

Esteganografía Aplicado a Imágenes PNG

Richard Villca Apaza

Universidad Autónoma del Beni

Oruro, Bolivia

richard.villca@ieee.org

a.k.a. @SixP4ck3r

Abstract— Nowadays there is great interest in the branches of cryptography or information security applying steganography that helps to understand the methods used to hide information. One of the strengths of steganography is emerging as a science to be studied for the transfer of files with higher levels of privacy, using known and common channels such as audio and images.

Keywords: esteganografía, criptografía, seguridad informática, mensajes.

I. INTRODUCCIÓN

El estudio de la esteganografía usado en imágenes, audios ha tomado relevancia en estos últimos años en las áreas de criptografía o seguridad informática. En la presente investigación se intentará tocar tópicos que infieren en el proceso de la Esteganografía aplicado a Imágenes PNG, desde la estructura de una imagen, composición, algoritmo para el ocultamiento de la información en la Imagen PNG y finalmente una herramienta en lenguaje de programación Python que nos ayudará acelerar el proceso de ocultar información en la Imagen y también como recuperarla.

II. DESARROLLO

Esteganografía

La esteganografía estudia diferentes técnicas para la ocultación de datos en otros objetos, conocidos como objetos portadores. Actualmente, estos objetos portadores suelen ser medios digitales, como por ejemplo imágenes, videos o archivos de sonido. No obstante, sin lugar a dudas, el medio más utilizado en la actualidad son las imágenes, por su amplia difusión en Internet[1].

Conoceremos la estructura de una imagen PNG y los métodos para el proceso de ocultar información en una Imagen.

Estructura de una Imagen

El formato PNG proporciona un estándar portátil, legalmente no comprendido, bien comprimido, bien comprimido, bien especificado para archivos de imagen de mapa de bits sin pérdida[5].



Figura 1: Imagen PNG de 24 bit

Una imagen está compuesta por miles de pixeles, así mismo un pixel está formado por tres canales de colores primarios

denominados RGB (Rojo, Verde, Azul), del inglés R=Red, G=Green y B=Azul cuyos valores van desde el 0 hasta 255.

Por ejemplo el valor 255,255,255 nos daría como resultado un color Blanco y por lo contrario un 0,0,0 daría como resultado el color negro.

Cada pixel puede almacenar 3 bytes, uno por cada canal de color, sobre esta base podemos deducir que cada nivel de color ocupa un byte, por ejemplo si su valor es 0, el byte sería 00000000 y si es 1, 00000001.

En esta sección se especifican temas detallados que forman parte de un título principal, como el de “Desarrollo de Contenidos”.

LSB y MSB

LSB (Least Significant Bit) también conocido como el bit menos significativo es la posición del bit en un número binario que tiene el menor valor.

El bit menos significativo (LSB) es una de las técnicas de incrustación más comunes. El bit menos significativo es el menor valor en un número binario. En el algoritmo LSB, los datos están ocultos en los bits menos significativos de la imagen de la cubierta, WFQHHICH no se notifica cuando se ve con el ojo humano. El bit más significativo (también llamado bit de orden superior) es la posición de bits en un número binario que tiene el mayor valor[2].

Si aplicamos el cambio el último bit por 0 nos dará un 11111110 convertido a decimal tendremos 254, de 255 la diferencia no es mucho.

MSB (Most Significant Bit) también conocido como el bit más significativo que de acuerdo a su posición tiene el mayor valor.

Si aplicamos la misma lógica del bit más significativo por ejemplo 11111111 por 01111111 al llevarlo a decimal tendremos el número 127; de 255 pasó a 127 la diferencia es muy grande es por eso que se usa el bit menos significativo para estos procesos.

Aplicando LSB

Como hemos observado que con el LSB pasó de 255 a 254, el valor siempre será en 1 puede aumentar o disminuir. Nuestro proceso será descomponer en cadena, convertir en bits y esconder un bit por cada nivel de color.

Vamos a ocultar el texto “bo” y sus valores en ascii son:

b = 98

o = 11

Ahora esos valores los convertimos a binario:

98 = 01100010

11 = 01101111

Tenemos estos píxeles en el orden R, G, B. Sin alterar se verían así:

1. (90, 40, 60)
2. (16, 40, 60)
3. (90, 60, 50)
4. (32, 40, 60)
5. (90, 22, 28)
6. (40, 72, 82)

Y la representación binaria:

1. (01011010, 00101000, 00111100)
2. (00110000, 00101000, 00111100)
3. (01011010, 00111100, 00110010)
4. (00100000, 00101000, 00111100)
5. (01011010, 00111110, 01001110)
6. (00101000, 01001000, 01010010)

Paso seguido alteramos bytes por cada bit

Por cada bit de "bo" vamos a proceder a intercambiar un byte de color en los píxeles.

