

Wodax: Texto entre las sombras

Joan Puigcerver Pérez

joapuipe@gmail.com

Texto publicado bajo los términos de la Licencia de Documentación Libre de GNU.

7 de febrero de 2009

Índice

1. Introducción	2
1.1. Esteganografía	2
1.2. Esteganografía y criptografía	2
1.3. Conceptos fundamentales	3
1.3.1. Vocabulario	3
1.3.2. Imágenes digitales	3
1.3.3. Caracteres digitales	4
2. Funcionamiento	5
2.1. Ocultando caracteres	5
2.2. Extrayendo caracteres	6
2.3. Efectividad y eficiencia	7
2.4. Ataques	8
2.4.1. Fuerza bruta	8
2.4.2. Comparación bit a bit	9
2.4.3. Reescribir la imagen	9
2.5. Mejoras de seguridad	9
2.5.1. Entropía	9
2.5.2. Cifrado	10
2.6. Restricciones	11
2.6.1. Tamaño de la imagen en relación a la del texto	11
2.6.2. Longitud de la contraseña	11
2.6.3. Imágenes JPEG	11
3. Sobre el proyecto en concreto	12
3.1. Historia	12
3.2. Instalación y uso	12
3.2.1. Instalación	12
3.2.2. Uso	13
3.3. Librerías y código fuente	13
3.4. Licencia	13
3.4.1. Del programa	13
3.4.2. De la documentación	13

1. Introducción

1.1. Esteganografía

El término esteganografía proviene de las palabras griegas *steganos* (oculto) y *graphos* (escritura) y es en la antigua Grecia donde aparece por primera vez de la mano de Heródoto y su *Historia*[1].

Aquí se puede leer un fragmento en el cual Histieo utiliza un método esteganográfico primitivo para rebelarse contra los persas.

Pues como Histieo hubiese querido prevenir a su deudo que convenía rebelarse, y no hallando medio seguro para ponerle el aviso por cuanto estaban los caminos tomados de parte del rey, en tal apuro había rasurado a navaja la cabeza del criado que tenía de mayor satisfacción, habíale marcado en ella con los puntos y letras que le pareció, esperó después a que le volviera a crecer el cabello, y crecido ya, habíalo despachado a Mileto sin más recado que decirle de palabra que puesto en Mileto pidiera de su parte a Aristágoras que, cortándole a navaja el pelo, le mirara la cabeza. Las notas grabadas en ella significaban a Aristágoras, como dije, que se levantase contra el persa.

Pero no fue hasta el siglo XVI cuando Johannes Trithemius puso el nombre de *Steganographia* a la técnica de ocultar texto, en su libro homónimo[8].

Sin embargo, al igual que sucedió con la criptografía (como por ejemplo, la famosa máquina Enigma empleada por el Tercer Reich) y la mayoría de los avances tecnológicos, la esteganografía se potenció básicamente como arma de guerra. Y por ejemplo, también en la II Guerra Mundial, se insertaron microfilms con información oculta en los signos de puntuación de los mensajes. De esta forma, una típica carta de amor escrita por un soldado en el frente, contenía en realidad la información de los movimientos del ejército.

Ahora, en plena época digital, la esteganografía ya no usa herramientas tradicionales como la tinta invisible obtenida a partir del zumo de limón, sino que utiliza métodos digitales y tiene muchas más utilidades que las puramente bélicas, como por ejemplo las marcas de agua en documentos digitales para verificar la autenticidad de un documento (como fotografías, vídeos, documentos de texto, etc).

Incluso la Agencia Federal de Investigación (FBI) o la Agencia Nacional de Seguridad (NSA) de los Estados Unidos, la tienen en cuenta para defenderse de posibles ataques terroristas y han escrito varios documentos al respecto[2].

1.2. Esteganografía y criptografía

A menudo se confunde la esteganografía con la criptografía, cuando en realidad, a pesar de que sus orígenes y finalidades son semejantes, las dos técnicas son bien distintas.

La criptografía es la técnica de transformar el contenido de un mensaje legible en ilegible (cifrado) de forma que únicamente el receptor sea capaz de hacer el proceso inverso (descifrado).

La esteganografía sin embargo, es la técnica de ocultar el contenido de un mensaje de forma que únicamente el destinatario sepa de su existencia.

De esta forma, una técnica criptográfica esconde el contenido del mensaje, pero no el hecho de que existe un mensaje. Si un tercero detecta un mensaje cifrado no sabrá el contenido del mensaje, pero sin duda sabrá de la existencia del mensaje oculto en sí. Pero con una técnica esteganográfica, es el propio mensaje el que se oculta de forma que un tercero no sea consciente de la existencia de éste. Esta diferencia entre las dos técnicas se ve claramente en el problema de los prisioneros descrito por G. J. Simmons[6].

Imaginamos que hay dos prisioneros que quieren planear la huida de la cárcel, pero todos sus mensajes pasan primero por las manos de un guardián. ¿Cómo harían para comunicarse sin que el guardián sea

capaz de darse cuenta de sus intenciones? Evidentemente, no pueden simplemente cifrar el mensaje, pues aunque el guardián no sepa el contenido de este, el hecho de encontrar un mensaje tan extraño lo haría sospechar. Así pues, no es suficiente con una técnica criptográfica. Será necesaria una técnica esteganográfica para engañar al guardián con un falso contenido (contenido tapadera) y que pase por alto el verdadero mensaje.

