图片分析

图片分析简介

图像文件能够很好地包含黑客文化，因此 CTF 竞赛中经常会出现各种图像文件。

图像文件有多种复杂的格式，可以用于各种涉及到元数据、信息丢失和无损压缩、校验、隐写或可视化数据编码的分析解密，都是 Misc 中的一个很重要的出题方向。涉及到的知识点很多（包括基本的文件格式，常见的隐写手法及隐写用的软件），有的地方也需要去进行深入的理解。

元数据（Metadata）

> 元数据（Metadata），又称中介数据、中继数据，为描述数据的数据（Data about data），主要是描述数据属性（property）的信息，用来支持如指示存储位置、历史数据、资源查找、文件记录等功能。

元数据中隐藏信息在比赛中是最基本的一种手法，通常用来隐藏一些关键的`Hint`信息或者是一些重要的如`password`等信息。

这类元数据你可以右键 --> 属性去查看, 也可以通过`strings`命令去查看，一般来说，一些隐藏的信息（奇怪的字符串）常常出现在头部或者尾部。

接下来介绍一个`identify`命令，这个命令是用来获取一个或多个图像文件的格式和特性。

-format用来指定显示的信息，灵活使用它的-format参数可以给解题带来不少方便。format 各个参数具体意义

> 题目 攻防世界 MISC高手进阶区-Mysterious GIF

这题的一个难点是发现并提取 GIF 中的元数据，首先strings是可以观察到异常点的。

Shell

```
GIF89a
   !!!"""###$$$%%%&&&'''((()))******+,,,--
-.../// 000111222333444555666777888999:::;;;<<<===>>>???
@@@AAABBBCCCDDDEEEFFFGGGHHHIIIJJJKKKLLLMMMNNNOOOPPPQQQRRRSSSTTTUUUVVVWWWXXXYYYZZ
Z[[[\\\]]]^^^___```aaabbbcccdddeeefffgggghhhiiijjjkkklllmmmnnnoooppppqqqrrrssstttu
uuvvvwwwxxxyyyzzz{{{|||}}}~~~

4d4949457677494241444 14e42676b71686b6947397730424151454641415343424b6b776767536c
41674541416f4942415144644d4e624c3571565769435172
NETSCAPE2.0
ImageMagick
...
```

这里的一串 16 进制其实是藏在 GIF 的元数据区

接下来就是提取，你可以选择 Python，但是利用identify显得更加便捷

Shell

```
$ identify -format "%s %c \n" Question.gif
0
4d4949457677494241444 14e42676b71686b6947397730424151454641415343424b6b776767536c
41674541416f4942415144644d4e624c3571565769435172
1
5832773639712f377933536849507565707478664177525162524f72653330633655772f6f4b3877
655a547834346d30414c6f75685634364b63514a6b687271
...
24
484b7735432b667741586c4649746d30396145565458772b787a4c4a623253723667415450574d35
715661756278667362356d58482f77443969434c684a536f
25 724b3052485a6b7450624573357974447371424864355046467 73d3d
```

其他过程这里不在叙述，可参考链接中的 Writeup

像素值转化

看看这个文件里的数据，你能想到什么?

