

Misc专题Lab1

3220102732-周伟战-Misc专题1

基础部分

Task1



将所给的图丢进010 Editor发现明显有附带文件


[illegible]

然后将灰色部分从FF D8开始的文件单独放在A2.jpg中，重新打开发现了下面这张图



很明显，flag出来了

AAA{the_true_fans_fans_nmb_-1s!}



Description

真正的CTF选手，就算我什么都不说，也能找出flag来。

Link 0

Hint >

Your Answer

AAA{the_true_fans_fans_nmb_-1s!}

Solved

Completed

Clapeysron traveler zuhxs icefires Sleepy saltyfish 850 ISLAB Pale_Shadow...

Task2

利用WSL中的wget url命令，获取html的源码，然后发现了一张jpg图像

```
~ 22:14:59
> ls
HelloWorld  capstone  miaomiaomiao_2290CB13158C1F7B821EF107B56999C9.html  powerlevel10k

~ 22:15:00
> cat miaomiaomiao_2290CB13158C1F7B821EF107B56999C9.html
<html>
<body>
  <script>
    for (;;) {
      alert('Miao~');
    }
  </script>
  
</body>
</html>

~ 22:15:53
> █
```



这个的最终打开的图片(可恶的miaomiaomiao找了好久)

然后在010Editor中发现了key:m1a0@888

在发现给了key后，利用steghide打开这个文件，发现了secret.txt

```

~ 23:57:02
> steghide extract -sf miao~870F6C667A6CDC0D1F533859E72C48E0.jpg -p m1a0@888
wrote extracted data to "secret_file.txt".

~ 23:57:08
> ls
HelloWorld  miaomiaomiao_2290CB13158C1F7B821EF107B56999C9.html  output_image.jpg  secret_file.txt
bin         miao~870F6C667A6CDC0D1F533859E72C48E0.jpg           output_image.tiff  zwz123
capstone    output                                                  powerlevel10k

~ 23:58:20
> cat secret_file.txt
0100000101000001010000010111101101000100001100000101111010110010011000001110101010111101001100001100010
11010110110010101011111010100110111010001100101001110010100100000110001011001000110010101011110100110100
110001011000010011000001111101

~ 23:58:26
> █

