Python3 图片隐写术

转载

oxuzhenyi 于 2017-03-21 14:47:27 发布
分类专栏： 实验楼课程 文章标签： python 图片隐写术


实验楼课程 专栏收录该内容
171 篇文章 24 订阅
订阅专栏

Python3 图片隐写术

一、实验简介

wikipedia 关于隐写术的介绍：

> **隐写术**是一门关于信息隐藏的技巧与科学，所谓信息隐藏指的是不让除预期的接收者之外的任何人知晓信息的传递事件或者信息的内容。隐写术的英文叫做Steganography，来源于特里特米乌斯的一本讲述密码学与隐写术的著作Steganographia，该书书名源于希腊语，意为“隐秘书写”。

1.1. 知识点

- Pillow 模块
- 最低有效位
- lambda 表达式递归
- UTF-8 编码

1.2. 效果展示

可以看到“施法”前后的图片肉眼看不出区别，然而图片却真实的隐藏了一些数据在里面。

二、实验步骤

2.1. 实验原理

还是引用 wikipedia 的解释：

> 载体文件（cover file）相对隐秘文件的大小（指数据含量，以比特计）越大，隐藏后者就越加容易。
>
> 因为这个原因，数字图像（包含有大量的数据）在因特网和其他传媒上被广泛用于隐藏消息。这种方法使用的广泛程度无从查考。例如：一个24位的位图中的每个像素的三个颜色分量（红，绿和蓝）各使用8个比特来表示。如果我们只考虑蓝色的话，就是说有2^8种不同的数值来表示深浅不同的蓝色。而像11111111和11111110这两个值所表示的蓝色，人眼几乎无法区分。因此，这个最低有效位就可以用来存储颜色之外的信息，而且在某种程度上几乎是检测不到的。如果对红色和绿色进行同样的操作，就可以在差不多三个像素中存储一个字节的信息。
>
> 更正式一点地说，使隐写的信息难以探测的，也就是保证“有效载荷”（需要被隐蔽的信号）对“载体”（即原始的信号）的调制对载体的影响看起来（理想状况下甚至在统计上）可以忽略。这就是说，这种改变应该无法与载体中的噪声加以区别。
>
> （从信息论的观点来看，这就是说信道的容量必须大于传输“表面上”的信号的需求。这就叫做信道的冗余。对于一幅数字图像，这种冗余可能是成像单元的噪声；对于数字音频，可能是录音或者放大设备所产生的噪声。任何有着模拟放大级的系统都会有所谓的热噪声（或称“1/f”噪声），这可以用作掩饰。另外，有损压缩技术（如JPEG）会在解压后的数据中引入一些误差，利用这些误差作隐写术用途也是可能的。）
>
> 隐写术也可以用作数字水印，这里一条消息（往往只是一个标识符）被隐藏到一幅图像中，使得其来源能够被跟踪或校验。

总而言之，本实验便是利用图片四个颜色分量（rgba）的最低有效位（英语：Least Significant Bit，lsb）来隐藏信息（本实验隐藏的是文字）

2.2. 安装包

本实验用到了 pillow 这个模块，在安装它前更新源

```
$ sudo apt-get update
```

首先我们需要处理一个问题：当前实验楼的环境中 python3 命令使用的 python 版本为 3.5，但源中却没有 python3.5-dev，这会导致安装 Pillow 出错。所以我们必须将 python3 命令使用的 python 版本切换为 3.4，然后再安装 python3-dev 和 python3-setuptools。

```
$ sudo update-alternatives --install /usr/bin/python3 python3 /usr/bin/python3.4 70 --slave /usr/bin/pyth
on3m python3m /usr/bin/python3.4m
$ sudo apt-get install python3-dev python3-setuptools
```

然后安装 Pillow 依赖包

```
$ sudo apt-get install libtiff5-dev libjpeg8-dev zlib1g-dev \
    libfreetype6-dev liblcms2-dev libwebp-dev tcl8.6-dev tk8.6-dev python-tk
```

最后安装 Pillow：

```
$ sudo pip3 install Pillow
```

2.3. 程序实现

先导入 Pillow 模块：

```
from PIL import Image
```

2.3.1. 编码

我们首先设计将隐藏信息编码到图片中的函数 `encodeDataInImage()`，其有两个参数，分别是用作载体的图片对象和需要被隐藏的字符串。也就是说我们可以这样调用它：

```
encodeDataInImage(Image.open("steganographia.png"), '你好世界，Hello world!')
```