Shell

```
255,255,255,255,255..........
```

是一串 RGB 值，尝试着将他转化为图片

Apache

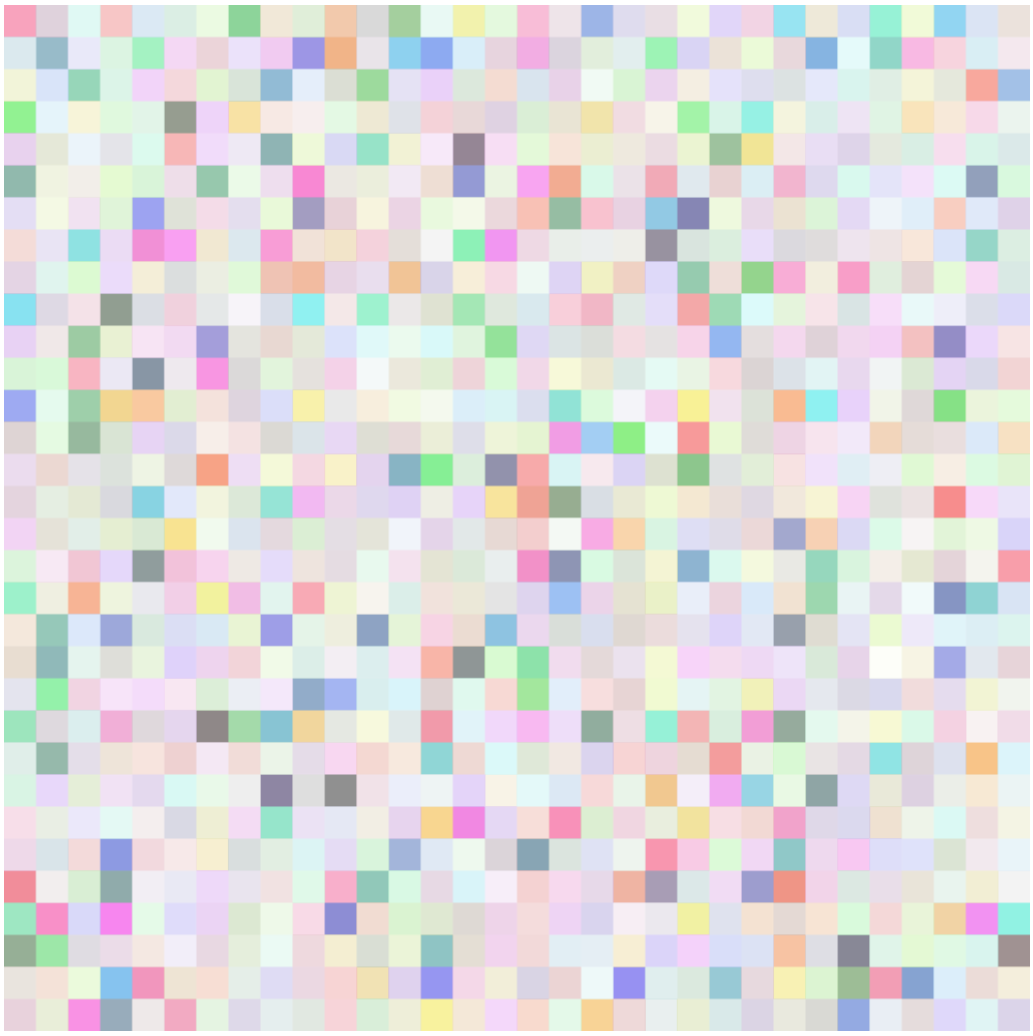
```
from PIL import Image
import re
x = 307 #x坐标  通过对txt里的行数进行整数分解
y = 311 #y坐标  x*y = 行数
rgb1 = [****]
print len(rgb1)/3
m=0
for i in xrange(0,x):
    for j in xrange(0,y):

        line = rgb1[(3*m):(3*(m+1))]#获取一行
        m+=1
        rgb = line

        im.putpixel((i,j),(int(rgb[0]),int(rgb[1]),int(rgb[2])))#rgb转化为像素
im.show()
im.save("flag.png")
```

而如果反过来的话，从一张图片提取 RGB 值，再对 RGB 值去进行一些对比，从而得到最终的 flag。

这类题目大部分都是一些像素块组成的图片，如下图



> 题目 CSAW-2016-quals:Forensic/Barinfun
>
> 题目 breakin-ctf-2017:A-dance-partner

PNG

文件格式

对于一个 PNG 文件来说，其文件头总是由位固定的字节来描述的，剩余的部分由 3 个以上的 PNG 的数据块（Chunk）按照特定的顺序组成。

文件头89 50 4E 47 0D 0A 1A 0A+ 数据块 + 数据块 + 数据块……

数据块 CHUNK

PNG 定义了两种类型的数据块，一种是称为关键数据块（critical chunk），这是标准的数据块，另一种叫做辅助数据块（ancillary chunks），这是可选的数据块。关键数据块定义了 4 个标准数据块，每个 PNG 文件都必须包含它们，PNG 读写软件也都必须要支持这些数据块。

