmxhZ3t3ZWxjb21ldG92ZW5lc30=" 原图.jpg
1 image files updated
-- press RETURN --
```

属性	值
说明	
标题	
主题	
分级	☆☆☆☆
标记	
备注	
来源	
作者	
拍摄日期	
程序名称	
获取日期	
版权	ZmxhZ2lzbm90aGVyZQ==
图像	
图像 ID	
分辨率	560 x 373
宽度	560 像素
高度	373 像素
水平分辨率	72 dpi
垂直分辨率	72 dpi
位深度	24

```
JFIF Version : 1.01
Exif Byte Order : Big-endian (Motorola, MM)
X Resolution : 72
Y Resolution : 72
Resolution Unit : inches
Y Cb Cr Positioning : Centered
Copyright : ZmxhZ2lzbm90aGVyZQ==
Comment : ZmxhZ3t3ZWxjb21ldG92ZW5lc30=
Image Width : 560
Image Height : 373
Encoding Process : Baseline DCT, Huffman coding
Bits Per Sample : 8
Color Components : 3
Y Cb Cr Sub Sampling : YCbCr4:2:0 (2 2)
Image Size : 560x373
Megapixels : 0.209
-- press RETURN --
```





PNG隐写



- 认识PNG图片结构
- PNG图像格式文件由一个8字节的PNG文件头标志和按照特定结构组织的3个以上的数据块组成。
- 文件头：89 50 4E 47 0D 0A 1A 0A
- 四个关键数据块如下：

PNG文件格式中的数据块				
数据块符号	数据块名称	多数据块	可选否	位置限制
IHDR	文件头数据块	否	否	第一块
PLTE	调色板数据块	否	是	在IDAT之前
IDAT	图像数据块	是	否	与其他IDAT连续
IEND	图像结束数据	否	否	最后一个数据块





PNG隐写



- 每个数据块都由4个域组成：

名称	字节数	说明
Length(长度)	4字节	指定数据块中数据域的长度，其长度不超过 $(2^{31} - 1)$ 字节
Chunk Type Code(数据块类型码)	4字节	数据块类型码由ASCII字母(A-Z和a-z)组成
Chunk Data(数据块数据)	可变长度	存储按照Chunk Type Code指定的数据
CRC(循环冗余检测)	4字节	存储用来检测是否有错误的循环冗余码





PNG隐写



- PNG图片的特点
- 像苹果手机等拍出来的png格式图片一般会由IHDR块来控制图像显示的大小而不改变图片真实大小。
- 无损压缩，将图片源码通过zlib压缩编码后以IDAT块的形式进行存储数据，每个IDAT块最多存储65524字节数据。只有存满才能往下一个IDAT块存。

所以PNG图片隐写常见有IHDR块隐写和IDAT块隐写。





● IHDR

- 文件头数据块IHDR(header chunk): 它包含有PNG文件中存储的图像数据的基本信息, 并要作为第一个数据块出现在PNG数据流中, 而且一个PNG数据流中只能有一个文件头数据块。
- 文件头数据块由13字节组成, 它的格式如下表所示。

域的名称	字节数	说明
Width	4 bytes	图像宽度, 以像素为单位
Height	4 bytes	图像高度, 以像素为单位
Bit depth	1 byte	图像深度: 索引彩色图像: 1, 2, 4或8 灰度图像: 1, 2, 4, 8或16 真彩色图像: 8或16
ColorType	1 byte	颜色类型: 0: 灰度图像, 1, 2, 4, 8或16 2: 真彩色图像, 8或16 3: 索引彩色图像, 1, 2, 4或8 4: 带α通道数据的灰度图像, 8或16 6: 带α通道数据的真彩色图像, 8或16
Compression method	1 byte	压缩方法(LZ77派生算法)
Filter method	1 byte	滤波器方法
Interlace method	1 byte	隔行扫描方法: 0: 非隔行扫描 1: Adam7(由Adam M. Costello开发的7遍隔行扫描方法)





IHDR隐写



- 看似普通的1.png图片如下:

Where Is The Key???



Winhex查看16进制代码:

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ANSI ASCII
00000000	89	50	4E	47	0D	0A	1A	0A	00	00	00	0D	49	48	44	52	%PNG IHDR
00000010	00	00	02	9C	00	00	01	DD	08	06	00	00	00	FE	1A	5A	α Ý þ Z
00000020	B6	00	00	00	04	73	42	49	54	08	08	08	08	7C	08	64	¶ sBIT d
00000030	88	00	00	00	09	70	48	59	73	00	00	0B	12	00	00	0B	^ pHYs
00000040	12	01	D2	DD	7E	FC	00	00	00	16	74	45	58	74	43	72	ÒÝ~ü tEXtCr
00000050	65	61	74	69	6F	6E	20	54	69	6D	65	00	31	32	2F	31	eation Time 12/1
00000060	39	2F	31	35	6C	F1	55	23	00	00	00	1C	74	45	58	74	9/15lñU# tEXt
00000070	53	6F	66	74	77	61	72	65	00	41	64	6F	62	65	20	46	Software Adobe F

我们知道IHDR能够控制图片显示的大小, 改变长度

Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	ANSI ASCII
00000000	89	50	4E	47	0D	0A	1A	0A	00	00	00	0D	49	48	44	52	%PNG IHDR
00000016	00	00	02	9C	00	00	02	DD	08	06	00	00	00	FE	1A	5A	α Ý þ Z
00000032	B6	00	00	00	04	73	42	49	54	08	08	08	08	7C	08	64	¶ sBIT d
00000048	88	00	00	00	09	70	48	59	73	00	00	0B	12	00	00	0B	^ pHYs
00000064	12	01	D2	DD	7E	FC	00	00	00	16	74	45	58	74	43	72	ÒÝ~ü tEXtCr
00000080	65	61	74	69	6F	6E	20	54	69	6D	65	00	31	32	2F	31	eation Time 12/1
00000096	39	2F	31	35	6C	F1	55	23	00	00	00	1C	74	45	58	74	9/15lñU# tEXt





IHDR隐写



- 打开图片可以清楚的看到下方隐写的隐秘信息

Where Is The Key???



CTF{PNG_IHDR_CRC}





IDAT隐写



- PNG图片隐写—IDAT块
- 利用pngcheck工具可以分析png格式的块元素

