MISC总结——隐写术(二）

转载

weixin_34269583 于 2018-07-30 06:39:00 发布 804 收藏 8
文章标签： 网络 python
原文链接：https://yq.aliyun.com/articles/649049

这一篇是继上一篇之后的另一篇关于隐写术在ctf比赛中常见的套路问题：

上一篇详情请见：https://www.cnblogs.com/lxz-1263030049/p/9388511.html

本文参考自：先知社区：https://xz.aliyun.com/t/1836

隐写术介绍：

隐写术是关于信息隐藏，即不让计划的接收者之外的任何人知道信息的传递事件（而不只是信息的内容）的一门技巧与科学。

英文写作Steganography，而这篇内容将带大家了解一下CTF赛场上常见的图片隐写方式，以及解决方法。有必要强调的是，隐写术与密码编码是完全不同的概念。

第一部分：基于文件结构的图片隐写

背景知识·：

首先这里需要明确一下我这里所说的文件结构是什么意思。文件结构特指的是图片文件的文件结构。

我们这里主要讲的是PNG图片的文件结构。

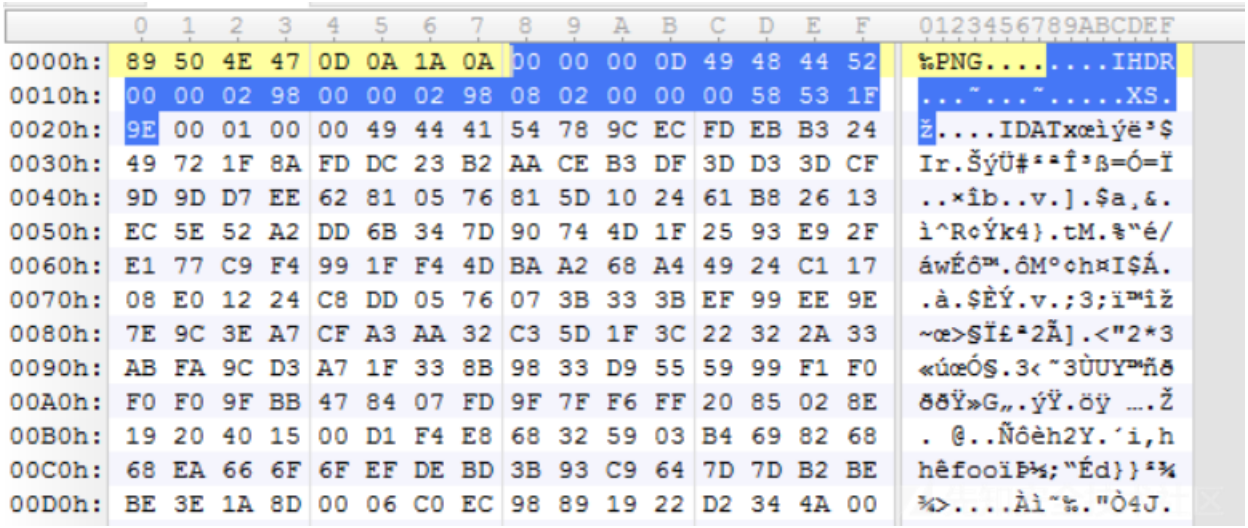
PNG，图像文件存储格式，其设计目的是试图替代GIF和TIFF文件格式，同时增加一些GIF文件格式所不具备的特性。

是一种位图文件(bitmap file)存储格式，读作“ping”。PNG用来存储灰度图像时，灰度图像的深度可多到16位，存储彩色图像时

，彩色图像的深度可多到48位，并且还可存储多到16位的α通道数据。

对于一个正常的PNG图片来讲，其文件头总是由固定的字节来表示的，以16进制表示即位 89 50 4E 47 0D 0A 1A 0A，这一部分称作文件头。
标准的PNG文件结构应包括：

