文法类隐写与分析技术实现

iAmAnyus1

1.西南科技大学计算机科学与技术学院 四川绵阳 621010

**摘要 ：** BMP是英文Bitmap（位图）的缩写，它是Windows操作系统中的一种标准图像文件格式，可以被多种Windows应用程序所识别和使用。但是，BMP文件占用的磁盘空间很大，不利于存储和传输。本实验从文法类隐写与分析技术的三个方面进行探索，分别是文件头隐写、文件尾隐写和数据区覆盖隐写。实验方法是：使用python语言读取BMP文件的二进制数据，并进行插入或覆盖操作，实现简单的隐写和分析功能。实验结论是：文件头隐写最容易被发现，因为它会改变文件头的信息；数据区覆盖隐写在覆盖数据很少时，如果没有原始图片进行对比，很难被检测出来；文件尾隐写可以通过16进制查看器直接看到隐藏的数据。

**The implementation of grammar-type cryptography and analysis technology**

iAmAnyus 1

(1. School of Computer Science and Technology,Southwest University of Science and Technology, Mianyang 621010, China )

**Abstract:** BMP stands for Bitmap, which is a standard image file format in the Windows operating system that can be recognized and used by various Windows applications. However, BMP files take up a lot of disk space, which is not conducive to storage and transmission. This experiment explores three aspects of grammar-based steganography and analysis techniques, namely file header steganography, file tail steganography and data area overwrite steganography. The experimental method is: use python language to read the binary data of BMP files, and perform insertion or overwrite operations, to achieve simple steganography and analysis functions. The experimental conclusion is: file header steganography is the easiest to detect, because it changes the information of the file header; data area overwrite steganography is hard to detect when the overwritten data is very small, if there is no original image for comparison; file tail steganography can be seen directly by using a hexadecimal viewer.

**Keywords: BMP; Python； Picture steganography; Data area; File header; redundancy**

# 引 言

## 0.1 BMP简述

BMP是Bitmap（位图）的英文缩写，是Windows操作系统的一种标准图像文件格式，可以兼容多种Windows应用程序。由于Windows操作系统的普及和Windows应用程序的发展，BMP位图格式也得到了广泛的应用。这种格式的优点是图像信息丰富，几乎没有压缩损失，但缺点是文件体积太大，不便于存储和传输。因此，BMP格式更适合在单机上使用。

## 原理介绍

bmp的原理是基于二进制位图的概念，它是一种将图像数据表示为由0和1组成的矩阵的方法。每个0或1称为一个位，每个位对应一个像素的颜色。位的数量决定了颜色的种类，比如1位只能表示两种颜色，4位可以表示16种颜色，24位可以表示1677万种颜色。位的排列方式决定了图像的形状，比如一个8x8的位图可以表示一个8x8的图像，每个像素占一个位。

bmp文件就是将这种位图的数据按照一定的格式存储在文件中，方便传输和显示。bmp文件的格式由三部分组成：文件头，信息头和图像数据。文件头和信息头用来存储图像的元数据，如文件类型，文件大小，图像的宽度，高度，颜色深度，压缩方式等。图像数据用来存储图像的像素值，按照从下到上，从左到右的顺序排列。每个像素的颜色由颜色深度和调色板决定。调色板是一个可选的部分，它存储了图像使用的颜色表，每个颜色占4字节，分别表示蓝色，绿色，红色和保留值。如果颜色深度为1，4或8，那么调色板是必须的，否则可以省略。

## 隐写方法原理

1. 头文件隐写：修改文件头和信息头中的保留字段隐藏信息
2. 文件尾部隐写：在图像文件尾部添加另一个文件的数据，秘密信息存放在文件尾部可以减少修改文件头的数据量，仅需修改文件头中文件长度的值即可。
3. 数据区覆盖隐写，如果数据量大，会导致，图片出现缺失：