```
H:\【渗透测试工具包AI0201711】\8CTF\隐写>pngcheck.exe 1.png
OK: 1.png (1000x562, 32-bit RGB+alpha, non-interlaced, 36.8%).

H:\【渗透测试工具包AI0201711】\8CTF\隐写>pngcheck.exe -v 1.png
File: 1.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 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
```

- 倒数第二个IDAT块没存储满，就存储下一个了，说明数据可能是后续人为添加进去的。





IDAT隐写



● 用winhex提取出数据

0015AFE0	95 00 FA 54 0D 21 BD BA 02 FF 3F 01 E7 98 5E 68	• úT !%° ŷ? ç~^h
0015AFF0	95 8F CD 00 00 00 8A 49 44 41 54 78 9C 5D 91 01	• í ŠIDATxæ]`
0015B000	12 80 40 08 02 BF 04 FF FF 5C 75 29 4B 55 37 73	€@ ħ ŷŷ\u)KU7s
0015B010	8A 21 A2 7D 1E 49 CF D1 7D B3 93 7A 92 E7 E6 03	Š!ç} IİŇ}°`z'çæ
0015B020	88 0A 6D 48 51 00 90 1F B0 41 01 53 35 0D E8 31	^ mHQ °A S5 è1
0015B030	12 EA 2D 51 C5 4C E2 E5 85 B1 5A 2F C7 8E 88 72	è-QĂLââ...±Z/ÇŽ~r
0015B040	F5 1C 6F C1 88 18 82 F9 3D 37 2D EF 78 E6 65 B0	ö oĂ^ ,ù=7-ixæe°
0015B050	C3 6C 52 96 22 A0 A4 55 88 13 88 33 A1 70 A2 07	ĂlR-" xU^ ^3;pç
0015B060	1D DC D1 82 19 DB 8C 0D 46 5D 8B 69 89 71 96 45	ÜŇ, ŪÇ FJ<i%q-E
0015B070	ED 9C 11 C3 6A E3 AB DA EF CF C0 AC F0 23 E7 7C	iox Ăjă«ŪiİĂ-ð#ç
0015B080	17 C7 89 76 67 D9 CF A5 A8 00 00 00 00 49 45 4E	Ç%vgŪİ% IEN
0015B090	44 AE 42 60 82	DøB`,

● 对提取出来的数据进行进一步处理就能得到flag





GIF隐写



- GIF格式的图片是动图，由于GIF的动态特性，由一帧帧的图片构成，所以每一帧的图片，多帧图片间的结合，都成了隐藏信息的一种载体。
- 对于此类题目，可以单独观察每一帧来判断
- 例如：1.gif





GIF隐写



- 对于gif图片隐写来说，需要防范的是gif图片的嵌套，就是多张gif图片合成，信息往往隐写在被嵌套的图片中当我们打开图片时，图片还是动图，人们思维就会被转移，需要小心。
- 可以使用binwalk 帮助我们分析文件结构，不能大意