1.3. Conceptos fundamentales

Antes de empezar a explicar el funcionamiento del programa, se explicaran algunos conceptos básicos sobre la representación de imágenes y caracteres digitales, y se presentará un vocabulario elemental sobre esteganografía.

1.3.1. Vocabulario

Documento tapadera Documento en el que se incrustará el mensaje que se pretende ocultar.

Bit menos significativo Es la posición de bit en un número binario que tiene el valor menor¹.

Píxel El píxel es la unidad mínima de información que forma una imagen digital.²

Modelo de color Un modelo de color permite representar los colores de forma numérica, utilizando típicamente tres o cuatro valores, llamados componentes cromáticos o canales. Existen distintos modelos de color como el RGB que utiliza los canales rojo (*Red*), verde (*Green*) y azul (*Blue*), el CMYK que utiliza los canales cian (*Cyan*), magenta (*Magenta*), amarillo (*Yellow*) y negro (*blacK*), o el tradicional RYB, utilizado por la comunidad artística a pesar de no ser un modelo correcto, que utiliza el rojo (*Red*), el amarillo (*Yellow*) y el azul (*Blue*)³.

Profundidad de color La profundidad de color hace referencia a la cantidad de bits con los que se codifica un píxel. La profundidad de color es un indicativo de la cantidad de colores diferentes que se pueden representar en un píxel. Sus unidades de medida son los bits por píxel (bpp) y las profundidades más comunes son 8 ($2^8 = 256$ colores), 16 ($2^{16} = 65536$) o 24 ($2^{24} = 16777216$)⁴.

Resolución de imagen La resolución de imagen indica la cantidad total de píxeles que contiene una imagen. Se calcula a partir del número de píxeles de anchura (*width*) y de altura (*height*) que tiene la imagen (*Resolución = Anchura × Altura*). En las imágenes capturadas por cámaras fotográficas, la resolución suele ser de millones de píxeles (Megapíxeles)⁵.

1.3.2. Imágenes digitales

Una imagen digital es la representación de una imagen utilizando bits. Existen dos tipos de imágenes (o gráficos) digitales, los gráficos basados en un mapa de bits (*raster graphics*) o los vectoriales (*vectorial graphics*). La diferencia está en que en los primeros, la imagen se representa como una matriz donde cada posición es ocupada por un píxel utilizando un modelo y una profundidad de color determinados y la segunda está formada por objetos geométricos (rectas, polígonos, etc.) definidos por atributos matemáticos de forma. A partir de aquí, siempre que se haga referencia a una imagen será a una imagen digital basada en un mapa de bits, utilizando el modelo de color RGB y con una profundidad de color de 24 bits (que son las más habituales a día de hoy).

¹http://en.wikipedia.org/wiki/Least_significant_bit

²<http://en.wikipedia.org/wiki/Pixel>

³http://en.wikipedia.org/wiki/Color_model

⁴http://en.wikipedia.org/wiki/Color_depth

⁵http://en.wikipedia.org/wiki/Image_resolution

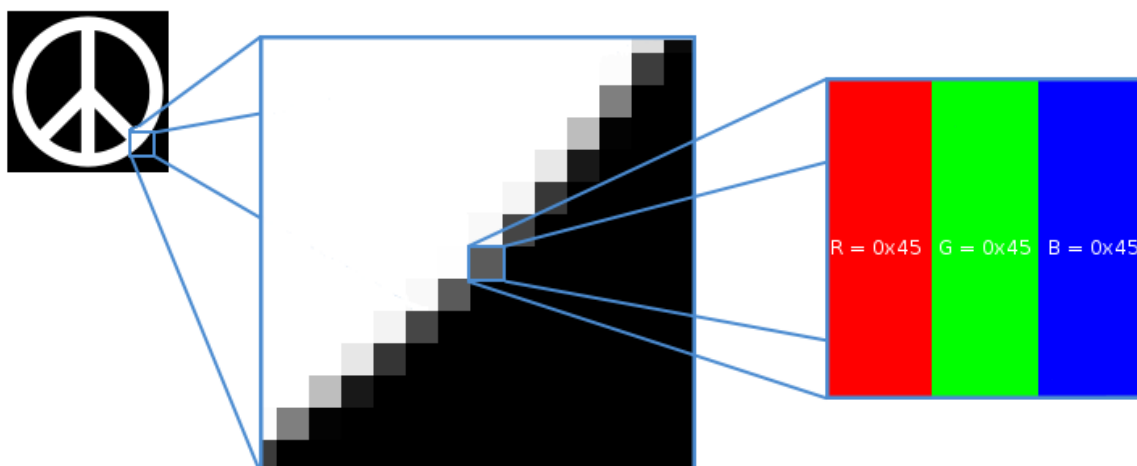


Figura 1: En la izquierda la imagen completa, en el centro los píxeles de una región y en la derecha los valores de los tres canales de un píxel.

Si se conoce la resolución de la imagen y la profundidad de color, se puede calcular el número de bits que ocupa la imagen y, por tanto, la cantidad de información contenida en esta. Por ejemplo, una imagen de 640×420 píxeles de resolución con una profundidad de color de 24 bits, tiene un total de $640 \times 420 \times 24 = 6451200 \text{ bits} = 6300 \text{ Kbits}$ (donde $1 \text{ Kbits} = 2^{10} \text{ bits}$)⁶.