# 1 BMP结构

## 1.1 BMP头部信息

bmp的结构由三部分组成，分别是：

- 文件头：文件头是bmp文件的第一部分，它包含了文件的基本信息，如文件类型，文件大小，数据偏移量等。文件头的长度为14字节，包含以下字段：

bfType：文件类型，必须为"BM"，占2字节。

bfSize：文件大小，单位为字节，占4字节。bfReserved1：保留字段，必须为0，占2字节。

bfReserved2：保留字段，必须为0，占2字节。

bfOffBits：图像数据的起始位置，相对于文件头的偏移量，单位为字节，占4字节。

- 信息头：信息头是bmp文件的第二部分，它包含了图像的详细信息，如宽度，高度，颜色深度，压缩方式等。信息头的长度可以有多种，最常见的是40字节，包含以下字段：

biSize：信息头的大小，单位为字节，占4字节。

biWidth：图像的宽度，单位为像素，占4字节。

biHeight：图像的高度，单位为像素，占4字节。

biPlanes：颜色平面数，必须为1，占2字节。

biBitCount：颜色深度，即每个像素的位数，常见的有1，4，8，16，24，32，占2字节。

biCompression：压缩方式，0表示不压缩，1表示RLE8压缩，2表示RLE4压缩，3表示每个像素的颜色由指定的掩码决定，占4字节。

biSizeImage：图像数据的大小，单位为字节，如果不压缩，可以为0，占4字节。

biXPelsPerMeter：水平分辨率，单位为像素/米，占4字节。

biYPelsPerMeter：垂直分辨率，单位为像素/米，占4字节。

biClrUsed：调色板中的颜色数，如果为0，表示使用所有颜色，占4字节。

biClrImportant：重要的颜色数，如果为0，表示所有颜色都重要，占4字节。

图像数据：图像数据是bmp文件的第三部分，它包含了图像的像素值，按照从下到上，从左到右的顺序排列。每个像素的颜色由颜色深度和调色板决定。调色板是一个可选的部分，它存储了图像使用的颜色表，每个颜色占4字节，分别表示蓝色，绿色，红色和保留值。如果颜色深度为1，4或8，那么调色板是必须的，否则可以省略。

# 2 BMP位图读取

## 2.1 16进制读取

Bmp位图文件是基于16进制的，我们需要以16进制的方式去读取文件，通过使用python的numpy库，我们可以将BMP图片的文件头用16进制读出，观察。如图1所示。

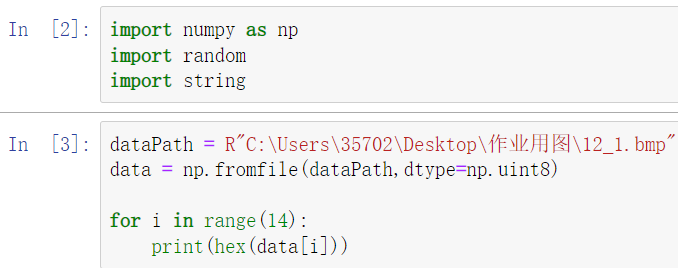


图1. 读取图像

我们将BMP图片的文件用16进制读出

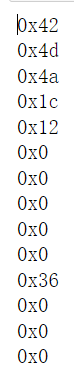


图2. 输出BMP图片前16位

使用UltraEdit软件，读取BMP图片的文件结构，然后与图2进行比较，是否一致



图3. UltraEdit查看结果

# 3 文法类隐写与分析技术实现

## 3.1位图头文件冗余的数据隐写、分析和提取

## 3.1.1 头文件冗余的数据隐写原理

我们的目的是利用 BMP 图片头文件进行隐写。通过学习 BMP 图片文件格式原理，我们知道：

BMP 图片第1-2 字节是“BM”，表示这是一个 BMP 文件，不能修改。

BMP 图片第3-6 字节是文件的大小，单位是字节，不能修改。

BMP 图片第7-10 字节是保留字段，没有实际意义，可以修改。

BMP 图片第11-14 字节是数据区的偏移量，单位是字节，不能修改。

因此，我们可以在保留字段中嵌入我们想要隐藏的信息，达到文件头隐写的效果。

## 3.1.2 头文件冗余的数据隐写

我们输入要隐藏的4个字节信息，将其以ASCLL码的形式，存储到数组中，再用我们之前得到的BMP图片的文件格式的第6-10字节进行覆盖（见图6），得到图5。



图4. 头文件数据隐写代码

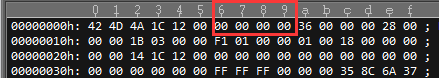


图5. 隐写前

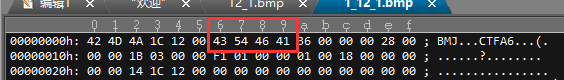


图6. 隐写后

## 3.1.3 头文件冗余数据的分析

根据3.1.1的原理分析，BMP图片如果想进行头文件隐写，那么保留字就不能为0，我们只需要输出第6-10字节的部分，观察是否为0，如果为0那就表示有信息隐藏，若为1，反之。