```
root@kali2017-64:~/桌面# binwalk gif.gif
```

DECIMAL	HEXADECIMAL	DESCRIPTION
0	0x0	GIF image data, version "87a"
58000	0xE290	Zip archive data, at least v2
ssed size: 10882, uncompressed size: 15579, name: 233333.gif		
69014	0x10D96	End of Zip archive

```
root@kali2017-64:~/桌面# foremost gif.gif -o gif
Processing: gif.gif
|foundat=233333.gif0[y8000p506004DIHHR40Kk000#0,e)0}g0d70
?000000s0000|>009000000A0a
*|
```





fireworks图层隐写



- CTF中往往会把一些隐秘信息隐藏在图层中而不被发现，当我们没有解题思路时，可以用photoshop来帮助我们分析。

- 例题：大白.jpg
- 描述：大白激活口令

19,9,10,5,6,1,5,22,6,3,2,19,5,12,25,24,17,8,1,19,18,25,26,2,3,18,7,4,5,
3,11,3,13,5,1,11,18,23,26,9,5,24,5,26,2,11,17,7,2,1,9,17,8,5,7,21,15,21
,17,17,10,9,20,7,2,12

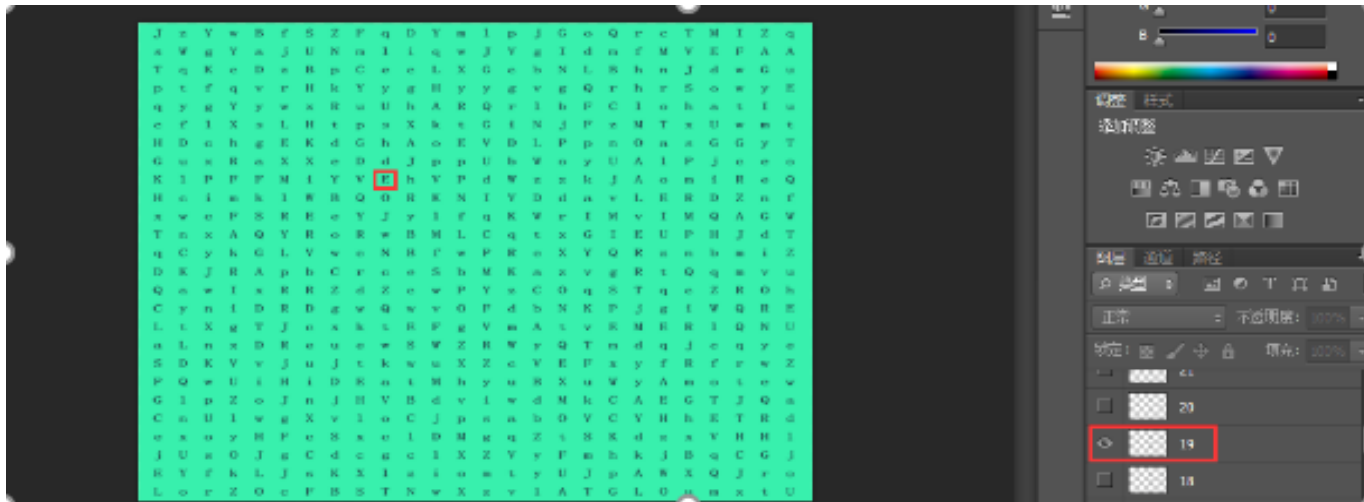




fireworks图层隐写



- 像类似这种的，仔细分析它的激活口令会发现最大的数字只有26,用PS打开，发现刚好26个图层，而且每个图层背景是字母表



- 并且我们的口令是3的倍数，我们3个一组就能形成坐标，获取字母，得打flag。





Image photography



- 简单的图形界面，输入密码字符串，或者也可以拖动一个密码文件，然后加密。将程序调为decode模式，拖动加密后的文件，点击START之后，密码字符串或者隐写文件就输出出来了。
- 在线链接：<http://www.atool.org/steganography.php>





Steghide隐写



- steghide
- 可以在图片或者音频文件中隐写信息，而且你不会注意到图片或音频文件发生了任何的改变。
- 加密：
`steghide embed -cf picture.jpg -ef secret.txt`
- 解密：
`steghide extract -sf picture.jpg`





F5隐写



- F5-steganography
- 这个工具可以将F5加密过后的文件进行解密，并且输出为output.txt文件
- 解密：
- `cd F5-steganography`
`java Extract 图片名 -p 密码`
后会生成output.txt文件，隐秘数据可能就在里面



>> Outguess隐写

- Outguess也是一种图片隐写方法，需要先编译安装
- linux下载安装：
./configure && make && make install
- 执行以下命令解密：
outguess -r 图片名 -t 生成文件名





Bftools



- Bftools工具也是命令行的一种图片隐写工具
- 常见语法，查看帮助

