图片隐写之尾部追加和内容覆盖法

原创

lansee_digital 于 2022-03-09 13:38:54 发布 43 收藏
分类专栏： 前端 文章标签： 前端

本文链接：https://blog.csdn.net/u012576295/article/details/123374857

★图片隐藏信息的用途

大致总结了一下，信息隐藏可以用于如下几种场合。

◇规避敏感词过滤

所谓的"敏感词过滤"，常翻墙的同学，应该都很熟悉了。用图片来隐藏信息，可以规避GFW的敏感词过滤。

◇规避肉眼审查

主要是为了躲开网站管理人员的人工审查。国内的很多网站，对于上传的图片，都会进行人工审查。如果能通过技术手段把信息隐藏在图片中，而图片本身又看不出什么异样，人工审核就看不出来。

◇传递加密信息

最后，图片还可以用来隐藏加密的信息。用图片来隐藏加密信息，除了具有加密的效果，还具有很大的欺骗性——因为外人难以知道一张图片是否包含有加密信息。

★准备工作——先压缩

下面，俺会介绍几种不同的隐藏方式。在动手之前，先说一下准备工作——把要隐藏得文件先用压缩工具（比如 7zip 或 WinRAR）压缩一下。

压缩有如下几个好处：

优点1: 如果你要隐藏的文件是文本格式或者 Office 格式，它内部的内容是明码的。如果里面包含敏感词，在通过网络传输时，会遭遇敏感词过滤。而压缩后的文件，原有的内容已经变得面目全非，可以规律敏感词过滤。

优点2: 压缩之后，体积变小，有利于增加隐蔽性。因此，应尽量使用"最大压缩"的选项。

优点3: 对于后面介绍的2种方法（尾部追加法、内容覆盖法），如果你隐藏的文件是压缩格式的，到时候提取信息会很简便——直接用压缩工具来解压，即可。

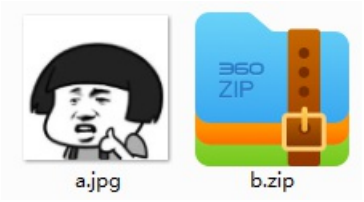
★尾部追加法

◇技术原理

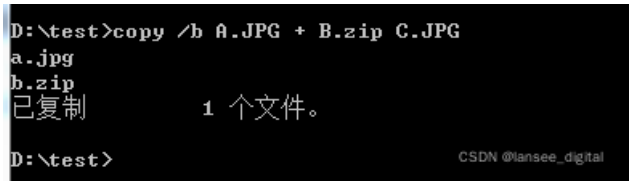
顾名思义，"尾部追加法"就是把要隐藏的文件追加到图片尾部。这种方法不会破坏图片原有的任何数据，因此，图片看起来和原来一模一样。

◇隐藏信息的步骤

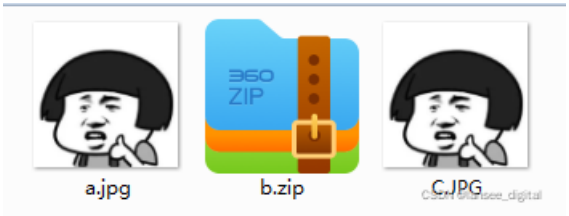
隐藏的过程很简单，用 Windows 内置的文件拷贝命令，即可完成。假设你的图片文件叫 A.JPG，需要隐藏的压缩文件叫 B.ZIP，那你只需要执行如下命令，就可以把两个文件合并成一个新文件。