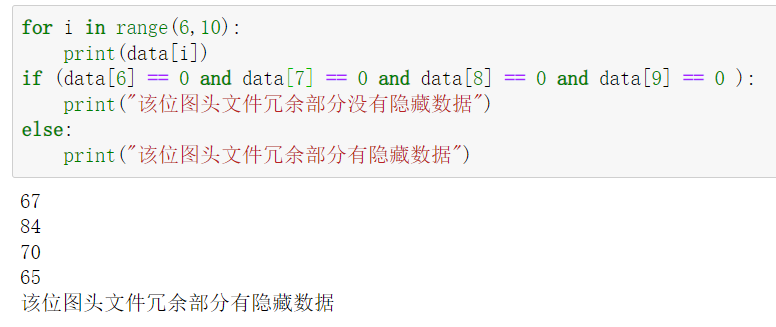


图7. 判断是否有隐藏数据

## 3.1.4 头文件冗余数据的提取

因为头文件只有第6-9字符进行改动，我们将其用数组保存并按照格式化字符及其 ASCII 码提取出来就可以了。

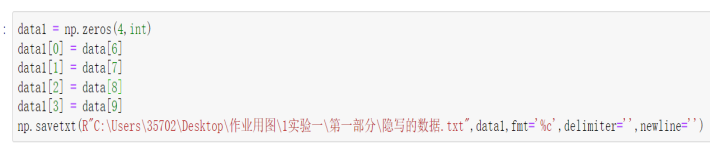


图8. 提取隐藏数据

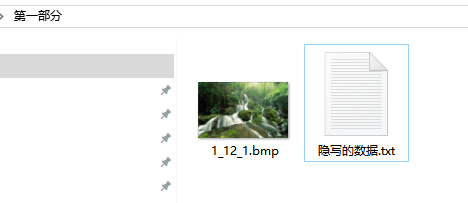


图9. 导出后的文件

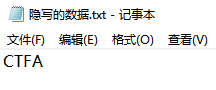


图10. 隐藏数据的内容

## 3.2文件尾部追加的数据隐写、分析和提取

## 3.2.1 使用CMD面板进行数据量较小的隐写（小于1%）

我们新建一个文本，输入我们需要隐藏的信息，然后将要隐藏在哪个JPG图片也准备好，然后再CMD命令窗口，使用copy语句将其合并。

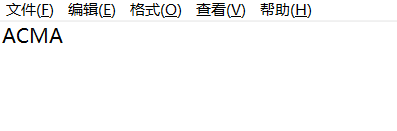


图11. 准备的txt文件

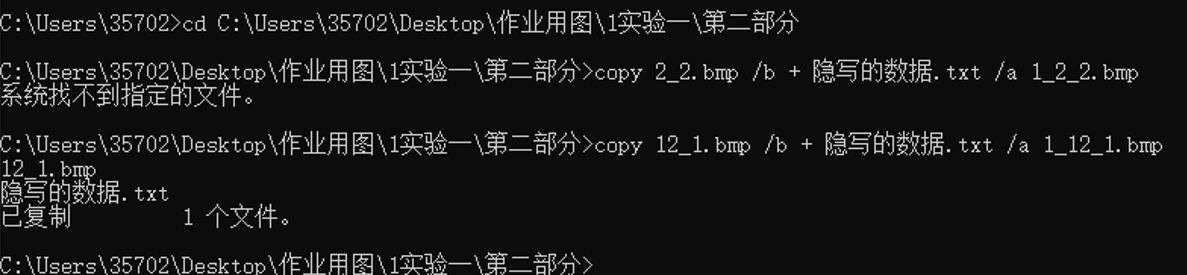


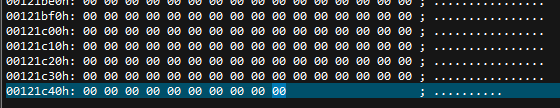
图12. 执行命令

图13. 执行命令前的BMP图片文件尾部

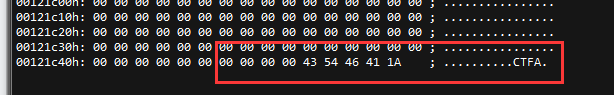


图14 执行命令后的BMP图片文件尾部

## 3.2.2 使用python进行数据量较小的隐写（小于1%）

我们打开图片，使用write() 方法将我们输入的隐藏信息追加到图片尾部。



图15 文件尾部隐写代码