- - PNG文件标志
  - PNG数据块
    PNG图片是有两种数据块的，一个是叫关键数据块，另一种是辅助数据块。正常的关键数据块，定义了4种标准数据块，个PNG文件都必须包含它们。
    它们分别是长度，数据块类型码，数据块数据，循环冗余检测即CRC。
    我们这里重点先了解一下，png图片文件头数据块以及png图片IDAT块，这次的隐写也是以这两个地方位基础的。
    png图片文件头数据块
    即IHDR，这是PNG图片的第一个数据块，一张PNG图片仅有一个IHDR数据块，它包含了哪些信息呢？IHDR中，包括了图片的宽，高，图像深度，颜色类型，压缩方法等等。



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000h:	89	50	4E	47	0D	0A	1A	0A	00	00	00	0D	49	48	44	52	%PNG....IHDR
0010h:	00	00	02	98	00	00	02	98	08	02	00	00	00	58	53	1F	...˜...˜.....XS.
0020h:	9E	00	01	00	00	49	44	41	54	78	9C	EC	FD	EB	B3	24	ž....IDATxœìýë³$
0030h:	49	72	1F	8A	FD	DC	23	B2	AA	CE	B3	DF	3D	D3	3D	CF	Ir.ŠýÜ#²ªΣ³ß=Ó=Ï
0040h:	9D	9D	D7	EE	62	81	05	76	81	5D	10	24	61	B8	26	13	..×îb..v.].$a¸&.
0050h:	EC	5E	52	A2	DD	6B	34	7D	90	74	4D	1F	25	93	E9	2F	ì^R¢Ýk4}.tM.%"é/
0060h:	E1	77	C9	F4	99	1F	F4	4D	BA	A2	68	A4	49	24	C1	17	áwÉô™.ôMº¢h¤I$Á.
0070h:	08	E0	12	24	C8	DD	05	76	07	3B	33	3B	EF	99	EE	9E	.à.$ÈÝ.v.;3;ï™îž
0080h:	7E	9C	3E	A7	CF	A3	AA	32	C3	5D	1F	3C	22	32	2A	33	~œ>§Ï£ª2Ã].<"2*3
0090h:	AB	FA	9C	D3	A7	1F	33	8B	98	33	D9	55	59	99	F1	F0	«úœÓ§.3‹˜3ÙUY™ñð
00A0h:	F0	F0	9F	BB	47	84	07	FD	9F	7F	F6	FF	20	85	02	8E	ððŸ»G„.ýŸ.öÿ ….Ž
00B0h:	19	20	40	15	00	D1	F4	E8	68	32	59	03	B4	69	82	68	. @..Ñôèh2Y.´i‚h
00C0h:	68	EA	66	6F	6F	EF	DE	BD	3B	93	C9	64	7D	7D	B2	BE	hêfooïÞ½;"Éd}}²¾
00D0h:	BE	3E	1A	8D	00	06	C0	EC	98	89	19	22	D2	34	4A	00	¾>....Àì˜‰."Ò4J.

如图中蓝色的部分即IHDR数据块。

IDAT 数据块

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。这是一个可以存在多个数据块类型的数据块。它的作用就是存储着图像真正的数据。