1.3.3. Caracteres digitales

En una computadora los caracteres se representan con un valor numérico que depende de la codificación empleada. Existen varias codificaciones, la más antigua es la ASCII (*American Standard Code for Information Interchange*), pero actualmente se utilizan muchas más como las ISO-8859-{0-8}, las propias de Microsoft Windows (Windows-125{0-8}), o las diferentes variantes de la Unicode.

La diferencia fundamental entre cada una de estas es el valor que utilizan para representar un mismo carácter y el número de bits que utilizan para hacerlo. Tanto la codificación Windows-1252⁷ como la ISO-8859-15⁸ utilizan 8 bits (1 byte) para representar un carácter. Por lo tanto hay $2^8 = 256$ caracteres diferentes que se pueden representar utilizando cada una de estas codificaciones. Pero, por ejemplo, en la codificación Windows-1252, el carácter “€” se codifica con el valor decimal 128 y la ISO-8859-15 lo hace con el 164.

Por otro lado la codificación ASCII⁹ sólo utiliza 7 bits para representar cada carácter, de forma que hay $2^7 = 128$ caracteres representables en la codificación (aunque existen extensiones de 8 bits para representar caracteres con acentos que son muy utilizadas).

Estos 7 bits podrían parecer suficientes para representar todo el alfabeto latino y los dígitos arábigos, por ejemplo, pero debido a la internacionalización de los programas informáticos, esta limitación puede suponer un problema, puesto que hay muchos más alfabetos (japonés, cirílico, árabe, chino, etc.)¹⁰, así como símbolos matemáticos o lingüísticos y habría que utilizar una codificación distinta en cada país. Este problema intenta resolverlo el estándar Unicode¹¹ y sus codificaciones UTF-8 y UTF-16 (de 8 y 16

⁶Esto no significa que una imagen PNG, por ejemplo, vaya a ocupar 6300 Kbits en memoria. La mayoría de formatos de imagen tienen un espacio reservado para la cabecera donde se escriben algunos atributos (como la resolución, la profundidad de color, el modo de color, etc.).

⁷<http://www.microsoft.com/globaldev/reference/sbcs/1252.mspx>

⁸<http://anubis.dkuug.dk/JTC1/SC2/WG3/docs/n404.pdf>

⁹<http://www.unicode.org/charts/PDF/U000000.pdf>

¹⁰El japonés es un silabario y el chino un sistema ideográfico, no son alfabetos en realidad.

¹¹<http://www.unicode.org>

bits de longitud variable, respectivamente).

Sin embargo, en casi todos los sistemas de codificación, los primeros 128 valores coinciden con los de la codificación ASCII por motivos de compatibilidad.

2. Funcionamiento

Una vez entendidos algunos conceptos fundamentales, se explicará detalladamente el funcionamiento de Wodax. Existen varias estrategias para incrustar información en documentos tapadera, pero Wodax utiliza la más común e intuitiva: la inserción en el bit menos significativo (*Least Significant Bit Insertion*). Se pueden encontrar varios documentos que explican el funcionamiento de este método y de otros como el Enmascarado y filtrado, y el basado en Transformaciones ([4], [3]).

La idea es bien sencilla: se insertarán los bits del mensaje en los bits menos significativos de la imagen tapadera.

Para explicar el funcionamiento supondremos un mensaje que utiliza la codificación ASCII que se ocultará en una imagen PNG (el programa trabaja con este formato, en la sección 2.6.3 se explicará porque no lo hace con otros formatos como el JPEG).

2.1. Ocultando caracteres

Se explicará primero un pequeño ejemplo de como esconder un carácter en una pequeña imagen y a partir de este se profundizará para explicar el funcionamiento entero del programa. Imaginemos que disponemos de una imagen, que utiliza el modo de color RGB y tiene una profundidad de color de 24 bits, compuesta por los píxeles $P_0(R, G, B) = (45_{16}, 24_{16}, 52_{16})$ y $P_1(R, G, B) = (93_{16}, 0_{16}, 89_{16})$ y que queremos ocultar el carácter “A” en esta imagen.

En la sección 1.3.3 comentábamos que la codificación ASCII extendida utilizaba 8 bits para representar cada caracter. Así pues, el carácter “A” (valor 41_{16} en la tabla ASCII) se puede ver de la siguiente manera bit a bit¹².

C_3		C_2		C_1		C_0	
0	1	0	0	0	0	0	1

Cuadro 1: El carácter “A” visto bit a bit.

Los píxeles mencionados anteriormente se pueden ver de esta forma bit a bit.

P_0								
R	0	1	0	0	0	1	0	1
G	0	0	1	0	0	1	0	0
B	0	1	0	1	0	0	1	0

Cuadro 2: Píxel P_0 visto bit a bit.

P_1								
R	1	0	0	1	0	0	1	1
G	0	0	0	0	0	0	0	0
B	1	0	0	0	1	0	0	1

Cuadro 3: Píxel P_1 visto bit a bit.

¹²El bit más significativo ocupa la posición a la izquierda del byte.

