PNG

EN | ZH

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。

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
cHRM	基色和白色点数据块	否	是	在 PLTE 和 IDAT 之前
gAMA	图像γ数据块	否	是	在 PLTE 和 IDAT 之前
sBIT	样本有效位数据块	否	是	在 PLTE 和 IDAT 之前
PLTE	调色板数据块	否	是	在 IDAT 之前
bKGD	背景颜色数据块	否	是	在 PLTE 之后 IDAT 之前
hIST	图像直方图数据块	否	是	在 PLTE 之后 IDAT 之前
tRNS	图像透明数据块	否	是	在 PLTE 之后 IDAT 之前

数据块符号	数据块名称	多数据块	可选否	位置限制
oFFs	（专用公共数据块）	否	是	在 IDAT 之前
pHYs	物理像素尺寸数据块	否	是	在 IDAT 之前
sCAL	（专用公共数据块）	否	是	在 IDAT 之前
IDAT	图像数据块	是	否	与其他 IDAT 连续
tIME	图像最后修改时间数据块	否	是	无限制
tEXt	文本信息数据块	是	是	无限制
zTXt	压缩文本数据块	是	是	无限制
fRAc	（专用公共数据块）	是	是	无限制
gIFg	（专用公共数据块）	是	是	无限制
gIFt	（专用公共数据块）	是	是	无限制
gIFx	（专用公共数据块）	是	是	无限制
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

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（231 - 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

名称	字节数	说明
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

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

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

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

```
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。

```
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。

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

```
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。

```
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 IEND `49 45 4E 44`，因此，CRC 码也总是 `AE 42 60 82`。

其余辅助数据块

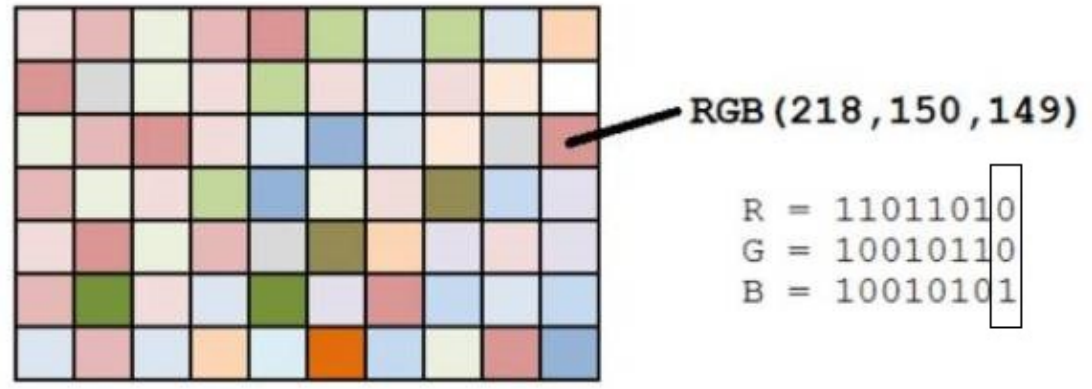
- 背景颜色数据块 bKGD（background color）
- 基色和白色度数据块 cHRM（primary chromaticities and white point），所谓白色度是指当 `R=G=B=最大值` 时在显示器上产生的白色度
- 图像 γ 数据块 gAMA（image gamma）
- 图像直方图数据块 hIST（image histogram）
- 物理像素尺寸数据块 pHYs（physical pixel dimensions）
- 样本有效位数据块 sBIT（significant bits）
- 文本信息数据块 tEXt（textual data）
- 图像最后修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

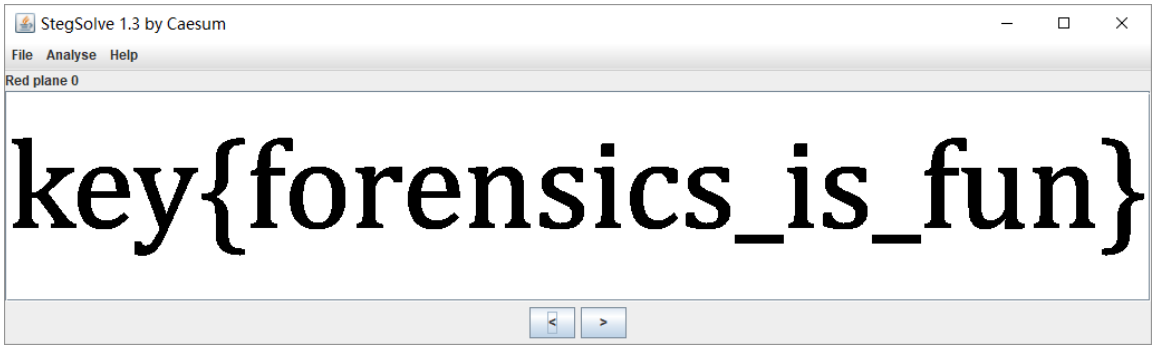
而人类的眼睛可以区分约 1000 万种不同的颜色，意味着人类的眼睛无法区分余下的颜色大约有 6777216 种。