	A	B	C	D	E
1	数据块符号	数据块名称	多数据块	可选否	位置限制
2	IHDR	文件头数据块	否	否	第一块
3	cHRM	基色和白色点数据	否	是	在 PLTE 和 IDAT 之
4	gAMA	图像γ数据块	否	是	在 PLTE 和 IDAT 之
5	sBIT	样本有效位数据块	否	是	在 PLTE 和 IDAT 之
6	PLTE	调色板数据块	否	是	在 IDAT 之前
7	bKGD	背景颜色数据块	否	是	在 PLTE 之后 IDAT
8	hIST	图像直方图数据块	否	是	在 PLTE 之后 IDAT
9	tRNS	图像透明数据块	否	是	在 PLTE 之后 IDAT
10	oFFs	（专用公共数据块	否	是	在 IDAT 之前
11	pHYs	物理像素尺寸数据	否	是	在 IDAT 之前
12	sCAL	（专用公共数据块	否	是	在 IDAT 之前
13	IDAT	图像数据块	是	否	与其他 IDAT 连续
14	tIME	图像最后修改时间	否	是	无限制
15	tEXt	文本信息数据块	是	是	无限制
16	zTXt	压缩文本数据块	是	是	无限制
17	fRAc	（专用公共数据块	是	是	无限制
18	gIFg	（专用公共数据块	是	是	无限制
19	gIFt	（专用公共数据块	是	是	无限制
20	gIFx	（专用公共数据块	是	是	无限制
21	IEND	图像结束数据	否	否	最后一个数据块

对于每个数据块都有着统一的数据结构，每个数据块由 4 个部分组成

	A	B	C
1	名称	字节数	说明
2	Length（长度）	4 字节	指定数据块中数据
3	Chunk Type Cod	4 字节	数据块类型码由 A
4	Chunk Data（数据	可变长度	存储按照 Chunk T
5	CRC（循环冗余检	4 字节	存储用来检测是否

CRC（Cyclic Redundancy Check）域中的值是对 Chunk Type Code 域和 Chunk Data 域中的数据进行计算得到的。

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，由 13 字节组成，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

其中我们关注的是前 8 字节的内容

	A	B	C
1	域的名称	字节数	说明
2	Width	4 bytes	图像宽度，以像素
3	Height	4 bytes	图像高度，以像素

我们经常会去更改一张图片的高度或者宽度使得一张图片显示不完整从而达到隐藏信息的目的。

这里可以发现在 Kali 中是打不开这张图片的，提示IHDR CRC error，而 Windows 10 自带的图片查看器能够打开，就提醒了我们 IHDR 块被人为的篡改过了，从而尝试修改图片的高度或者宽度发现隐藏的字符串。

例题

WDCTF-FINALS-2017

观察文件可以发现, 文件头及宽度异常

Ruby

```
00000000  80 59 4e 47 0d 0a 1a 0a  00 00 00 0d 49 48 44 52  |.YNG........IHDR|
00000010  00 00 00 00 00 00 02 f8  08 06 00 00 00 93 2f 8a  |................/.|
00000020  6b 00 00 00 04 67 41 4d  41 00 00 9c 40 20 0d e4  |k....gAMA...@ ..|
00000030  cb 00 00 00 20 63 48 52  4d 00 00 87 0f 00 00 8c  |.... cHRM.......|
00000040  0f 00 00 fd 52 00 00 81  40 00 00 7d 79 00 00 e9  |....R...@..}y...|
...
```

这里需要注意的是，文件宽度不能任意修改，需要根据 IHDR 块的 CRC 值爆破得到宽度, 否则图片显示错误不能得到 flag。

Go

```
import os import binascii import struct   misc = open("misc4.png","rb").read()
for i in range(1024):     data = misc[12:16] + struct.pack('>i',i)+ misc[20:29]
crc32 = binascii.crc32(data) & 0xffffffff     if crc32 == 0x932f8a6b:
print i
```