"bo" representan en decimal el 98 y 111, en binario **01100010** y **01101111**

Estos valores en binario **01100010** y **01101111** que representan a nuestra información a ocultar "bo" vamos a proceder a alterar en cada posición de R, G, B.

1. (0101101**0**, 0010100**1**, 0011110**1**)
2. (0011000**0**, 0010100**0**, 0011110**0**)
3. (0101101**1**, 0011110**0**, 0011001**0**)
4. (0010000**1**, 0010100**1**, 0011110**0**)
5. (0101101**1**, 0011111**1**, 0100111**1**)
6. (0010100**1**, 01001000, 01010010)

Los últimos 2 niveles del último píxel vamos a dejar en sin efectuar ningún cambio, por que ya hemos afectado con **01100010** y **01101111**

Una vez terminado de alterar procedemos a convertir a decimal se verían así:

1. (90, 41, 61)
2. (48, 40, 60)
3. (91, 60, 50)
4. (33, 41, 60)
5. (91, 63, 79)
6. (41, 72, 82)

y los píxeles originales eran estos:

1. (90, 40, 60)
2. (16, 40, 60)
3. (90, 60, 50)
4. (32, 40, 60)
5. (90, 22, 28)
6. (40, 72, 82)

Observando a nivel general se puede ver que hubo un incremento de $n+1$ y $n-1$, como por ejemplo en el primer nivel:

Pixel original: (90, 41, 61)

Pixel alterado: (90, 40, 60)

Esto traduciendo como colores nos daría el resultado siguiente:



Figura 2: Color original y color alterado

Como se puede observar en ambas imágenes, la imagen con los píxeles originales y la imagen con los píxeles modificados no existe una diferencia significativa a la visión del ojo humano.

Python

Python cuenta con facilidades para la programación orientada a objetos, imperativa y funcional, por lo que se considera un lenguaje multi-paradigmas. Fue basado en el lenguaje ABC y se dice que fue influenciado por otros como C, Algol 60, Modula-3 e Icon según su propio autor[3].

Utilizaremos Python para automatizar el proceso de ocultar y recuperar del texto de la imagen.

Librería para Procesar Imágenes

Para la lectura y escritura de imágenes Python cuenta con varios paquetes que son los siguientes: Matplotlib, PIL (Python Image Library), Scikit Image, Scipy y OpenCV. Cada uno de estos paquetes carga la imagen de una manera diferente ya que en algunos casos se ve como una clase imagen y en otros como un arreglo de Numpy. Dependiendo del tipo de manipulación que se le dé será el paquete para utilizar. De igual manera, al desplegar la imagen o conjunto de imágenes se hará con el visor del sistema operativo, el visor particular del módulo o con la ayuda de otro paquete auxiliar[4].

Ocultando Texto

Iniciamos leyendo los píxeles como un arreglo bidimensional con el ancho y el alto, luego leemos las posiciones x,y

```
Terminal - six@ub: ~/tools/master
Archivo Editar Ver Terminal Pestañas Ayuda
six@ub:~/tools/master$ python3 ocultar.py --text "Bolivia" --image bolivia.png
Tool orientada a ocultar texto en una Imagen
Desarrollado solo con fines orientativos y conceptuales
Desarrollado por @SixP4ck3r

Iniciando...
El texto fue escrito correctamente!
six@ub:~/tools/master$
```

Figura 3: ocultar.py ocultando texto "Bolivia" y una imagen

Para proceder a ocultar texto en la imagen necesitamos un texto y una imagen PNG

```
def start(message, image_input):
    print("Iniciando...".format(message))
    image = Image.open(image_input)
    pixels = image.load()

    image_size = image.size
    image_x = image_size[0]
    image_y = image_size[1]

    list_all_bits = all_bits(message)
    bit_write_count = 0
    image_length = len(list_all_bits)

    for x in range(image_x):
        for y in range(image_y):
            if bit_write_count < image_length:
                pixel = pixels[x, y]

                base_r = pixel[0]
                base_g = pixel[1]
                base_b = pixel[2]
```

