Trabajo práctico especial: Esteganografía en BMP

Garrigó, Mariano
54393

Marcantonio, Nicolás
56288

Raies, Tomás A.
56099

Saqués, M. Alejo
56047

1. Ej. 1

Para la implementación del programa stegobmp se pide que la ocultación comience en el primer componente del primer pixel. ¿Sería mejor empezar en otra ubicación? ¿Por qué?

Si analizamos las distintos puntos dónde podría comenzar la esteganografía, debemos tomar las siguientes consideraciones:

- Si el ocultamiento comenzase en el header del archivo *BMP* portador, la capacidad de un lector de imágenes de abrir el archivo correctamente se vería afectada. Por ejemplo, si se modificara el *header field* o el tamaño del archivo.
- Si se eligiera como comienzo un *byte* que no sea el primero, el tamaño máximo del archivo a esteganografiar será obligatoriamente menor.
- Debe haber un protocolo entre quien escribe el mensaje y quien lo lee para que se puedan comunicar efectivamente. Por ejemplo, esteganografiar sólo cada cierta cantidad de *bytes*, o similarmente al método *LSBE*, utilizar propiedades de la imagen para decidir qué *bytes* esteganografiar. Esto dificultaría la detección del esteganografiado, pero tiene el costo de una menor tamaño de mensaje disponible.
- Proponer un order de lectura diferente al lineal. Por ejemplo, comenzando desde el final o leyendo un caracter de cada extremo. Aunque sería más dificultoso descifrar el mensaje oculto, el efecto visual en la imagen sería similar. Además la dificultad de tener que leer archivos de forma no lineal (que implicaría tener que leer todo el archivo en un buffer en memoria a la hora de escribir o leer el mensaje) hace que no encotremos estas opciones muy recomendables.

Teniendo en cuenta que preferimos evitar leer el mensaje de un modo distinto al lineal, y si el deseo es aprovechar al máximo el espacio dentro del archivo portador, el mejor lugar para empezar a esteganografiar es el primer pixel. Si fuera deseable evitar escribir sobre los primeros bytes, uno podría idear un esquema dónde primero se esteganografía en el primer

byte un offset indicando donde estará oculto el resto del mensaje, lo que puede servir si se considera que comenzar a esteganografiar todos los mensajes al principio de la imagen es demasiado obvio (aunque, una vez se conoce este esquema, también es muy obvio)

2. Ej. 2

¿Qué ventajas podría tener ocultar siempre en una misma componente? Por ejemplo, siempre en el bit menos significativo de la componente azul.

De una forma similar a cómo un códec de imagen como JPEG utiliza *Chroma Subsampling* para reducir la tasa de muestreo del color de la imagen sin resultar en un cambio perceptible por un humano, se puede hacer uso práctico del hecho de que la visión es más susceptible a las variaciones de luminiscencia que en variaciones cromáticos para ocultar las sutiles diferencias en la gama de los pixeles alterados por la esteganografía. Así, la mejor opción sería utilizar la componente azul para almacenar información, sabiendo que la sensibilidad del ojo humano a la luminosidad es menor para colores cercanos al espectro de ese color.

3. Ej. 3

Esteganografiar un mismo archivo en un .bmp con cada uno de los tres algoritmos, y comparar los resultados obtenidos. Hacer un cuadro comparativo de los tres algoritmos estableciendo ventajas y desventajas.