得到宽度值为 709 后，恢复图片得到 flag。

flag is wdflag{Png_

C2c_u_kn0W}

PLTE

调色板数据块 PLTE（palette chunk）：它包含有与索引彩色图像（indexed-color image）相关的彩色变换数据，它仅与索引彩色图像有关，而且要放在图像数据块（image data chunk）之前。真彩色的 PNG 数据流也可以有调色板数据块，目的是便于非真彩色显示程序用它来量化图像数据，从而显示该图像。

IDAT

图像数据块 IDAT（image data chunk）：它存储实际的数据，在数据流中可包含多个连续顺序的图像数据块。

- 储存图像像数数据
- 在数据流中可包含多个连续顺序的图像数据块
- 采用 LZ77 算法的派生算法进行压缩
- 可以用 zlib 解压缩

值得注意的是，IDAT 块只有当上一个块充满时，才会继续一个新的块。

用pngcheck去查看此 PNG 文件

JavaScript

```
λ .\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 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).
```

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的块里.

利用python zlib解压多余 IDAT 块的内容，此时注意剔除**长度、数据块类型及末尾的 CRC 校验值**。

Python

```
import zlib import binascii IDAT = "789...667".decode('hex') result = binascii.hexlify(zlib.decompress(IDAT)) print result
```

IEND

图像结束数据 IEND（image trailer chunk）：它用来标记 PNG 文件或者数据流已经结束，并且必须要放在文件的尾部。

```
Apache
1  00 00 00 00 49 45 4E 44 AE 42 60 82
```

IEND 数据块的长度总是00 00 00 00，数据标识总是 IEND49 45 4E 44，因此，CRC 码也总是AE 42 60 82。

其余辅助数据块

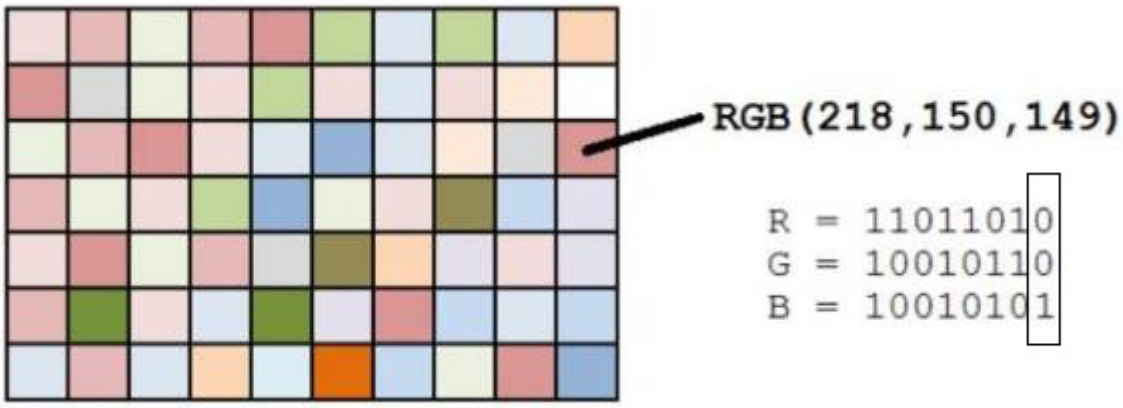
- 背景颜色数据块 bKGD（background color）
- 基色和白色度数据块 cHRM（primary chromaticities and white point），所谓白色度是指当R＝G＝B＝最大值时在显示器上产生的白色度
- 图像 γ 数据块 gAMA（image gamma）
- 图像直方图数据块 hIST（image histogram）
- 物理像素尺寸数据块 pHYs（physical pixel dimensions）
- 样本有效位数据块 sBIT（significant bits）
- 文本信息数据块 tEXt（textual data）
- 图像最后修改时间数据块 tIME （image last-modification time）
- 图像透明数据块 tRNS （transparency）
- 压缩文本数据块 zTXt （compressed textual data）

LSB

LSB 全称 Least Significant Bit，最低有效位。PNG 文件中的图像像数一般是由 RGB 三原色（红绿蓝）组成，每一种颜色占用 8 位，取值范围为0x00至0xFF，即有 256 种颜色，一共包含了 256 的 3 次方的颜色，即 16777216 种颜色。