```

然后利用给的二进制转化成ascii码获得flag

AAA{D0_Y0u_L1ke_Ste9H1de_M1a0}

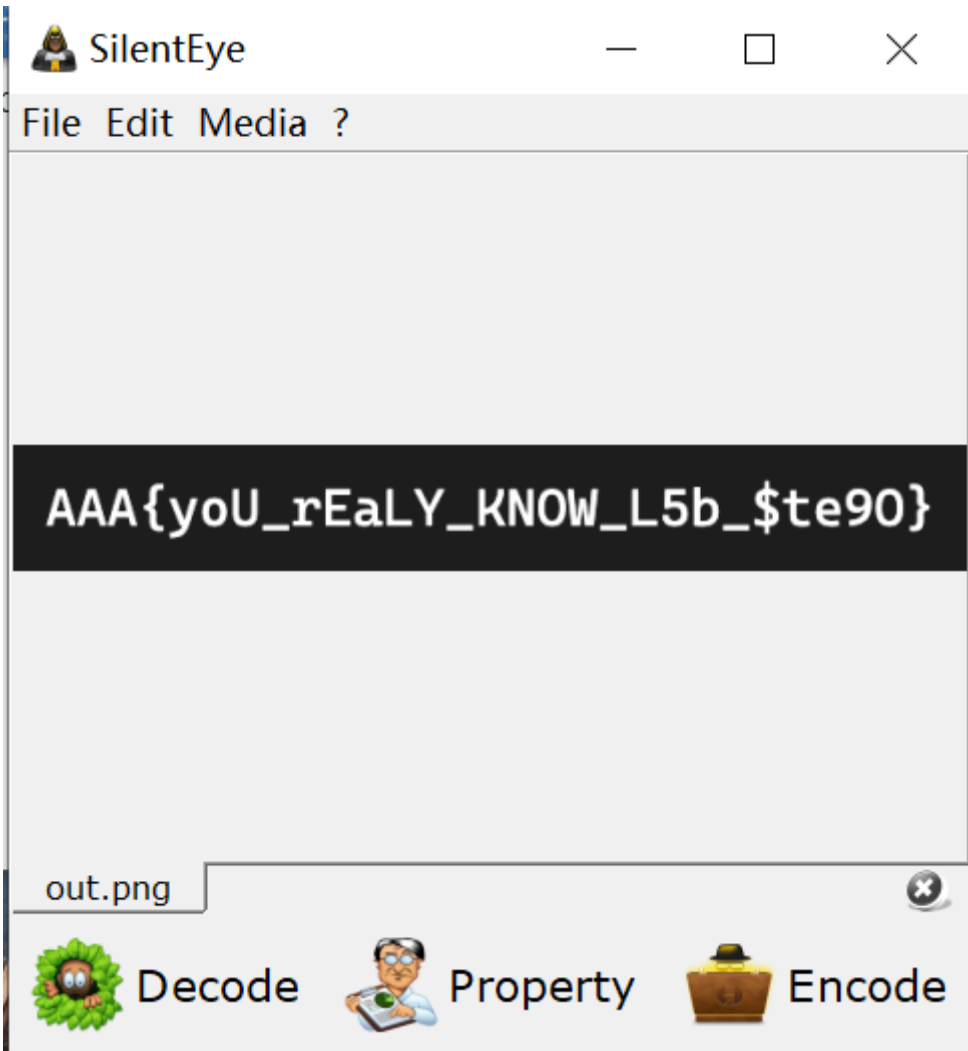
Task3

LSB隐写利用stegsolve



利用zsteg工具 发现b1,bgr,lsb,xy文件是一个隐藏的PNG image, 然后利用命令行
zsteg -e b1,bgr,lsb,xy p5.png -> out.png 输出为out.png文件, 于是利用silenteye打开这个png文件

```
/mnt/d/ZJU/【CS】/Capture The Flag/lab专题/misc1 15:11:37
> zsteg P5.png
imagedata      .. text: "0*\\"VPH.$"
b1,rgb,msb,xy  .. file: OpenPGP Secret Key
b1,bgr,lsb,xy  .. file: PNG image data, 583 x 78, 8-bit/color RGB, non-interlaced
b2,r,msb,xy    .. text: "U}UUUWU}UWU"
b2,g,msb,xy    .. text: "UwU_}uww"
b2,b,msb,xy    .. text: "]u_UWUu}"
b2,rgb,msb,xy  .. text: "W}}uU_WwUUUUUUu"
b2,bgr,msb,xy  .. text: "wUwUUUUUUU"
b3,b,msb,xy    .. file: OpenPGP Secret Key
b3,rgb,msb,xy  .. file: OpenPGP Public Key
b4,r,lsb,xy    .. text: "\"#2#\"\"2\"\"2\"\"\"3\"\"#\"\"\"\"\"2\"\"\"#2\"\"2\"\"\"#2\"##\"\"
\"22\"##\"##3\"\"2\"\"#333\"#23\"\"\"\"2\"##\"3\"\"#\"#\"2332\"\"#3\"#\"\"2#32\"2\"32\"\"2#\"\"\"32\"22\"
\"2\"\"2\"\"#3#2\"\"#\"\"##\"\"\"##32\"\"\"3##32\"22233###2\"\"#3#3\"\"3\"33333223322233##33###3#\"\"\"
222\"\"\"3#\"\"\"#3\"##33#323\"#3##\"\"\"33\"2\"\"2\"3#\"\"#2#\"\"\"\"\"\"2322"
b4,r,msb,xy    .. text: "DDLDDLDDDD"
b4,g,lsb,xy    .. text: "gvfgvvvggfffffffwvffvffffffvffffffgffffffgffwvffffgwgfgvvffvfvfvvgfgvggffvfvffwfg"
```



得到了flag:AAA{yoU_rEaLy_KNOW_L5b_\$te90}

选做部分

Task1

对于png中的PLTE是作为可选的数据块，如果在IHDR中的第10个字节处是03，那么就需要调用PLTE调色盘，他里面记录了各个像素块所对应的RGB颜色，例如，一段。调色板的长度一定是3的倍数，不然就会显示Error。