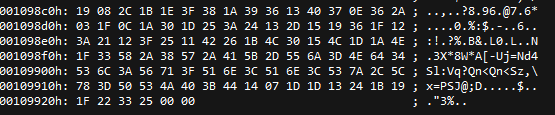


图16 文件尾部隐写前

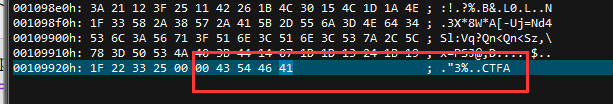


图17 文件尾部隐写后

## 3.2.3 使用CMD面板进行数据量较大的隐写（大于25%）

与3.2.1的步骤类似，只是隐藏的数据更大了而已。

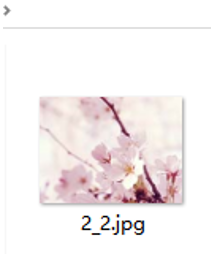


图18 需要进行信息隐藏的材料



图19 执行命令

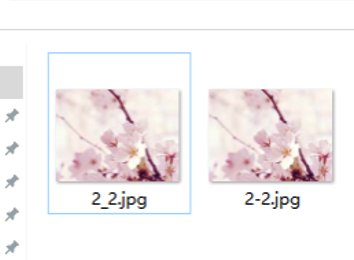


图20 执行命令后的情况

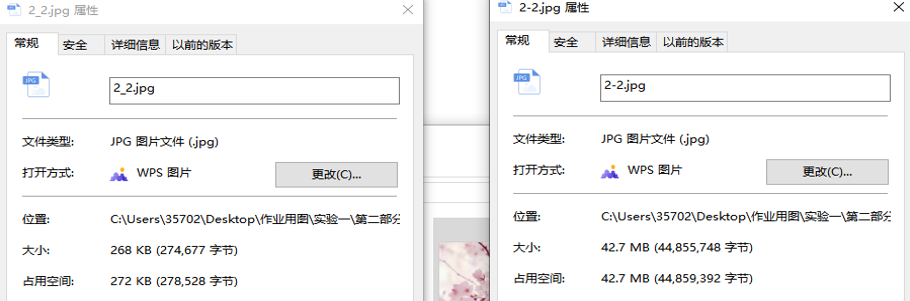


图21 对比两者的内存的大小

## 3.2.3 使用python进行数据量较大的隐写（大于25%）

可以参照3.2.2的内容进行实验，当我们输入的数据过大时，仍然能将其进行隐写。

## 3.2.4 文件尾部追加的数据分析

我们可以根据 JPG 图片文件尾部的隐写数据量的不同，采用不同的方法来检测是否有隐写。如果隐写数据量较小，我们可以使用 UltraEdit 等软件，观察文件尾部的结构是否有异常（如图 17）。如果隐写数据量较大，我们可以查看图片的属性，比较文件大小是否合理（如图 21）。如果文件大小过大，就有可能是有隐写数据的。

## 3.2.5 文件尾部追加的数据提取

如果是直接使用CMD面板进行合并的，我们可以将后缀名改为.zip，并将它解压，就能得到隐藏的数据信息。如果是使用python代码对其进行尾部添加，可以使用数组的切片功能，直接将隐藏的数据读取出来，保存到文件中。