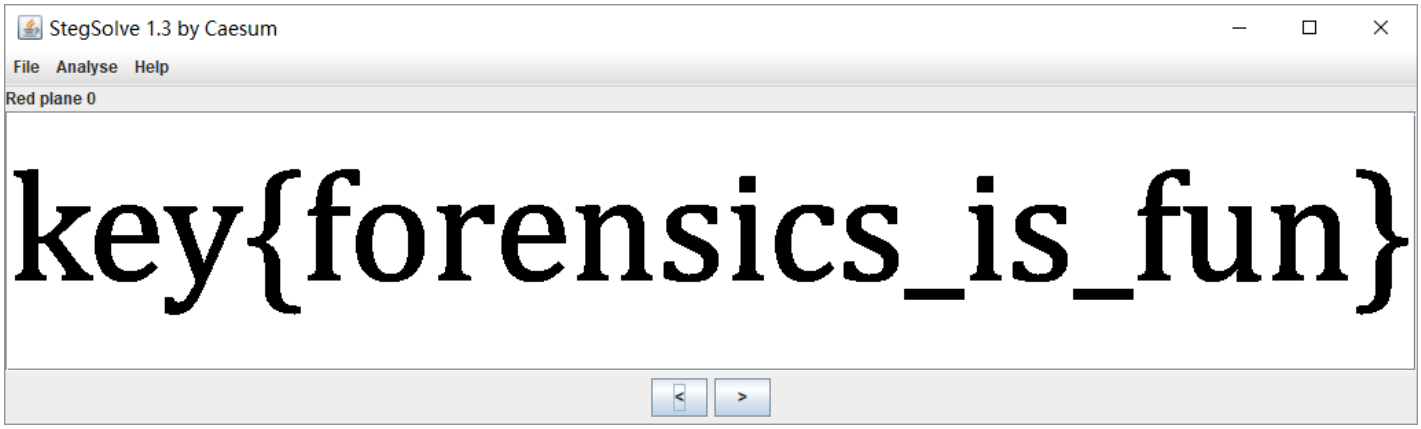
而人类的眼睛可以区分约 1000 万种不同的颜色，意味着人类的眼睛无法区分余下的颜色大约有 6777216 种。

LSB 隐写就是修改 RGB 颜色分量的最低二进制位（LSB），每个颜色会有 8 bit，LSB 隐写就是修改了像数中的最低的 1 bit，而人类的眼睛不会注意到这前后的变化，每个像素可以携带 3 比特的信息。