El carácter es separado en parejas de bits como se ha indicado en la tabla 1 y se insertan en los dos bits menos significativos de cada canal, empezando por el rojo. Cuando todos los canales de un píxel han sido modificados, entonces se cambia de píxel y continúa incrustándose el carácter, empezando por el canal donde se había detenido.

A continuación se muestra como quedarían los píxeles del ejemplo anterior tras esconder el carácter “A”. Los bits que han sido modificados se han marcado en **negrita**.

P_0								
R	0	1	0	0	0	1	0	1
G	0	0	1	0	0	1	0	0
B	0	1	0	1	0	0	0	0

Cuadro 4: Píxel P_0 visto bit a bit después de ocultar el carácter “A”.

P_1								
R	1	0	0	1	0	0	0	1
G	0	0	0	0	0	0	0	0
B	1	0	0	0	1	0	0	1

Cuadro 5: Píxel P_1 visto bit a bit después de ocultar el carácter “A”.

Si ahora se incrustara un nuevo carácter, la inserción de este empezaría por el canal verde (G) del píxel P_1 hasta llenar este píxel y continuaría en el canal rojo (R) de un nuevo píxel P_2 que sería necesario.

De forma más genérica, este es el algoritmo que se utiliza para esconder un archivo de texto¹³.

Algorithm 1 Algoritmo para ocultar texto en una imagen.

Require: $Image = (p_1, p_2, p_3, \dots, p_n) / \forall p_i \in Pixel$

Require: $Text = (c_1, c_2, c_3, \dots, c_n) / \forall c_j \in [0, 255]$

$i \leftarrow \alpha$

$j \leftarrow 0$

$C \leftarrow Red$

repeat

for $k = 0$ to 3 **do**

$p_i(C) \leftarrow \lfloor p_i(C) \div 2^2 \rfloor + (\lfloor c_j \div 2^{2k} \rfloor \bmod 2^2)$

if $C = Blue$ **then**

$i \leftarrow \beta$

end if

$C \leftarrow (C + 1) \bmod 3$

end for

$j \leftarrow j + 1$

until $c_j = EOF$

2.2. Extrayendo caracteres

Una vez entendido como se incrusta un texto en una imagen, se puede comprender fácilmente como se extrae éste. Simplemente se debe hacer el proceso inverso. Recordemos cómo habían quedado los píxeles anteriores (las tablas 4 y 5).

¹³El significado de los valores α y β se explicarán en la sección 2.5.1, cuando se explicará el concepto de entropía en el programa. En este punto de la explicación, podéis considerar $\alpha = 0$ y $\beta = y + 1$.

Ahora se extraen los dos bits menos significativos de cada canal y se agrupan formando grupos de ocho bits. A partir del canal rojo (R) del píxel P_0 se obtiene la pareja de bits $C_0 = 01$, del canal verde (G) se obtienen los bits $C_1 = 00$ y del canal azul (B) los bits $C_2 = 00$. Como que ya se ha extraído la pareja de cada canal de P_0 , cambiamos de píxel y pasamos a P_1 para obtener la pareja de bits menos significativa del canal rojo, y se obtiene $C_3 = 01$.

C_3		C_2		C_1		C_0	
0	1	0	0	0	0	0	1

Cuadro 6: El carácter “A” extraído a partir de dos píxeles.

El programa, pues, recorre todos los píxeles de la imagen extrayendo la pareja menos significativa de bits de cada canal y agrupa estas parejas de cuatro en cuatro para formar caracteres de 8 bits. Vemos el algoritmo detalladamente.

Algorithm 2 Algoritmo para extraer un texto oculto en una imagen.

Require: $Image = (p_1, p_2, p_3, \dots, p_n) / \forall p_i \in Pixel$

Ensure: $Text = (c_1, c_2, c_3, \dots, c_n) / \forall c_j \in [0, 255]$

$i \leftarrow \alpha$

$j \leftarrow 0$

$C \leftarrow Red$

repeat

$c_j = 0$

for $k = 0$ to 3 **do**

$c_j \leftarrow (p_i(C) \text{ mód } 2^2) \times 2^{2k} + c_j$

if $C = Blue$ **then**

$i \leftarrow \beta$

end if

$C \leftarrow (C + 1) \text{ mód } 3$

end for

$j \leftarrow j + 1$

until $c_j = EOF$

2.3. Efectividad y eficiencia

Ya se ha explicado el funcionamiento del programa. La pregunta que se debe contestar ahora es si este método es efectivo (si resuelve nuestro problema) y eficiente (si lo hace en un tiempo y con un consumo de memoria adecuado).

El problema planteado, recordemos, era el de conseguir manipular una imagen para incrustar un mensaje de texto y que una tercera persona no pudiera darse cuenta de la existencia de este mensaje.

Si se tiene cualquier canal $C = XXXXXX_2 / X \in [0, 1]$, cuando se modifican sus dos bits menos significativos, los posibles valores son $C_0 = XXXXXX00_2$ ($\Delta C = 0$), $C_1 = XXXXXX01_2$ ($\Delta C = 1$), $C_2 = XXXXXX10_2$ ($\Delta C = 2$) o $C_3 = XXXXXX11_2$ ($\Delta C = 3$). Por lo tanto, para un canal, la variación máxima de su valor es de 3 unidades. Teniendo en cuenta que la diferencia entre el valor máximo y mínimo en un canal es 255 ($FF_{16} - 00_{16}$), la variación relativa del canal $\Delta CR \leq 0,011764706$. Y ya que cada píxel está formato por tres canales, la variación relativa de cada píxel será $\Delta PR \leq 0,035294118$.