因为它是可以存在多个的，所以即使我们写入一个多余的IDAT也不会多大影响肉眼对图片的观察

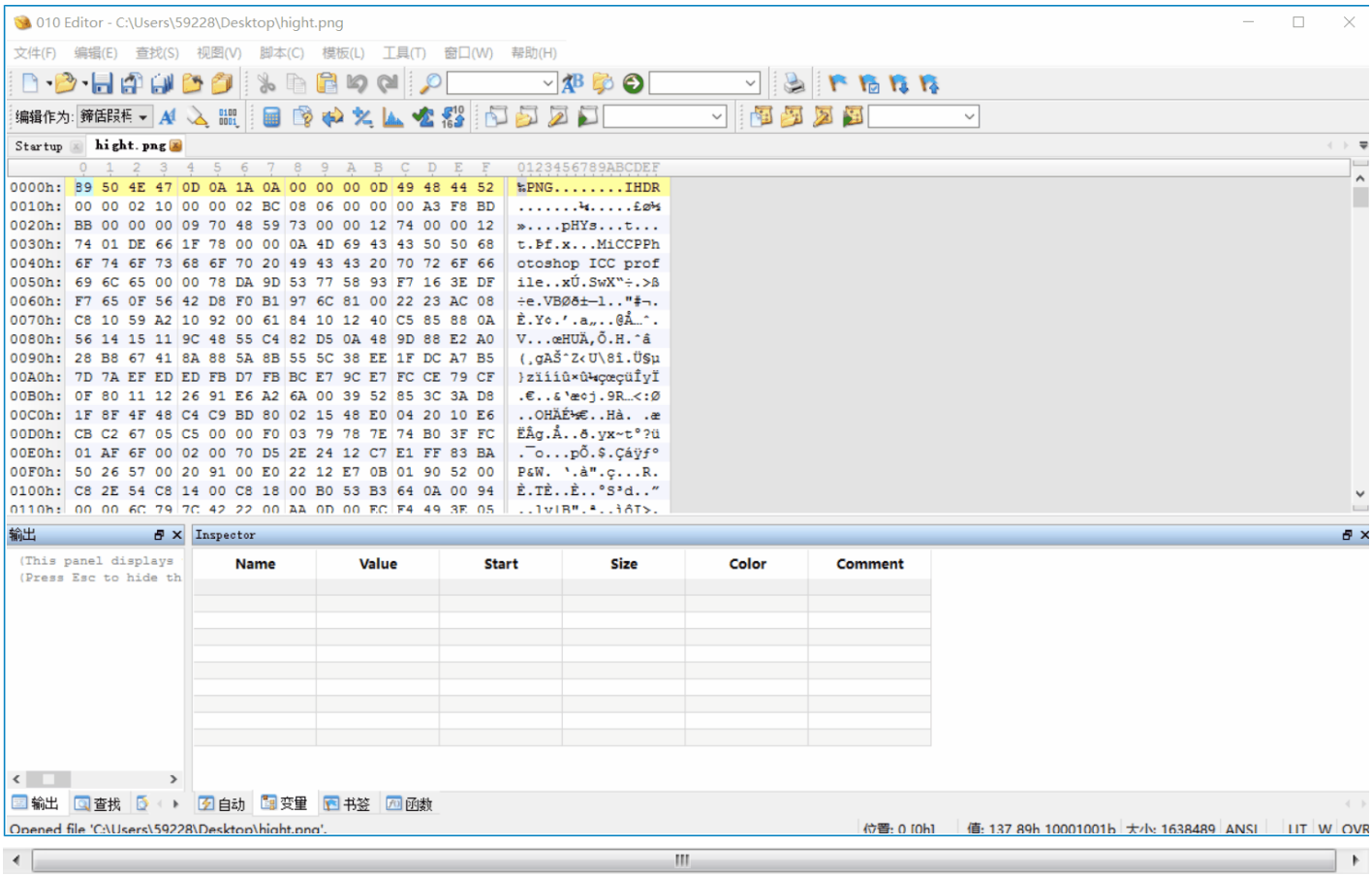
高度被修改引起的隐写：

背景知识中，我们了解到，图片的高度，宽度的值存放于PNG图片的文件头数据块，那么我们就是可以通过修改PNG图片的高度值，来对部分信息进行隐藏的。

- 在实验中找到隐写术目录，打开图片隐写，打开图片隐写第二部分文件夹
- 在该文件夹找到 hight.png，
- 双击打开图片，我们先确认一下图片内容并没有什么异常
- 正如前文所说，我们这个实验部分讲的是图片高度值被修改引起的的隐写方式，所以我们010Editor
- 在010Editor运行PNG模板，这样方便于我们修改PNG图片的高度值
- 找到PNG图片高度值对应的地方，然后修改为一个较大的值，并重新计算，修改CRC校验值，并保存文件
- 打开保存后的图片，发现底部看到了之前被隐写的信息