```
D:\【渗透测试工具包AI0201711】\8CTF\bftools>bftools.exe -h
```

```
Command name not recognized.
```

```
Available commands are:
```

```
run          - Run the given Brainfuck program.  
encode       - Encode input file using one of the languages.  
decode       - Decode input image using one of the languages.  
enlarge      - Enlarge an image by a given factor.  
reduce       - Shrink an image by a given factor.
```

```
help <name> - For help with one of the above commands
```





- 具体命令用法
- bftools.exe help decode

```
D:\【渗透测试工具包AIO201711】\8CTF\bftools>bftools.exe help decode
'decode' - Decode input image using one of the languages.

Expected usage: bftools.exe decode <options> <brainloller | braincopter> <image | ->
<options> available:
  -o, --output=VALUE      OPTIONAL. Output file. Defaults to stdout.
```





Stegdetect



- Stegdetect程序主要用于分析JPEG文件。
- 因此用Stegdetect可以检测到通过JSteg、JPHide、OutGuess、Invisible Secrets、F5、appendX和Camouflage等这些隐写工具隐藏的信息。
- 参考链接：
- <https://book.2cto.com/201407/45279.html>





Stegdetect



- Stegdetect的主要选项如下：
- q – 仅显示可能包含隐藏内容的图像
- n – 启用检查JPEG文件头功能，以降低误报率。如果启用，所有带有批注区域的文件将被视为没有被嵌入信息。如果JPEG文件的JFIF标识符中的版本号不是1.1，则禁用OutGuess检测。
- s – 修改检测算法的敏感度，该值的默认值为1。检测结果的匹配度与检测算法的敏感度成正比，算法敏感度的值越大，检测出的可疑文件包含敏感信息的可能性越大。
- d – 打印带行号的调试信息。
- t – 设置要检测哪些隐写工具（默认检测jopi），可设置的选项如下：
 - j – 检测图像中的信息是否是用jsteg嵌入的。
 - o – 检测图像中的信息是否是用outguess嵌入的。
 - p – 检测图像中的信息是否是用jphide嵌入的。
 - i – 检测图像中的信息是否是用invisible secrets嵌入的。





Stegdetect



- Stegdetect通过修改敏感度进行检测是否隐写

```
I:\CTF类\CTF-新-未整理\隐写\stegdetect-0.4\stegdetect>stegdetect.exe -tjopi Pcat. jpg
Pcat. jpg : negative

I:\CTF类\CTF-新-未整理\隐写\stegdetect-0.4\stegdetect>stegdetect.exe -tjopi -s100 Pc
at. jpg
Pcat. jpg : jphide(***)

I:\CTF类\CTF-新-未整理\隐写\stegdetect-0.4\stegdetect>_
```



感谢观看



启明星辰
网络空间安全学院