A continuación, se presenta una demostración de la efectividad del programa con el ejemplo que se ha utilizado en el apartado anterior. La diferencia de color es inapreciable para el ojo humano.

(0x45, 0x24, 0x52)	(0x93, 0x00, 0x89)
(0x45, 0x24, 0x50)	(0x91, 0x00, 0x89)

Figura 2: Demostración de la efectividad del programa con el ejemplo anterior. Arriba antes y abajo tras modificar los píxeles.

La eficiencia es más sencilla de discutir incluso. Vemos que tanto en el algoritmo para esconder texto como en el de extraerlo, el bucle principal depende linealmente del número de caracteres del texto. Por lo tanto, el coste temporal de ambos algoritmos es $\Theta(C)$, donde C es el número de caracteres del texto.

Con respecto al coste espacial, los únicos datos que necesitan ser cargados en memoria son la imagen donde esconder o de donde obtener el texto y, por otra parte, el texto mismo. Entonces, el coste espacial para los dos algoritmos es $\Theta(C + P)$, donde C es el número de caracteres del texto y P el número de píxeles de la imagen.

2.4. Ataques

La esteganografía no es la técnica definitiva para comunicarse en secreto con una segunda persona, tiene ciertas restricciones y debilidades¹⁴. Estas son sólo algunas de las técnicas más sencillas que se pueden utilizar para invalidar esta técnica, pero es importante saber que existen otras más sofisticadas. Se pueden consultar algunos documentos por Internet como el de Niels Provos y Peter Honeyman[5].

2.4.1. Fuerza bruta

Un ataque por fuerza bruta es el ataque más “sencillo” que se puede aplicar a cualquier técnica esteganográfica o criptográfica, y en cierto modo no hay ninguna medida que pueda inhibirlo. Pero en la práctica este ataque, si el sistema está bien diseñado, resulta *computacionalmente*¹⁵ imposible de llevar a cabo.

Si el atacante conoce el píxel donde empieza a ocultarse el texto, podría fácilmente hacerse con el texto incrustado repitiendo el proceso descrito en la sección 2.2, ya que el algoritmo es conocido por todo el mundo¹⁶.

Si no conoce la posición inicial, pero los píxeles se modifican con un determinado incremento de posición entonces, probando con diferentes posiciones iniciales y diferentes incrementos de posición podría también hacerse con un mensaje escondido. El tiempo que tarde en hacerlo depende del tamaño de la imagen. Tendría que probar para cada posible posición inicial, todos los posibles incrementos de posición,

¹⁴La técnica que explota estas debilidades se denomina esteganoanálisis, al igual que el criptoanálisis a la criptografía.

¹⁵Se necesitarían miles de millones de años.

¹⁶La seguridad de un determinado algoritmo nunca puede estar basada en la ignorancia del atacante sobre dicho algoritmo, pues si falla este principio falla todo el sistema. Este enunciado se conoce como el Principio de Kerckhoff.

que irían desde 1 hasta el mismo tamaño de la imagen. Esto supondría un coste $O(P^2)$.

Si la imagen es suficientemente grande, esta tarea se haría en la práctica imposible. En la sección 2.5.1 se explicará una medida para mejorar la defensa contra este ataque.

2.4.2. Comparación bit a bit

Si el atacante cuenta con la imagen original donde se ha escondido el texto, obtenerlo es una tarea casi trivial. Podemos comparar bit a bit las dos imágenes (el original y la sospechosa de haber sido modificada) y conocer cuales son las parejas de bits que difieren y agruparlas con tal de obtener el mensaje completo.

Podrían haber parejas de bits que al modificarse quedasen igual que en la imagen original (en el ejemplo que se planteaba en la sección 2.1 esta situación se daba al esconder el par de bits C_1 y C_2), pero sin duda facilitaría mucho el trabajo a un ataque por fuerza bruta probando las combinaciones restantes.

2.4.3. Reescribir la imagen

Recordemos el problema descrito por Simmons en el que los dos prisioneros habían de evitar que el guardia se diese cuenta de que estaban planeando la huida. Hemos discutido una forma que tienen estos prisioneros de comunicarse secretamente bastante eficaz, ¿pero qué pasa si el guardia sospecha de ellos? Si el guardia llenase de *ruido* los mensajes que se intercambian los dos prisioneros, el contenido de estos llegaría alterado del uno al otro y sin sentido.

Si se sabe o se sospecha que una determinada imagen tiene texto oculto, únicamente llenando con bits aleatorios todas las parejas de bits menos significativos de los canales de cada píxel, se asegura que el contenido secreto de la imagen queda destruido. El atacante no conocerá el contenido del mensaje, pero el destinatario tampoco.

Este método fue propuesto por un estudiante de la Universidad de Northeastern con tal de implementarlo en los servidores de correo de forma que, cuando se enviara algún determinado tipo de archivo (imágenes, música, vídeos, etc.), se aplicara el método, destruyendo así la posible información escondida <http://spectrum.ieee.org/aug08/6593>.