用010Editor打开图片，运行PNG模板

10 editor呢？因为这个16进制编辑器，有模版功能，当我们运行模版后，可以轻易的找到图片的各个数据块的位置以及内容。

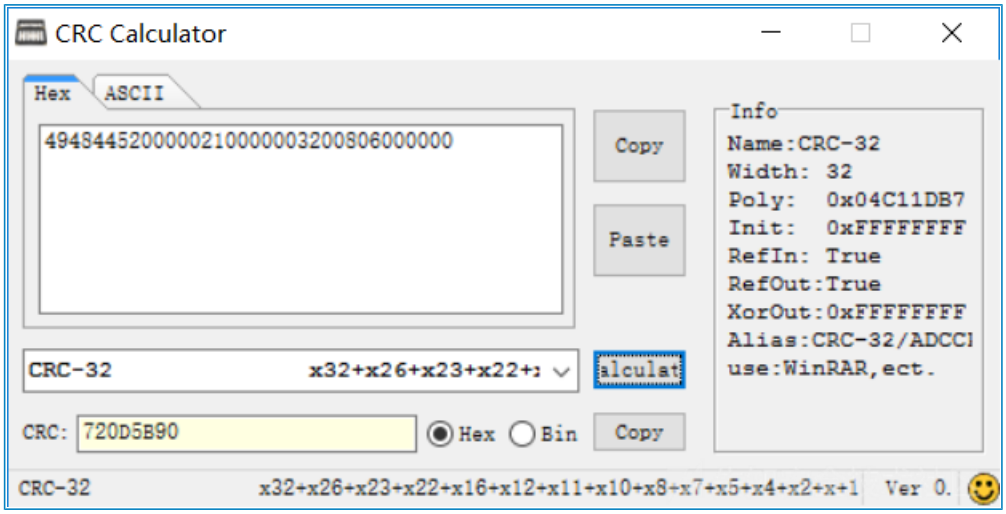


10 editor这个16进制编辑器，有模版功能，当我们运行模版后，可以轻易的找到图片的各个数据块的位置以及内容。

找到PNG图片高度值所对应的位置，并修改为一个较大的值

我们找到IHDR数据块，并翻到struct IHDR Ihdr位置，修改height的值到一个较大的值，如从700修改到800。

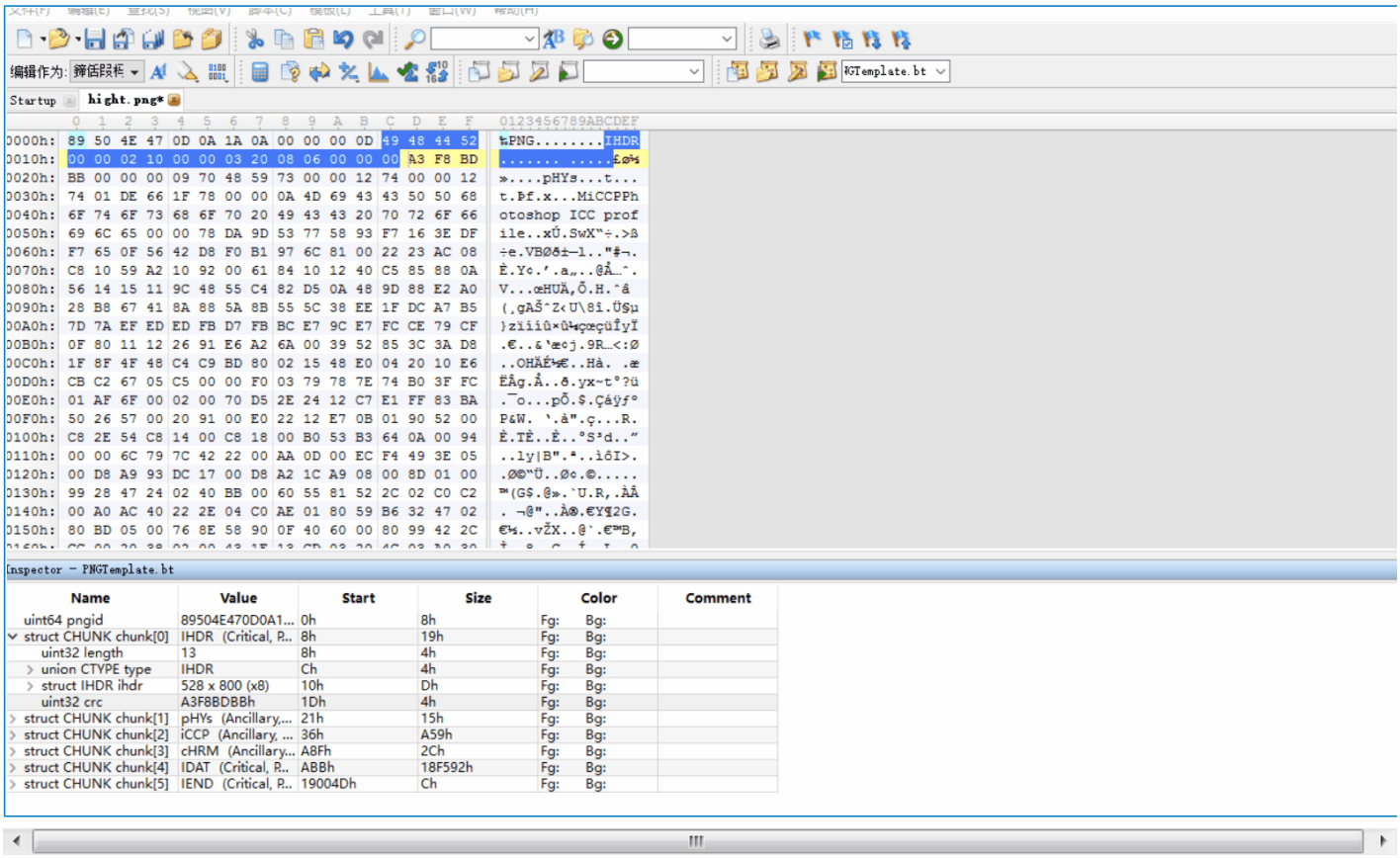
使用CRC Calculator重新计算CRC校验值



输入参数，然后点击Calculator计算，得到CRC值

为什么要重新计算CRC校验值呢？防止图片被我们修改后，自身的CRC校验报错，导致图片不能正常打开。