如果是要寻找这种 LSB 隐藏痕迹的话，有一个工具Stegsolve是个神器，可以来辅助我们进行分析。

通过下方的按钮可以观察每个通道的信息，例如查看 R 通道的最低位第 8 位平面的信息。

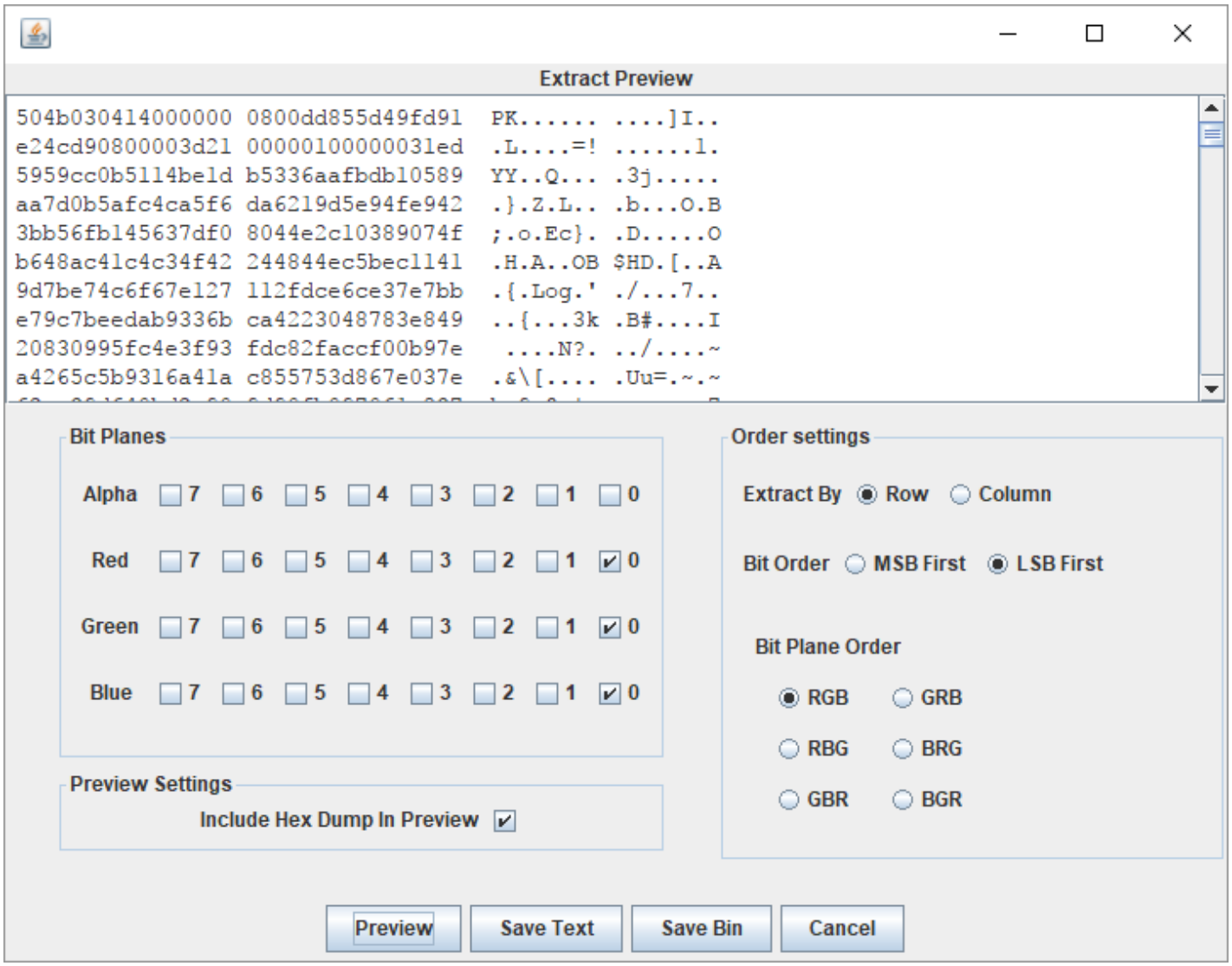


LSB 的信息借助于 Stegsolve 查看各个通道时一定要细心捕捉异常点，抓住 LSB 隐写的蛛丝马迹。

例题

> HCTF - 2016 - Misc

这题的信息隐藏在 RGB 三个通道的最低位中，借助Stegsolve-->Analyse-->Data Extract可以指定通道进行提取。



可以发现zip头，用save bin保存为压缩包后，打开运行其中的 ELF 文件就可以得到最后的 flag。

> 更多关于 LSB 的研究可以看这里。

隐写软件

Stepic

JPG

文件结构

- JPEG 是有损压缩格式，将像素信息用 JPEG 保存成文件再读取出来，其中某些像素值会有少许变化。在保存时有个质量参数可在 0 至 100 之间选择，参数越大图片就越保真，但图片的体积也就越大。一般情况下选择 70 或 80 就足够了
- JPEG 没有透明度信息

JPG 基本数据结构为两大类型：「段」和经过压缩编码的图像数据。

	A	B	C	D
1	名 称	字节数	数据	说明
2	段 标识	1	FF	每个新段的开始标
3	段类型	1		类型编码（称作标
4	段长 度	2		包括段内容和段长
5	段内容	2		≤65533 字节

- 有些段没有长度描述也没有内容，只有段标识和段类型。文件头和文件尾均属于这种段。
- 段与段之间无论有多少FF都是合法的，这些FF称为「填充字节」，必须被忽略掉。