`encodeDataInImage()` 函数如下：

```
def encodeDataInImage(image, data):
    evenImage = makeImageEven(image)  # 获得最低有效位为 0 的图片副本
    binary = ''.join(map(constLenBin,bytearray(data, 'utf-8'))) # 将需要被隐藏的字符串转换成二进制字符串
    if len(binary) > len(image.getdata()) * 4:  # 如果不可能编码全部数据， 抛出异常
        raise Exception("Error: Can't encode more than " + len(evenImage.getdata()) * 4 + " bits in this
image. ")
    encodedPixels = [(r+int(binary[index*4+0]),g+int(binary[index*4+1]),b+int(binary[index*4+2]),t+int(bi
nary[index*4+3])) if index*4 < len(binary) else (r,g,b,t) for index,(r,g,b,t) in enumerate(list(evenImage
.getdata()))] # 将 binary 中的二进制字符串信息编码进像素里
    encodedImage = Image.new(evenImage.mode, evenImage.size)  # 创建新图片以存放编码后的像素
    encodedImage.putdata(encodedPixels)  # 添加编码后的数据
    return encodedImage
```

`makeImageEven()` 函数的实现如下：

```
def makeImageEven(image):
    pixels = list(image.getdata())  # 得到一个这样的列表： [(r,g,b,t),(r,g,b,t)...]
    evenPixels = [(r>>1<<1,g>>1<<1,b>>1<<1,t>>1<<1) for [r,g,b,t] in pixels]  # 更改所有值为偶数（魔法般的
移位）
    evenImage = Image.new(image.mode, image.size)  # 创建一个相同大小的图片副本
    evenImage.putdata(evenPixels)  # 把上面的像素放入到图片副本
    return evenImage
```

相关函数的文档链接：

> Image.getdata()
> PIL.Image.new()
> PIL.Image.Image.putdata()

`encodeDataInImage()` 中， `bytearray()` 将字符串转换为整数值序列（数字范围是 0 到 2^8-1），数值序列由字符串的字节数据转换而来，如下图：

utf-8 编码的中文字符一个就占了3个字节，那么四个字符共占 3×4=12 个字节，于是共有12个数字。（可以在右下角切换到中文输入法，这样就能输入中文了）

然后 `map(constLenBin,bytearray(data, 'utf-8'))` 对数值序列中的每一个值应用 `constLenBin()` 函数，将十进制数值序列转换为二进制字符串序列。

```
def constLenBin(int):
    binary = "0"*(8-(len(bin(int))-2))+bin(int).replace('0b','')  # 去掉 bin() 返回的二进制字符串中的 '0b'
，并在左边补足 '0' 直到字符串长度为 8
    return binary
```

在这里 `bin()` 的作用是将一个 int 值转换为二进制字符串，详见： https://docs.python.org/3/library/functions.html#bin

2.3.2. 解码

`decodeImage()` 返回图片解码后的隐藏文字，其接受一个图片对象参数。

```
def decodeImage(image):
    pixels = list(image.getdata())  # 获得像素列表
    binary = ''.join([str(int(r>>1<<1!=r))+str(int(g>>1<<1!=g))+str(int(b>>1<<1!=b))+str(int(t>>1<<1!=t))
for (r,g,b,t) in pixels])  # 提取图片中所有最低有效位中的数据
    # 找到数据截止处的索引
    locationDoubleNull = binary.find('0000000000000000')
    endIndex = locationDoubleNull+(8-(locationDoubleNull % 8)) if locationDoubleNull%8 != 0 else location
DoubleNull
    data = binaryToString(binary[0:endIndex])
    return data
```

> str.find()

找到数据截止处所用的字符串 `'0000000000000000'` 很有意思，他的长度为16，而不是直觉上的 8，因为两个包含数据的字节的接触部分可能有 8 个 0。