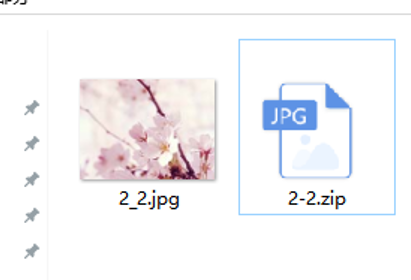


图22 将图片后缀修改为.zip



图23 压缩得到信息

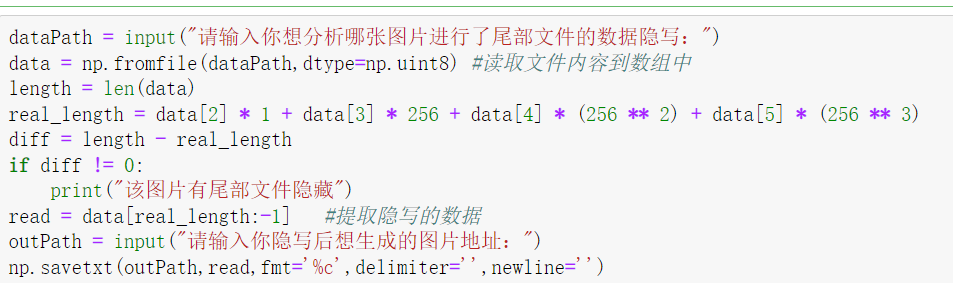


图24 尾部隐藏信息提取的代码

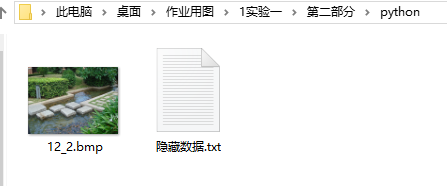


图25 尾部隐藏信息提取的文件

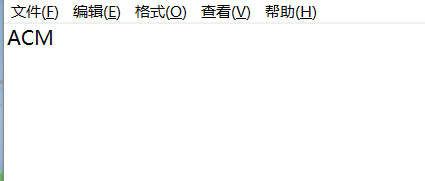


图26 尾部隐藏的信息

## 3.3 BMP位图格式图像的数据区进行数据直接覆盖方式的隐写和分析

## 3.3.3 完成BMP位图格式图像的数据区进行数据直接覆盖方式的隐写（大于25%）

我们使用Python脚本对其进行覆盖，但是覆盖的压缩包的大小不能大于等于我们用于被覆盖的图片。我们可以将后缀名改为.zip，并将它解压，就能得到隐藏的数据信息

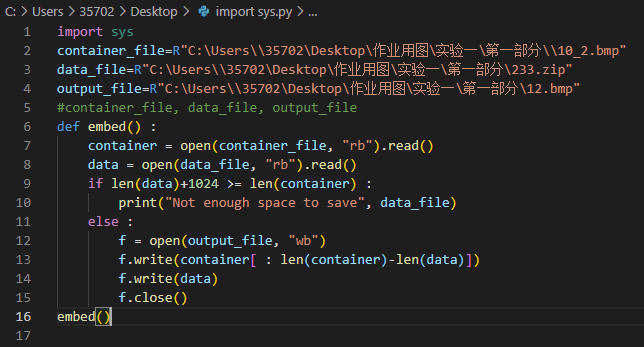


图27 数据区直接覆盖隐写代码

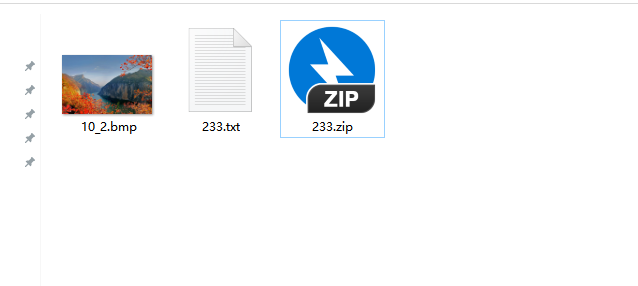


图28 需要覆盖隐写的材料

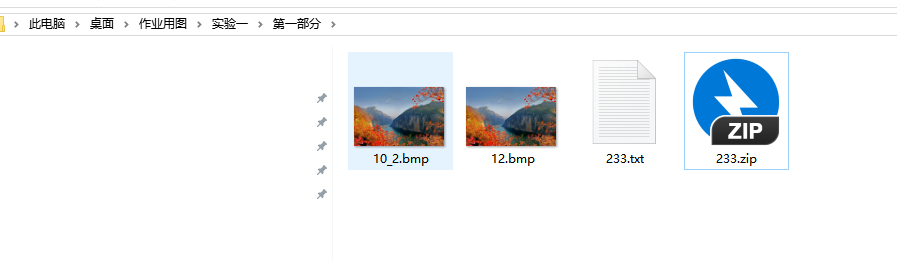


图29 数据区直接覆盖隐写后

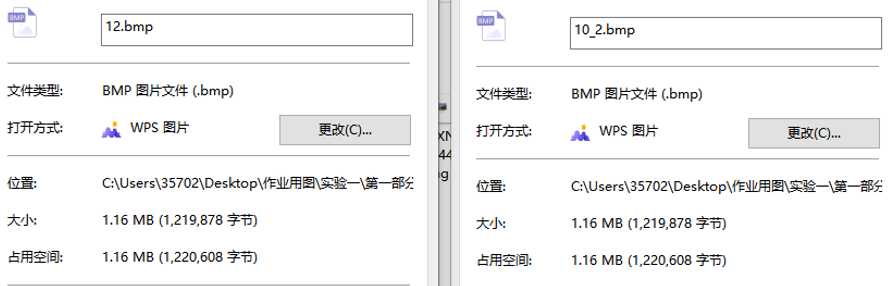