一些常见的段类型

Short name	Bytes	Payload	Name	Comments
SOI	0xFF, 0xD8	*none*	Start Of Image	
SOF0	0xFF, 0xC0	*variable size*	Start Of Frame (baseline DCT)	Indicates that this is a baseline DCT-based JPEG, and specifies the width, height, number of components, and component subsampling (e.g., 4:2:0).
SOF2	0xFF, 0xC2	*variable size*	Start Of Frame (progressive DCT)	Indicates that this is a progressive DCT-based JPEG, and specifies the width, height, number of components, and component subsampling (e.g., 4:2:0).
DHT	0xFF, 0xC4	*variable size*	Define Huffman Table(s)	Specifies one or more Huffman tables.
DQT	0xFF, 0xDB	*variable size*	Define Quantization Table(s)	Specifies one or more quantization tables.
DRI	0xFF, 0xDD	4 bytes	Define Restart Interval	Specifies the interval between RST*n* markers, in Minimum Coded Units (MCUs). This marker is followed by two bytes indicating the fixed size so it can be treated like any other variable size segment.
SOS	0xFF, 0xDA	*variable size*	Start Of Scan	Begins a top-to-bottom scan of the image. In baseline DCT JPEG images, there is generally a single scan. Progressive DCT JPEG images usually contain multiple scans. This marker specifies which slice of data it will contain, and is immediately followed by entropy-coded data.
RST*n*	0xFF, 0xD*n* (*n*=0..7)	*none*	Restart	Inserted every *r* macroblocks, where *r* is the restart interval set by a DRI marker. Not used if there was no DRI marker. The low three bits of the marker code cycle in value from 0 to 7.
APP*n*	0xFF, 0xE*n*	*variable size*	Application-specific	For example, an Exif JPEG file uses an APP1 marker to store metadata, laid out in a structure based closely on TIFF.
COM	0xFF, 0xFE	*variable size*	Comment	Contains a text comment.
EOI	0xFF, 0xD9	*none*	End Of Image	

0xffd8和0xffd9为 JPG 文件的开始结束的标志。

隐写软件

Stegdetect

通过统计分析技术评估 JPEG 文件的 DCT 频率系数的隐写工具, 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，并且还具有基于字典暴力破解密码方法提取通过 Jphide、outguess 和 jsteg-shell 方式嵌入的隐藏信息。

Python

```
-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。
```

JPHS

JPEG 图像的信息隐藏软件 JPHS，它是由 Allan Latham 开发设计实现在 Windows 和 Linux 系统平台针对有损压缩 JPEG 文件进行信息加密隐藏和探测提取的工具。软件里面主要包含了两个程序 JPHIDE 和 JPSEEK。JPHIDE 程序主要是实现将信息文件加密隐藏到 JPEG 图像功能，而 JPSEEK 程序主要实现从用 JPHIDE 程序加密隐藏得到的 JPEG 图像探测提取信息文件，Windows 版本的 JPHS 里的 JPHSWIN 程序具有图形化操作界面且具备 JPHIDE 和 JPSEEK 的功能。

SilentEye

> SilentEye is a cross-platform application design for an easy use of steganography, in this case hiding messages into pictures or sounds. It provides a pretty nice interface and an easy integration of new steganography algorithm and cryptography process by using a plug-ins system.

GIF

文件结构

一个 GIF 文件的结构可分为

- 文件头（File Header）
  - GIF 文件署名（Signature）
  - 版本号（Version）
- GIF 数据流（GIF Data Stream）
  - 控制标识符
  - 图象块（Image Block）
  - 其他的一些扩展块
- 文件终结器（Trailer）

下表显示了一个 GIF 文件的组成结构：

中间的那个大块可以被重复任意次

文件头

GIF 署名（Signature）和版本号（Version）。GIF 署名用来确认一个文件是否是 GIF 格式的文件，这一部分由三个字符组成：GIF；文件版本号也是由三个字节组成，可以为87a或89a。

逻辑屏幕标识符（Logical Screen Descriptor）

Logical Screen Descriptor（逻辑屏幕描述符）紧跟在 header 后面。这个块告诉 decoder（解码器）图片需要占用的空间。它的大小固定为 7 个字节，以 canvas width（画布宽度）和 canvas height（画布高度）开始。