`binaryToString()` 函数将提取出来的二进制字符串转换为隐藏的文本：

```
def binaryToString(binary):
    index = 0
    string = []
    rec = lambda x, i: x[2:8] + (rec(x[8:], i-1) if i > 1 else '') if x else ''
    # rec = lambda x, i: x and (x[2:8] + (i > 1 and rec(x[8:], i-1) or '')) or ''
    fun = lambda x, i: x[i+1:8] + rec(x[8:], i-1)
    while index + 1 < len(binary):
        chartype = binary[index:].index('0')
        length = chartype*8 if chartype else 8
        string.append(chr(int(fun(binary[index:index+length],chartype),2)))
        index += length
    return ''.join(string)
```

要看明白这个，必须要先搞懂 UTF-8 编码的方式，可以在 wikipedia 上了解 UTF-8 编码：https://zh.wikipedia.org/wiki/UTF-8

UTF-8 是 UNICODE的一种变长度的编码表达方式，也就是说一个字符串中，不同的字符所占的字节数不一定相同，这就给我们的工作带来了一点复杂度，如果我们要支持中文的话（那不是废话么）。

码点的位数	码点起值	码点终值	字节序列	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
7	U+0000	U+007F	1	`0xxxxxxx`					
11	U+0080	U+07FF	2	`110xxxxx`	`10xxxxxx`				
16	U+0800	U+FFFF	3	`1110xxxx`	`10xxxxxx`	`10xxxxxx`			
21	U+10000	U+1FFFFF	4	`11110xxx`	`10xxxxxx`	`10xxxxxx`	`10xxxxxx`		
26	U+200000	U+3FFFFFF	5	`111110xx`	`10xxxxxx`	`10xxxxxx`	`10xxxxxx`	`10xxxxxx`	
31	U+4000000	U+7FFFFFFF	6	`1111110x`	`10xxxxxx`	`10xxxxxx`	`10xxxxxx`	`10xxxxxx`	`10xxxxxx`

在上图中，只有 `x` 所在的位置（也即是字节中第一个 0 之后的数据）存储的是真正的字符数据，因此我们使用下面两个匿名函数来提取出这些数据：

```
rec = lambda x, i: x[2:8] + (rec(x[8:], i-1) if i > 1 else '') if x else ''
fun = lambda x, i: x[i+1:8] + rec(x[8:], i-1)
```

`fun()` 接受 2 个参数，第一个参数为表示一个字符的二进制字符串，这个二进制字符串可能有不同的长度（8、16、24...48）；第二个参数为这个字符占多少个字节。

`lambda x, i: x[i+1:8] + rec(x[8:], i-1)` 中 `x[i+1:8]` 获得第一个字节的数据，然后调用 rec()，以递归的方提取后面字节中的数据。

这里要提一句，`rec = lambda x, i: x[2:8] + (rec(x[8:], i-1) if i > 1 else '') if x else ''`，你可能对在表达式里面引用了 `rec` 感到不可理解，的确，严格意义上这样是不能实现递归的，但在 python 里这样是可以的，这就是 python 的语法糖了。

使用 lambda 表达式写递归从来不是一件简单的事，因为匿名函数引用自身并不简单，大家可以参考一下大牛刘未鹏的博文：http://blog.csdn.net/pongba/article/details/1336028

我们注意到，字符的字节数据中，第一个字节开头 `1` 的数目便是字符所占的字节数：

```
chartype = binary[index:].index('0') # 存放字符所占字节数，一个字节的字符会存为 0
```

`string.append(chr(int(fun(binary[index:index+length],chartype),2)))` 这一行中用到的函数 `int()` 以及 `chr` 的作用如下：

`int()`: 接受两个参数，第一个参数为数字字符串，第二个参数为这个数字字符串代表的数字的进制。详见：https://docs.python.org/3/library/functions.html#int

`chr()`：接受一个参数，参数为 int 值，返回 Unicode 码点为这个 int 值的字符。见下图：

`while` 循环的最后我们将当前字符的索引增加当前字符的长度，得到下一个字符的索引。

这样，我们可以识别出二进制字符串中哪些部分代表哪些字符了，然后就能调用 `fun()` 取得各个字符的数据了。

2.4. 测试效果

输入下面的命令获得测试用图片：

```
$ wget http://labfile.oss.aliyuncs.com/courses/651/coffee.png
```

在我们的代码后添加这两行：

```
encodeDataInImage(Image.open("coffee.png"), '你好世界，Hello world!').save('encodeImage.png')
print(decodeImage(Image.open("encodeImage.png")))
```