对于EZStego隐写的主要几个步骤：

Step1: 将调色板的颜色亮度依次排序，利用公式 $Y=0.299R+0.587G+0.114B$ 可以求得各像素块的亮度。

Step2: 为每个调色板上的颜色分配一个索引序号

Step3: 将调色板的图像像素内容使用LSB隐写代替，并将图像像素索引值改为新的亮度索引值（改变的只是索引值，像素本身的颜色是不变的）。

Step4: 用奇数序号表示嵌入秘密比特1，用偶数序号表示嵌入秘密比特0。

e.g.EZStego隐写举例

#Y_index_inverse是将按照亮度排序之后的索引值重新放回对应

[3 0 6 4] 和 [7 0 0 4] 都表示像素的序号而并非索引值

```
5  """
6  假设调色板索引为
7      0 1 2 3 4 5 6 7
8  假设亮度序号(Y_index)为
9  index:  0 1 2 3 4 5 6 7
10         2 5 4 1 7 3 6 0
11  则
12  # Y_index_inverse
13  index:  0 1 2 3 4 5 6 7
14         7 3 0 5 2 1 6 4
15
16  # 例子
17  载体  [3 0 6 4] ; 待嵌入信息 0110
18  嵌入: [3 0 6 4]=>[5 7 6 2]=>(by 0110) [4 7 7 2]=>[7 0 0 4]
19  结果: [3 0 6 4]=>(by 0110)=>[7 0 0 4]
20  提取: [7 0 0 4]=>[4 7 7 2]=>0110
21  """
```

总体思路是：

原本的PLTE中的像素亮度索引应该是一一对应的，因为在按照亮度降序后，是每个序号的像素颜色块

都对应一个索引值，但是就如上面举的例子来看 [3 0 6 4] 表示的是哪几个位置的像素块需要加密，通过将隐写内容放进他们所对应的索引值 [5 7 6 2]，使得其索引值发生改变，变成了 [4 7 7 2]，然后 [7 0 0 4] 是这个索引值所对应的像素块序号。

然后，已知针对像素的处理是一行一行进行的，但是PIL图片的排列是先列再行，(x,y)表示的是x列，y行。那么解密的时候就是一列一列的处理。对列开始循环进行像素的处理。

自以为对于EZStego隐写的处理是先在PIL图像中按列循环,寻找到 80 76 84 69 这四个十六进制字符,类似于寻找到了一个sign,是PLTE部分的heading,然后与在下一个Heading 73 68 65 84 之前的部分就是相关调色板中的相关像素的存放信息.(通过 `img.palette` 可直接访问),对于处理过后的图片,在PLTE部分的相关RGB值是不变的,变的是他们的索引值,那么在 IDAT 部分需要调用颜色的时候,所用的索引值就发生了变化.所以要进行处理,就主要看 IDAT 部分里面的索引值规律.