2.5. Mejoras de seguridad

Tal y como se había comentado en el anterior apartado, hay ciertas medidas que se pueden aplicar para hacer más seguro el programa.

2.5.1. Entropía

Esta es una medida que se implementó para intentar hacer ineficaces los ataques por fuerza bruta. El concepto entropía en el programa proviene del concepto físico, la medida del desorden de un determinado sistema físico. En nuestro caso, podríamos entender la imagen como un sistema donde la entropía se refiere al desorden de los píxeles modificados en la imagen. Así, si los píxeles se modifican de forma contigua o siguiendo una determinada secuencia lógica, la imagen tiene un grado de entropía bajo, pero si por el contrario los píxeles se modifican aleatoriamente, podemos decir que la imagen tiene un grado de entropía elevado.

Evidentemente, no podemos modificar los píxeles completamente de forma aleatoria puesto que se necesita determinar cuales son los píxeles que se han modificado con tal de poder extraer la información posteriormente. Por esta razón se utiliza un generador de números pseudo-aleatorios.

Una generador de este tipo genera una secuencia de números a partir de una determinada “semilla”. Los números generados serán diferentes para cada semilla, pero siempre serán los mismos para una misma semilla.

En Wodax la semilla se obtiene al aplicarle una función *hash* de 32 bits a la contraseña que el usuario indica (esta contraseña también servirá en un futuro para cifrar el mensaje antes de esconderlo en la imagen). Aunque con una función *hash* de 32 bits hay numerosas colisiones¹⁷, esto no tiene “demasiada”¹⁸ importancia puesto que sirve como barrera para dificultar los ataques (pero no imposibilitarlos).

El procedimiento que se sigue para determinar la posición inicial a partir de donde empiezan a ocultarse los caracteres es la siguiente.

Algorithm 3 Selecciona el primer píxel a modificar

$X \leftarrow \text{Random}() \text{ mód } Width$
 $Y \leftarrow \text{Random}() \text{ mód } Height$

Donde *Width* y *Height* son la anchura y la altura en píxeles de la imagen y *Random()* es una función generadora de números pseudo-aleatorios (se usan las funciones `srand()` y `rand()` incluidas en la librería estándar de C).

Cuando se ha seleccionado un nuevo píxel porque ya se han modificado todos los canales del actual, se sigue el siguiente procedimiento.

Algorithm 4 Selecciona el nuevo píxel a modificar

$X' \leftarrow (X + (\text{Random}() \text{ mód } Entropy) + 1) \text{ mód } Width$
if $X' \leq X$ **then**
 $Y \leftarrow (Y + 1) \text{ mód } Height$
end if
 $X \leftarrow X'$

Los valores (X, Y) hacen referencia a la posición del píxel que se modifica. Ahora se entiende el significado de los valores α y β que aparecían en las secciones 2.1 y 2.2. El valor de α viene determinado por el resultado del primero de los dos algoritmos anteriores y el de β por el segundo.

Con estos dos procedimientos se consigue introducir ese grado de desorden que se pretendía. El valor de la entropía se obtiene a partir de la longitud de la contraseña, por tanto, cuanto más larga sea ésta mayor grado de desorden se consigue. Opcionalmente, también se puede forzar al programa a utilizar un valor determinado¹⁹.

2.5.2. Cifrado

En primero lugar decir que Wodax no tiene integrada una funcionalidad que permita cifrar el texto al mismo tiempo que se oculta en una imagen. El motivo es que no conozco con detalle un método de cifrado realmente seguro para implementarlo (dos posibles candidatos son el Tiger y el Whirlpool) y he preferido no implementar uno débil y dar una falsa sensación de seguridad donde no la hay. Wodax es un programa con una finalidad puramente didáctica, así que poco aprendería si simplemente utilizara una librería ya hecha para incorporar esta funcionalidad (durante un tiempo estuve pensando en usar *Crypto++*). Hecha esta aclaración, se ha decidido incluir este concepto dentro de la sección “Mejoras de seguridad” puesto que debe ser una característica fundamental y en la que ya se está trabajando para futuras versiones.

Volvamos a la situación de los prisioneros descrita por Simmons. Si por cualquier motivo, el guardia descubre que en los mensajes que se intercambian los prisioneros hay información oculta y la descubre, los dos prisioneros están perdidos.

¹⁷Una “colisión” *hash* ocurre cuando para dos llaves diferentes, la función *hash* vuelve el mismo valor.

¹⁸¡En realidad tiene muchísima importancia puesto que con una contraseña distinta a la usada para cifrar se podría descifrar el mensaje!

¹⁹Como se puede extraer a partir de su uso en los algoritmos, el valor de la entropía deberá ser mayor o igual a 1.

Lo que deben hacer los dos prisioneros es acordar un método de cifrado y de esta forma, si el guardia descubre que hay un contenido oculto en sus mensajes, antes de poder leerlo deberá descifrarlo (y si se elige un buen algoritmo de cifrado, es poco probable que eso ocurra). Esta es la idea: en lugar de esconder en la imagen el texto original, esconder el texto cifrado.

2.6. Restricciones

A continuación se explican algunas restricciones que tiene el programa o algunas características que se deberán tener en cuenta a la hora de ocultar mensajes.