假设你的源代码文件为 steganography.py：

```
$ python3 steganography
```

它应该打印出 "你好世界，Hello world!"。

三、实验总结

本实验了解了图片隐写术的原理，这是一个很经典的技术了，完成本试验后应该明白了 lsb 到底是什么东西，有什么用。并且，本实验也用了不少的“魔法”（移位，lambda 递归，列表推导式...），这些”魔法“大家也应该掌握。

四、完整代码

使用下面的命令可以下载源代码：

```
$ wget http://labfile.oss.aliyuncs.com/courses/651/steganography.py
```

贴出全部代码：

```
from PIL import Image

"""
取得一个 PIL 图像并且更改所有值为偶数（使最低有效位为 0）
"""
def makeImageEven(image):
    pixels = list(image.getdata())  # 得到一个这样的列表： [(r,g,b,t),(r,g,b,t)...]
    evenPixels = [(r>>1<<1,g>>1<<1,b>>1<<1,t>>1<<1) for [r,g,b,t] in pixels]  # 更改所有值为偶数（魔法般的移位）
    evenImage = Image.new(image.mode, image.size)  # 创建一个相同大小的图片副本
    evenImage.putdata(evenPixels)  # 把上面的像素放入到图片副本
    return evenImage

"""
内置函数 bin() 的替代，返回固定长度的二进制字符串
"""
def constLenBin(int):
    binary = "0"*(8-(len(bin(int))-2))+bin(int).replace('0b','')  # 去掉 bin() 返回的二进制字符串中的 '0b'，并在左边补足 '0' 直到字符串长度为 8
    return binary

"""
```

```
将字符串编码到图片中
"""
def encodeDataInImage(image, data):
    evenImage = makeImageEven(image)  # 获得最低有效位为 0 的图片副本
    binary = ''.join(map(constLenBin,bytearray(data, 'utf-8'))) # 将需要被隐藏的字符串转换成二进制字符串
    if len(binary) > len(image.getdata()) * 4:  # 如果不可能编码全部数据， 抛出异常
        raise Exception("Error: Can't encode more than " + len(evenImage.getdata()) * 4 + " bits in this
image. ")
    encodedPixels = [(r+int(binary[index*4+0]),g+int(binary[index*4+1]),b+int(binary[index*4+2]),t+int(bi
nary[index*4+3])) if index*4 < len(binary) else (r,g,b,t) for index,(r,g,b,t) in enumerate(list(evenImage
.getdata()))] # 将 binary 中的二进制字符串信息编码进像素里
    encodedImage = Image.new(evenImage.mode, evenImage.size)  # 创建新图片以存放编码后的像素
    encodedImage.putdata(encodedPixels)  # 添加编码后的数据
    return encodedImage

"""
从二进制字符串转为 UTF-8 字符串
"""
def binaryToString(binary):
    index = 0
    string = []
    rec = lambda x, i: x[2:8] + (rec(x[8:], i-1) if i > 1 else '') if x else ''
    # rec = lambda x, i: x and (x[2:8] + (i > 1 and rec(x[8:], i-1) or '')) or ''
    fun = lambda x, i: x[i+1:8] + rec(x[8:], i-1)
    while index + 1 < len(binary):
        chartype = binary[index:].index('0') # 存放字符所占字节数，一个字节的字符会存为 0
        length = chartype*8 if chartype else 8
        string.append(chr(int(fun(binary[index:index+length],chartype),2)))
        index += length
    return ''.join(string)

"""
解码隐藏数据
"""
def decodeImage(image):
    pixels = list(image.getdata())  # 获得像素列表
    binary = ''.join([str(int(r>>1<<1!=r))+str(int(g>>1<<1!=g))+str(int(b>>1<<1!=b))+str(int(t>>1<<1!=t))
for (r,g,b,t) in pixels]) # 提取图片中所有最低有效位中的数据
    # 找到数据截止处的索引
    locationDoubleNull = binary.find('0000000000000000')
    endIndex = locationDoubleNull+(8-(locationDoubleNull % 8)) if locationDoubleNull%8 != 0 else location
DoubleNull
    data = binaryToString(binary[0:endIndex])
    return data

encodeDataInImage(Image.open("coffee.png"), '你好世界，Hello world!').save('encodeImage.png')
print(decodeImage(Image.open("encodeImage.png")))
```