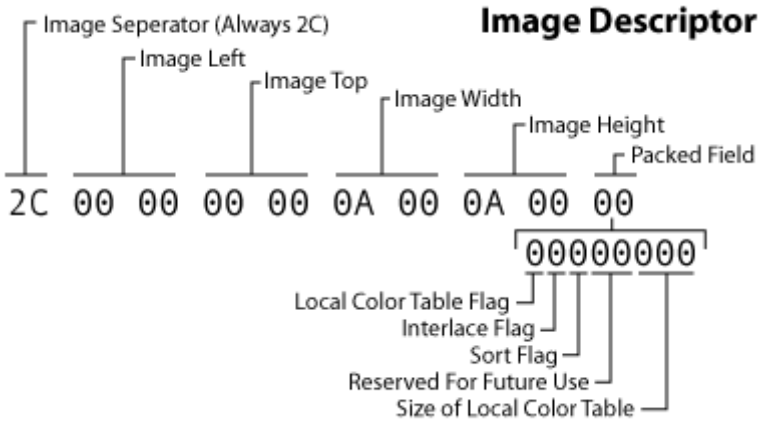
全局颜色列表（Global Color Table）

GIF 格式可以拥有 global color table，或用于针对每个子图片集，提供 local color table。每个 color table 由一个 RGB（就像通常我们见到的（255，0，0）红色 那种）列表组成。

图像标识符（Image Descriptor）

一个 GIF 文件一般包含多个图片。之前的图片渲染模式一般是将多个图片绘制到一个大的（virtual canvas）虚拟画布上，而现在一般将这些图片集用于实现动画。

每个 image 都以一个 image descriptor block（图像描述块）作为开头，这个块固定为 10 字节。



图像数据（Image Data）

终于到了图片数据实际存储的地方。Image Data 是由一系列的输出编码（output codes）构成，它们告诉 decoder（解码器）需要绘制在画布上的每个颜色信息。这些编码以字节码的形式组织在这个

块中。

文件终结器（Trailer）

该块为一个单字段块，用来指示该数据流的结束。取固定值 0x3b.

更多参见gif 格式图片详细解析

空间轴

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。

对于需要分离的 GIF 文件, 可以使用convert命令将其每一帧分割开来

` sourceCode shell root in ~/Desktop/tmp λ convert cake.gif cake.png root in ~/Desktop/tmp λ ls cake-0.png cake-1.png cake-2.png cake-3.png cake.gif

Shell

```
### 例题  > WDCTF-2017:3-2  打开gif后，思路很清晰，分离每一帧图片后，将起合并得到完整的二维码即可  ``` sourceCode python from  PIL import Image   flag = Image.new("RGB",(450,450))  for i in range(2):     for j in range(2):  pot = "cake-{}.png".format(j+i*2)         potImage = Image.open(pot)  flag.paste(potImage,(j*225,i*225)) flag.save('./flag.png')
```

扫码后得到一串 16 进制字符串

03f30d0ab8c1aa5....74080006030908

开头03f3为pyc文件的头，恢复为python脚本后直接运行得到 flag

时间轴

GIF 文件每一帧间的时间间隔也可以作为信息隐藏的载体。

例如在当时在 XMan 选拔赛出的一题

> XMAN-2017:100.gif

通过identify命令清晰的打印出每一帧的时间间隔

Ruby

```
$ identify -format "%s %T \n" 100.gif 0 66 1 66 2 20 3 10 4 20 5 10 6 10 7 20 8 20 9 20 10 20 11 10 12 20 13 20 14 10 15 10
```

推断20 & 10分别代表0 & 1，提取每一帧间隔并进行转化。

Shell

```
$ cat flag|cut -d ' ' -f 2|tr -d '66'|tr -d '\n'|tr -d '0'|tr '2' '0'
0101100001001101010000010100111001111011001110010011011000110101001101111001101010110001001100101011001010110010000011010001100100011001010110000100110001001110000110010001100101011001000011010000011011100110011001101010011011000110100001100110110000101100101011000110110011001100001001100110011010101111101#
```

最后转 ASCII 码得到 flag。

隐写软件

- F5-steganography