图29 隐写后与隐写前的属性对比

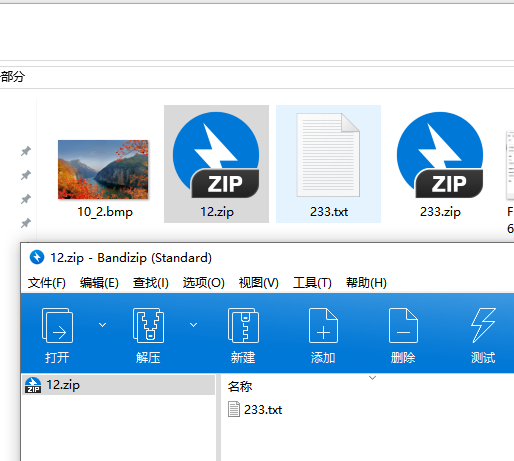


图30 隐写后的隐藏信息提取

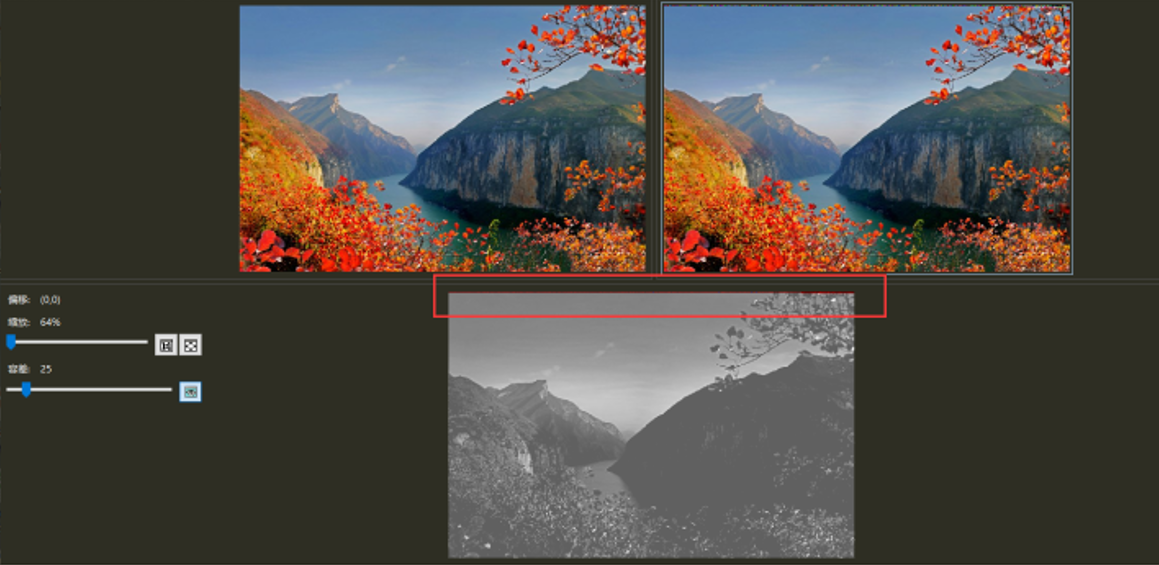
## 3.3.4 完成BMP位图格式图像的数据区进行数据直接覆盖方式的隐写（小于1%）

可以根据3.3.3的代码进行数据区直接覆盖隐写。

## 3.3.5 BMP位图格式图像的数据区进行数据直接覆盖方式的分析

虽我们可以利用图片数据区来隐藏文件。这种方法虽然会破坏原图像的一部分内容，但是不会影响图片文件的格式和大小，所以可以正常上传到各种贴图网站。但是，这种方法也有一些缺点和要求：

隐藏文件不能太大，否则会超过图片数据区的容量。

可以使用 Beyond Compare 等软件来对比原图和隐写后的图，看看有没有明显的差异。

# 4 结论和心得

在这次实验中，我学到了很多关于 BMP 文件格式和文件长度计算的知识。我在实验过程中也查阅了很多文献，但是发现很多文献不太有用，或者和我想要的内容不太一致。但经过不少的网络查询资料，最终完成了这次实验。

# 参考文献(References)

[1]星辞归野. 基于Python的BMP信息隐藏实验CSDN:.

[2]nanfangwudi. 信息隐藏之bmp图片的冗余，尾部追加和数据区覆盖的隐写方法和隐写分析 CSDN:.