修改相应的CRC校验值，为我们重新计算后数值



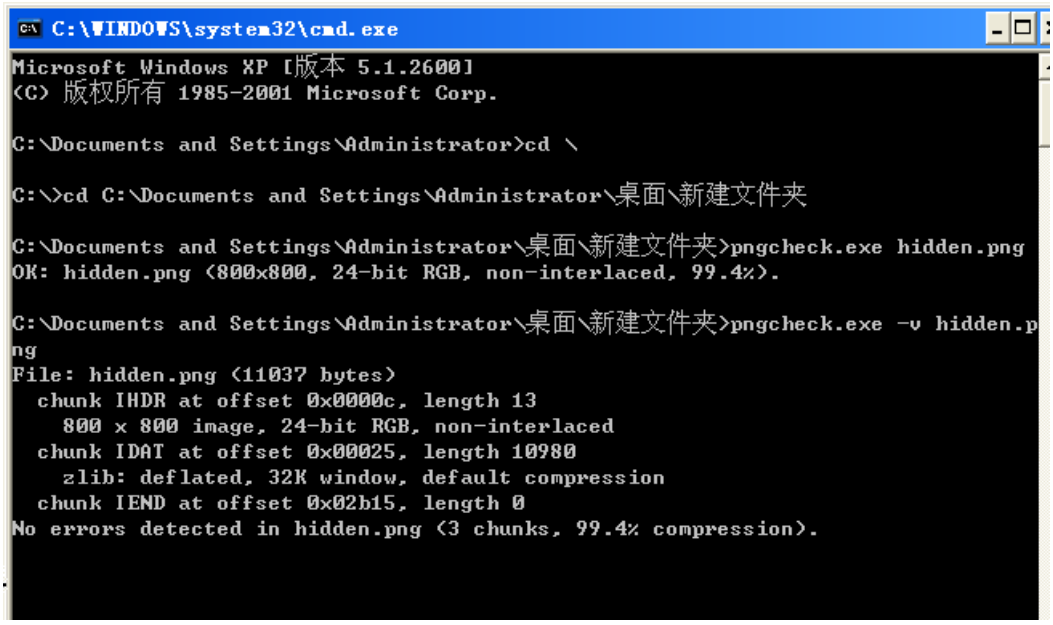
各位小伙伴可以思考一下： JPG图片是否也有这样的隐写形式呢？

如果感兴趣可以：了解JPG以及GIF等图片文件的格式。

第二部分：隐写信息以IDAT块加入图片

在背景知识中，我们提到了一个重要的概念就是图片的IDAT块是可以存在多个的，这导致了我们可以将隐写西信息以IDAT块的形似加入图片

- 在实验机找到隐写术目录，打开图片隐写，打开图片隐写第二部分文件夹
- 在该文件夹找到 hidden.png，
- 双击打开图片，我们先确认一下图片内容并没有什么异常
- 使用pngcheck先对图片检测
- 在pngcheck的检测下，我们会发现异常信息，我们对异常的块进行提取
- 编写脚本，提取异常信息（关于脚本，你可以上网搜索，也可以自己写）