2.6.1. Tamaño de la imagen en relación a la del texto

Como la posición donde empezar a ocultar el texto se elige aleatoriamente, esta podría ser la posición de uno de los últimos píxeles de la imagen. Si esto sucede, el programa continúa escribiendo en la imagen por el principio, de forma que la imagen forma una especie de círculo donde no hay ni principio ni final.

Por lo tanto se ha de elegir con cuidado el tamaño de la imagen para asegurarse de que todo el texto que se pretende ocultar tiene cabida en la imagen. De la forma que Wodax incrusta los caracteres en la imagen, se necesitan $\frac{4}{3}$ píxeles para ocultar un carácter. Entonces, si se han de esconder c caracteres, el número de píxeles p de la imagen debería ser, en principio, $p \geq \lceil c \times \frac{4}{3} \rceil$.

Esto sería así si los píxeles se modificasen de forma contigua, pero con el concepto de entropía que se ha explicado anteriormente, esto cambia ya que en la imagen quedarán píxeles que no se aprovecharán y por esto deberá ser más grande.

El valor del incremento de posición con una entropía ϵ está en el intervalo $[1, \epsilon]$. Así pues, el número de píxeles p necesarios para esconder un texto de c caracteres y utilizando un valor para la entropía ϵ , serán $\lceil c \times \frac{4}{3} \rceil \leq p \leq \lceil (\frac{1}{3} + \epsilon) \times c \rceil$.

2.6.2. Longitud de la contraseña

La restricción en este aspecto es que en el proceso de compilación del programa se define una longitud máxima para la contraseña (por comodidad). Si compiláis vosotros mismos el programa, podéis definir la longitud que encontráis oportuna, si descargáis un binario ya compilado desde la página web, la longitud máxima predeterminada es de 500 caracteres.

En la sección 3.2 se explicará como definir la longitud de la contraseña en el proceso de compilación.

2.6.3. Imágenes JPEG

Podría surgir la pregunta de por qué se ha utilizado el formato PNG para trabajar con las imágenes y no otro mucho más extendido como el JPEG. Hay dos motivos por los cuales no se ha utilizado el JPEG²⁰, un técnico y el otro ético.

El primer motivo es que el JPEG es un formato de compresión de imágenes con pérdida de calidad y los métodos para esconder texto en una imagen JPEG y después extraerlo correctamente son más complicados que sustituir los bits menos significativos en la imagen. Pero hay técnicas para poder utilizar este tipo de imagen [9].

El segundo motivo es que hay cierta incertidumbre²¹ relacionada con el formato JPEG y patentes a las que están sujetos algunos de los algoritmos que utiliza. Por esto se ha utilizado el PNG, un formato muy extendido en la Web también y de uso libre.

²⁰<http://www.w3.org/Graphics/JPEG/itu-t81.pdf>

²¹<http://en.wikipedia.org/wiki/JPEG#Potentialpatenteissues>

3. Sobre el proyecto en concreto

3.1. Historia

La idea del proyecto surgió tras leer el libro *Cryptonomicon* de Neal Stephenson[7]. El libro había hecho interesarme por la criptografía y la esteganografía y leyendo sobre el segundo me surgió la idea de hacer un programa *estegano-criptográfico* con tal de poner en práctica algunas de las ideas sobre las que había leído. Aquí está el resultado (parte de él, porque esto todavía no ha acabado).

El nombre “Wodax” proviene de “Shadow”. Cuando tenía lista la primera versión del programa estaba buscando un buen nombre, original y sencillo y que a la vez estuviera relacionado con el mundo de la criptografía y la esteganografía. De pronto, me vino a la cabeza la frase “Se ocultaba entre las sombras...” (no me preguntéis porqué... Seguramente, debo tener un trauma infantil). “Shadow” era un buen nombre para mi programa, ya que uno puede esconder cosas fácilmente en la sombra, igual que el programa en las imágenes. Pero el nombre no era demasiado original.

Entonces, cambié las letras “Sh” por la “X” pues en catalán la “X” al principio de una palabra tiene la misma pronunciación en algunos casos que la “Sh” en inglés. Finalmente, le di la vuelta al nombre y... “*tukun’ tish!*”: Ya tenía nombre para el programa.

Otra curiosidad sobre el programa es el orden que sigue el número de versiones. El número de la versión vendrá dado por un número real de la forma $x.y$ (x es la parte entera y y la fraccionaria), donde $x_i = y_{i-1}$ y $y_i = \frac{\varphi^i - (-\varphi)^{-i}}{\sqrt{5}}$, partiendo de $x_0 = 0$ y $y_0 = 1$. Esto significa que en la versión i 0 la parte fraccionaria se desplaza a la entera y se introduce el término y -ésimo de la Sucesión de Fibonacci a la parte fraccionaria. Las versiones serán las siguientes: **0,1**, **1,1**, 1,2, 2,3, 3,5, 5,8, 8,13, etc (marcadas en negrita las que ya han sido publicadas).

Esta forma de numerar las versiones es un guiño al proyecto T_EX, que a partir de su tercera versión incluye una cifra decimal más de forma que el número de la versión tiende a π (cuando Donald Knuth, el creador, muera se fijará el número de la versión a π)²².

3.2. Instalación y uso

3.2.1. Instalación

Para compilar el programa hay suficiente con ejecutar la orden `make` en la carpeta donde hayas descargado el programa.