代码尽量写了,但没完全写完,只写了获取了PLTE部分的调色板的值,并且按照亮度升序的代码,定义了 `Png_Steg` 的一个类

```

from PIL import Image
import numpy as np
import math
import zlib

class Png_Steg:
#这里定义了一个类 Png_Steg()
    def __init__(self):
        #初始化
        self.im = None

    def load_png(self, png_file):
        #将png_file的数据传给self.im之中
        self.im=Image.open(png_file)
        self._load_palette()
        self._sort_palette()
        self._load_palette_data()
        self.available_info_len = len(self.palette_data)

    def _load_palette(self):
        #获得PLTE中的RGB数据,因为每个像素的RGB值都是3份一组,因此只需要按组来获取
        self.palette = []
        palette = self.im.palette.palette
        for i in range(0, len(palette), 3):
            self.palette.append((palette[3 * i], palette[3 * i + 1], palette[3 * i + 2]))

    def _sort_palette(self):
        #对于调色板子中的亮度进行排序
        #Y=0.299R+0.587G+0.114B
        Y=[]#一个空列表,用来存放所需要的亮度值
        palette = self.im.palette.palette
        for x in range(0, len(self.palette), 3):
            P=0.299*palette[x]+0.587*palette[x+1]+0.114*palette[x+2]
            Y.append= P
        #此时获得了各像素块的颜色亮度大小
        self.Y_index=np.argsort(Y)
        self.inverse_index=[]*256
        for _ in range(len(Y)):
            self.inverse_index[self.Y_index(x)]=x

    def _load_palette_data(self):
        self.palette_data = self.im.getpalette()

if __name__ == '__main__':
    img=Png_Steg()

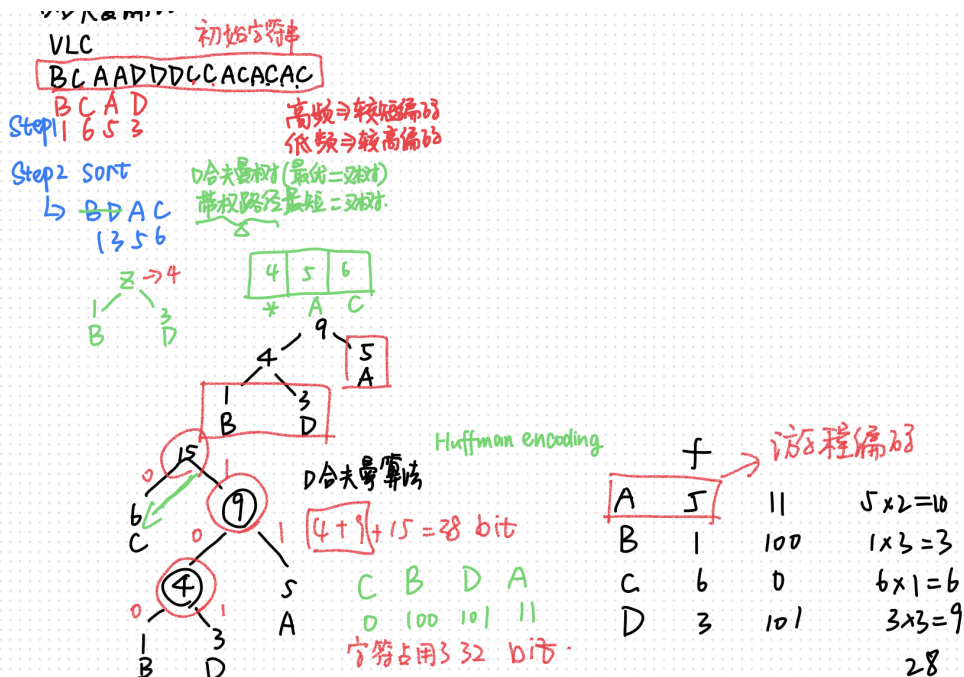
```


Task2

Task3

虽然这题的难度还不是我目前水平能解的,但在学JPEG编码的时候自认为较为详细的把Huffman编码学了一遍.觉得写下来还是比较有意义的.也是我目前能做的

Huffman编码是一种较高压缩效率的编码方式,它主要依靠的是字符出现的频率来作为压缩的依据.Huffman Tree能比较好的帮助理解



- Huffman Tree(最优二叉树)的好处就是低频的字符利用较长的编码,而高频的字符利用较短的编码,这样可以尽最大程度的降低编码所需要的字节数.

上图是Huffman编码的大致过程:

Step1:统计某字符串中各字符出现的频率,并且进行升序排列.就比如 {B:1,D:3,A:5,C:6} .

Step2:然后从小往大的画Tree,从 B:1,D:3 开始,二者就是两个子叶,他们合起来所需要的字节数为4,可以作为一个子节点,并且与 A:5 组成了新的层次,然后二者又能形成了一个新的节点 9 ,最后和 c:6 组成了最终的 root 通过层次的高低,也很能直观的看到,越底层的叶子是字符出现频率越低的,那么在进行Huffman编码的时候,从root出发,要到达Leaf(k),它的编码就稍越长.

Step3:左0右1的原则,从root出发,往下画,所经过的路径合起来就是某个子叶的Huffman编码.

Step4:按照Huffman编码, BCAADDDCCACACAC 这个字符串原本需要的是120bit(15×8),但在经过处理过后,只需要 $28+32+15=75$ bit,是一种压缩效率很高的编码方式,后续的编码只需要变成 1000111110110110100110110110 然后提供Huffman Tree的结构,就能实现解压的过程.

而这道题的出题方式是在Huffman Tree中插入了一些没有出现过的字符,那么所提供的Huffman Tree结构肯定是有冗余的.