我们使用命令：

```
pngcheck -v hidden.png
```

对图片的文件结构进行检测。

发现异常，并判断异常的原因

我们会发现，图片的的数据块形式是如下的

Type: IHDR

Size: 13

CRC： 5412913F

Pos： 33

Type: IDAT

Size: 10980

CRC： 98F96EEB

Pos： 11025

Type: IEND

Size: 0

CRC： AE426082

我们会惊讶的发现pos为11025的size居然为0，这是一块有问题的地方，我们可以怀疑，这一块是隐写的信息。

自己编写脚本代码（也可以去网上搜索）：

```
#!/usr/bin/python

from struct import unpack
from binascii import hexlify, unhexlify
import sys, zlib

# Returns [Position, Chunk Size, Chunk Type, Chunk Data, Chunk CRC]
def getChunk(buf, pos):
    a = []
    a.append(pos)
    size = unpack('!I', buf[pos:pos+4])[0]
    # Chunk Size
```

```
    a.append(buf[pos:pos+4])
    # Chunk Type
    a.append(buf[pos+4:pos+8])
    # Chunk Data
    a.append(buf[pos+8:pos+8+size])
    # Chunk CRC
    a.append(buf[pos+8+size:pos+12+size])
    return a

def printChunk(buf, pos):
    print 'Pos : '+str(pos)+''
    print 'Type: ' + str(buf[pos+4:pos+8])
    size = unpack('!I', buf[pos:pos+4])[0]
    print 'Size: ' + str(size)
    #print 'Cont: ' + str(hexlify(buf[pos+8:pos+8+size]))
    print 'CRC : ' + str(hexlify(buf[pos+size+8:pos+size+12]).upper())
    print

if len(sys.argv)!=2:
    print 'Usage: ./this Stegano_PNG'
    sys.exit(2)

buf = open(sys.argv[1]).read()
pos=0

print "PNG Signature: " + str(unpack('cccccccc', buf[pos:pos+8]))
pos+=8

chunks = []
for i in range(3):
    chunks.append(getChunk(buf, pos))
    printChunk(buf, pos)
    pos+=unpack('!I',chunks[i][1])[0]+12


decompressed = zlib.decompress(chunks[1][3])
# Decompressed data length = height x (width * 3 + 1)
print "Data length in PNG file : ", len(chunks[1][3])
print "Decompressed data length: ", len(decompressed)

height = unpack('!I',(chunks[0][3][4:8]))[0]
width = unpack('!I',(chunks[0][3][:4]))[0]
blocksize = width * 3 + 1
filterbits = ''
for i in range(0,len(decompressed),blocksize):
    bit = unpack('2401c', decompressed[i:i+blocksize])[0]
    if bit == '\x00': filterbits+='0'
    elif bit == '\x01': filterbits+='1'
    else:
        print 'Bit is not 0 or 1... Default is 0 - MAGIC!'
        sys.exit(3)

s = filterbits
endianess_filterbits = [filterbits[i:i+8][::-1] for i in xrange(0, len(filterbits), 8)]

flag = ''
for x in endianess filterbits:
```

```
    if x=='00000000': break
    flag += unhexlify('%x' % int('0b'+str(x), 2))

print 'Flag: ' + flag
```

最后得到答案：flag DrgnS{WhenYouGazeIntoThePNGThePNGAlsoGazezIntoYou}

题目.zip (0.414 MB) 下载附件

您可以考虑给博主来个小小的打赏以资鼓励，您的肯定将是我最大的动力。

作者： 落花四月
出处： https://www.cnblogs.com/lxz-1263030049/
关于作者：潜心于网络安全学习。如有问题或建议，请多多赐教！