Luego de leer los colores R, G, B, para luego intercambiar los bits LSB

```
def get_binary(x):
    return bin(x)[2:].zfill(8)

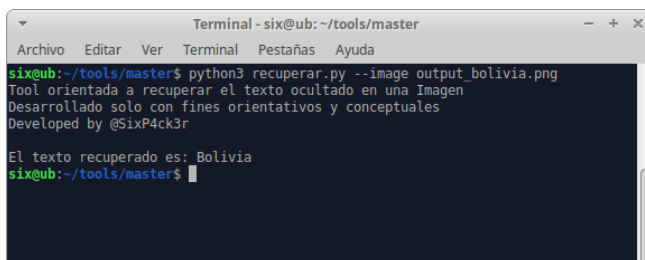
def change_lsb(byte, new):
    return byte[:-1] + str(new)

def change_color(color_base, bit):
    binary_color = get_binary(color_base)
    color_updated = change_lsb(binary_color, bit)
    return int(color_updated, 2)

def all_bits(text):
    list_all_bits = []
    for char in text:
        in_ascii = ord(char)
        in_binary = get_binary(in_ascii)
        for bit in in_binary:
            list_all_bits.append(bit)
    for bit in end_chars_base:
        list_all_bits.append(bit)
    return list_all_bits
```

Recuperando Texto

El proceso de recuperación del texto oculto en la imagen leemos LSB de cada nivel de píxel por cada byte, obtenemos su carácter y lo concatenamos en una variable del texto final a mostrar.



```
Terminal - six@ub: ~/tools/master
Archivo Editar Ver Terminal Pestañas Ayuda
six@ub:~/tools/master$ python3 recuperar.py --image output_bolivia.png
Tool orientada a recuperar el texto oculto en una Imagen
Desarrollado solo con fines orientativos y conceptuales
Developed by @SixP4ck3r

El texto recuperado es: Bolivia
six@ub:~/tools/master$
```

Figura 4: recuperar.py recuperando el texto desde la imagen

```
def start(image):
    imagen = Image.open(image)
    pixels = imagen.load()

    image_size = imagen.size
    image_x = image_size[0]
    image_y = image_size[1]

    byte = ""
    hidden_data = ""

    for x in range(image_x):
        for y in range(image_y):
            pixel = pixels[x, y]

            base_r = pixel[0]
            base_g = pixel[1]
            base_b = pixel[2]
```

Imagen de Salida

La imagen de salida mediante un análisis se ha cambiado algunos segmentos de RGB que son indetectables ante la observación del ojo humano.

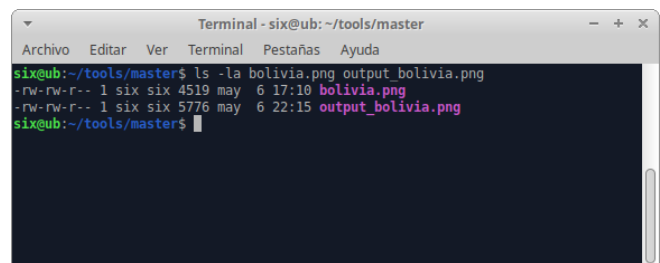


Figura 5: Diferencias entre la imagen original y la imagen con texto oculto

Como se observa en la captura la imagen original "bolivia.png" pesa 4519 bytes y "output_bolivia.png" tiene un peso de 5776 bytes, podemos inferir que creció a 1257 bytes.

El tamaño de la imagen se ha visto incrementada al 28% respecto a la imagen original y sin pérdida calidad ni compresión usadas en tecnologías de imágenes como JPG.

III. CONCLUSIÓN

- Con técnicas de esteganografía se puede ocultar texto en una imagen y sin mucha pérdida de la calidad.
- El ocultamiento de la información mediante la esteganografía sigue siendo estudiado para optimizar y transferir información.
- El texto oculto muchas veces solo está en texto plano, se recomienda tal vez ocultar un texto cifrado.
- La esteganografía tendrá mayor éxito cuando se pueda optimizar la forma en que un texto es embebido en audio, imagen o cualquier otro fichero.

IV. HERRAMIENTAS Y CÓDIGO FUENTE

Se proporciona el código fuente en Github para reproducir con mayor comodidad <https://github.com/rithchard/esteganografia>

V. REFERENCIAS

- [1] Daniel Lerch-Hostalot, David Megias, “Esteganografía en zonas ruidosas de la imagen”, Universitat Oberta de Catalunya, vol. 1, no. 1, pp. 1-1, feb. 2014.
- [2] AKINOLA, Solomon O.; OLATIDOYE, Adebanks A. On the image quality and encoding times of LSB, MSB and combined LSB-MSB steganography algorithms using digital images. International Journal of Computer Science & Information Technology (IJCSIT), 2015, vol. 7, no 4, p. 79-91.
- [3] CHALLENGER-PÉREZ, Ivet; DÍAZ-RICARDO, Yanet; BECERRA-GARCÍA, Roberto Antonio. El lenguaje de programación Python. Ciencias Holguín, 2014, vol. 20, no 2, p. 1-13.
- [4] MORENO FERNÁNDEZ, Samuel. Herramienta de Reconocimiento de Imágenes en Python. 2020.
- [5] BOUTELL, T. RFC2083: PNG (Portable Network Graphics) Specification Version 1.0. 1997.