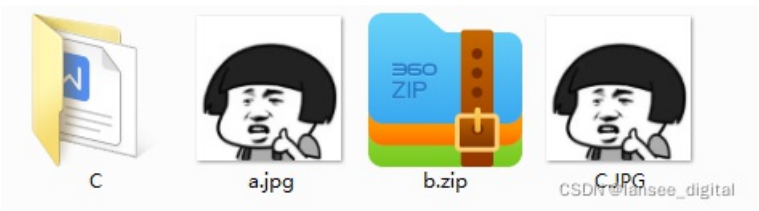

```
copy /b A.JPG + B.zip C.JPG
```



执行完如上命令，即可得到一个新的图片文件 C.JPG。这个图片文件的大小是前两者的总和。你可以用各种看图工具来打开 C.JPG，不会看到什么异常。



◇提取信息的步骤



由于你追加的是压缩文件，提取的时候就简单了——只要用压缩工具打开 C.JPG，就可以直接看到压缩包里面的内容了。

◇优点

1、 制作简单，一条 copy 命令就可以搞定；如果隐藏的是压缩文件，提取的过程也很简单。
2、 用看图工具看生成的新文件，还是跟原来一样。
3、 隐藏的文件，大小不受限制。比如，你可以在一张100K的图片尾部，追加200K的隐藏数据。

◇缺点

1、 由于隐藏的文件附加在尾部。当你把这个新的图片文件上传到某些贴图的网站，（假如这个网站对图片格式的校验比较严格）它有可能会发现图片尾部有多余的数据，并且会把这个多余的数据丢弃掉。
2、 追加后，图片的文件尺寸变大了。如果你追加的文件太大，容易被发现破绽。
比方说，一张640*480的 JPEG 图片，大小竟然有好几兆，对于有经验的IT技术人员，一下子就会觉得有猫腻。

★内容覆盖法

说完尾部追加的办法，再来介绍内容覆盖的办法。

◇技术原理

通常，图片文件都有包含2部分：文件头和数据区。而"内容覆盖法"，就是把要隐藏的文件，直接覆盖到图片文件的数据区的尾部。比方说，某图片有100K，其中文件头占1K，那么，数据区就是99K。也就是说，最多只能隐藏99K的文件。

切记：覆盖的时候，千万不可破坏文件头。文件头一旦破坏，这个图片文件就不再是一个合法的图片文件了。

使用这种方法，对图片文件的格式，是有讲究的——最好用 24位色的 BMP 格式。一来，BMP 格式本身比较简单，数据区随便覆盖，问题不大；二来，24位色的 BMP 相对其它的格式 BMP，文件尺寸更大，可以隐藏更多内容。

◇隐藏信息的步骤

用这个招数来隐藏信息，稍微有点麻烦，需要借助一些小工具。对于这种简单的活计，俺通常用Python脚本来搞定。以下是俺写的一个简单 Python 脚本。你的电脑中如果有Python环境，可以直接拿这个脚本去用。

事先声明：如下代码没有严格计算 BMP 的文件头尺寸，俺只是大致预留了 1024 字节，感觉应该够了。

- -

```
import sys

def embed(container_file, data_file, output_file) :
    container = open(container_file, "rb").read()
    data = open(data_file, "rb").read()

    if len(data)+1024 >= len(container) :
        print "Not enough space to save", data_file
    else :
        f = open(output_file, "wb")
        f.write(container[ : len(container)-len(data)])
        f.write(data)
        f.close()

if "__main__" == __name__ :
    try :
        if len(sys.argv) == 4 :
            embed(sys.argv[1], sys.argv[2], sys.argv[3])
        else :
            print "Usage:"
            print sys.argv[0], "container data output"
    except Exception,err :
        print err
```

上述Python的代码，很好懂，有编程基础的同学，10分钟之内就可以用自己熟悉的语言重写一个类似的。

另外，没学过 Python 的同学，如果有兴趣，可以看看俺之前写的系列帖子——

◇提取信息的步骤

和前一种方法类似。如果你覆盖的是压缩文件，提取的时候，可以用压缩工具打开图片，就可以直接看到压缩包里面的内容了。

◇优点

1、 图片的文件尺寸没变。
2、 虽然隐藏文件覆盖到数据区，破坏了原图像的内容。但是从格式上来讲，该图片文件的格式还是合法的。因此，你可以把这种图片上传到各种贴图的网站，技术上不会出问题。
3、 如果隐藏的是压缩文件，提取的过程很简单。

◇缺点

1、 由于隐藏的文件覆盖了数据区，因此，图片在显示的时候，会有一块区域变成灰蒙蒙的。
2、 隐藏文件的大小，有一定的限制——不能大于图片数据区的尺寸。
3、 对图片格式有一定要求。此处再啰嗦一下，建议用 24位色的 BMP 格式。