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。

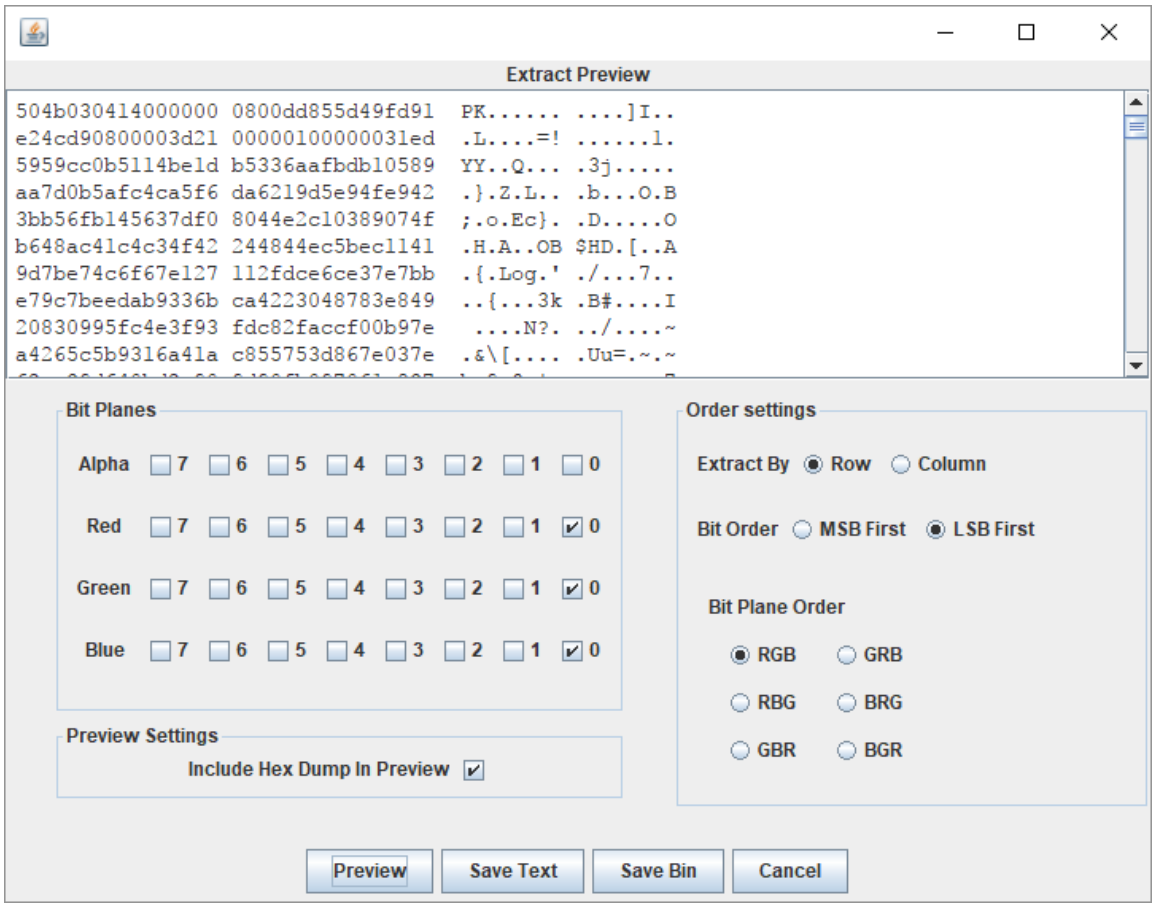


LSB 的信息借助于 Stegsolve 查看各个通道时一定要细心捕捉异常点，抓住 LSB 隐写的蛛丝马迹。

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

评论