Puedes modificar las opciones de compilación editando el archivo `makefile` que acompaña a los archivos de programa. A continuación se muestra una tabla que describe la funcionalidad de cada opción.

Opción	Función	Valor por defecto
CXX	Compilador	g++
FLAGS	Opciones del compilador	-Wall -pedantic -O3
INSTALL_PATH	Ruta de la instalación	/usr/bin
MAX_PASS	Longitud máxima de la contraseña	500

Una vez compilado puedes ejecutar la orden `make clean` para limpiar el directorio y eliminar los archivos temporales.

Para instalar el programa en la carpeta indicada en la variable `INSTALL_PATH` ejecuta la orden `make install` y para eliminarlo `make uninstall`.

²²<http://en.wikipedia.org/wiki/Softwareversioning#TeX>

3.2.2. Uso

Para ocultar texto ejecuta `wodax --hide (-h) image.png [options]` y para extraer texto oculto `wodax --seek (-s) image.png [options]`. Las opciones disponibles vienen resumidas en esta tabla.

Opción	Función [Modo = Hide o Seek]	Valor por defecto
<code>--output</code>	Nombre de la imagen con el texto oculto [H]	Imagen de entrada
<code>--file</code>	Fichero para ocultar en o extraer de una imagen [H,S].	Entrada/Salida estándar
<code>--entropy</code>	Valor de la entropía [H,S].	Longitud de la contraseña.

3.3. Librerías y código fuente

Las librerías extra que se han utilizado para construir el programa, aparte de las librerías estándar de C y C++, son la `libpng`²³ y un envoltorio en C++ para esta, llamado `png++`²⁴. La primera tiene una licencia Zlib y la segunda una modificación de la licencia BSD, ambas compatibles con la licencia GPLv3 bajo la que se publica Wodax.

La última versión del código fuente la podéis encontrar en el repositorio del proyecto²⁵ o en mi página web²⁶.

3.4. Licencia

3.4.1. Del programa

Wodax es software libre: usted puede redistribuirlo y/o modificarlo bajo los términos de la Licencia Pública General GNU publicada por la Fundación para el Software Libre, ya sea la versión 3 de la Licencia, o (a su elección) cualquier versión posterior.

Wodax se distribuye con la esperanza de que sea útil, pero SIN GARANTÍA ALGUNA; ni siquiera la garantía implícita MERCANTIL o de APTITUD PARA UN PROPÓSITO DETERMINADO. Consulte los detalles de la Licencia Pública General GNU para obtener una información más detallada.

Debería haber recibido una copia de la Licencia Pública General GNU junto a este programa. En caso contrario, consulte <http://www.gnu.org/licenses/>.

3.4.2. De la documentación

Copyright (c) 2008 Joan Puigcerver Pérez. Se concede permiso para copiar, distribuir y/o modificar este documento bajo los términos de la Licencia de Documentación Libre de GNU, Versión 1.3 o cualquier otra versión posterior publicada por la Free Software Foundation; sin Secciones Invariantes ni Textos de Cubierta Delantera ni Textos de Cubierta Trasera.

Este documento debería ir junto con una copia de la licencia. En caso contrario consulte <http://www.gnu.org/licenses/>.

Referencias

- [1] HERÒDOT. *Els nou llibres de la Història*, vol. V. 440aC.
- [2] KESSLER, G. C. An overview of steganography for the computer forensics examiner. Tech. rep., Computer and Digital Forensics Program, Champlain College, 2004. http://www.fbi.gov/hq/lab/fsc/backissu/july2004/research/2004_03_research01.htm.

²³<http://www.libpng.org/>

²⁴<http://www.nongnu.org/pngpp/>

²⁵<http://svn.asemeja.como/svn/wodax/>

²⁶<http://www.jpucgserver.net>

- [3] KRENN, J. Steganography and steganalysis, 2004. <http://www.krenn.nl/univ/cry/steg/article.pdf>.
- [4] NEIL F. JOHNSON, S. J. Steganography: Seeing the unseen. *IEEE Computer Journal* (1998). <http://www.jjtc.com/pub/r2026.pdf>.
- [5] NIELS PROVOS, P. H. Detecting steganographic content on the internet. Tech. rep., Center for Information Technology Integration, University of Michigan, 2001. <http://www.citi.umich.edu/techreports/reports/citi-tr-01-11.pdf>.
- [6] SIMMONS, G. J. The prisoner's problem and the subliminal channel. Tech. rep., Sandia National Laboratories, Albuquerque, 1983. <http://dsns.csie.nctu.edu.tw/research/crypto/HTML/PDF/C83/51.PDF>.
- [7] STEPHENSON, N. *Cryptonomicon*. Avon, 1999. <http://en.wikipedia.org/wiki/Cryptonomicon>.
- [8] TRITHEMIUS, J. *Steganographie: Ars per occultam Scripturam animi sui voluntatem absentibus aperiendi certu*. 1500.
- [9] YEUAN-KUEN LEE, L.-H. C. Secure error-free steganography for jpeg images. Tech. rep., Department of Computer and Information Science, National Chiao Tung University, 2003. <http://www.csie.mcu.edu.tw/~ykleee/Publications/SEFSJ.pdf>.