Comparación de los algoritmos		
	Ventajas	Desventajas
LSB1	El cambio en la imágen es imperceptible al ojo humano debido a la manipulación del bit menos significativo	El archivo portador debe ser de tamaño al menos 8 veces mayor que el archivo oculto
LSB4	Consiste de la mejor opción para esteganografiar un archivo grande en un portador debido a la cantidad de bits utilizados	En ciertos casos la imágen presenta una diferencia considerable que puede ser perceptible al observar en detalle
LSBE	El cambio en la imágen es imperceptible al ojo humano. El método de ofuscación es más complejo, dificultando a un tercero el proceso de estegoanálisis y recupero del archivo	Constituye la opción más limitada en cuanto al tamaño del archivo a ocultar, debido a la baja probabilidad de encontrar numerosos bytes de valor 254 y 255. Además hace al método muy dependiente a las características de la imágen, ya que podría no poseer ninguno de estos bytes

4. Ej. 4

Para la implementación del programa stegobmp se pide que la extensión del archivo se oculte después del contenido completo del archivo. ¿por qué no conviene ponerla al comienzo, después del tamaño de archivo?

Ubicar la extensión del archivo al final en lugar de preceder al tamaño es conveniente, ya que resulta útil para la lectura de un mensaje encriptado. En primer lugar, se lee el tamaño el mensaje cifrado, y a partir del mismo se lee una cantidad de bytes igual a este valor (en el orden determinado por el método de esteganografiado). El mensaje encriptado pasa a encontrarse en memoria, y luego de desencriptar el mismo se obtiene un buffer con el tamaño, el archivo desencriptado y la extensión, en ese orden.

Debido a este orden es posible leer a un buffer auxiliar el archivo desencriptado (ya que su tamaño exacto es conocido), siendo la extensión del mismo la que queda en el buffer. Además permite ahorrarse medidas para evitar *buffer overflows*, ya que se conoce exactamente la longitud de la extensión al momento de la lectura (remanente del buffer). Sólo será necesaria una comprobación: que el tamaño del archivo sea menor al buffer auxiliar de descifrado menos 4 (los 32 bits del tamaño). De ser así, se tiene la certeza de que no habrá ningún *buffer overflow*.

5. Ej. 5

Explicar detalladamente el procedimiento realizado para descubrir qué se había ocultado en cada archivo y de qué modo.

Siguiendo las recomendaciones de la Cátedra, como prodedimiento previo a cualquier análisis con `stegobmp` se inspeccionó cada archivo con un *hex editor*. Además, se tomó como hipótesis que al menos uno de los archivos debería contener embebida − de alguna forma − información plana, caso contrario cualquier tipo de análisis sería computacionalmente inviable. Considerando estas aclaraciones, a continuación se discutirá el proceso de descubrimiento del secreto.

5.1. hugo.bmp

Analizando el archivo con el *hex editor* a partir del *byte* de offset 54, se puede hacer una estimación sobre el tamaño del *payload* (encriptado o no) que sigue después de los primeros 4 *bytes* esteganografiados. De esta forma, el siguiente análisis fue determinante:

- De ser el modo de inserción `LSB4`, el tamaño del contenido embebido debería ser de 1923545182 *bytes*, es decir, 1,93 *gigabytes*. Siendo el tamaño del archivo 3,75 *megabytes* (3932214 *bytes*), claramente `LSB4` queda rotundamente descartado.

- Similarmente, si el modo de inserción fuese `LSB1`, simplemente considerando que el primer *bit* del entero estaría encendido permite afirmar lo mismo que el caso anterior, dado que la presunta longitud L del *payload* sería $L \geq 2^{31} \gg 3932214$.

Por ende, se determinó que el modo de inserción debía ser `LSBE`. Se probó extraer el contenido asumiendo que éste fuese texto plano, lo cual fué exitoso. Se encontró un archivo `.pdf` con la leyenda *al .png cambiarle la extension por .zip y descomprimir*.

5.2. lima.bmp

En este caso, al explorar la apariencia del contenido binario del archivo, saltó a la vista que, a partir del offset 26892054, se encuentra en texto plano y sin esteganografiado la frase *la password es camaleon*. Se dejó de analizar este archivo considerando que este debía ser el cuarto archivo con información oculta.

5.3. secreto1.bmp

Sin encontrarse nada extraño a simple vista en el archivo binario, se procedió a hacer un análisis similar al del primer caso, con la salvedad de que el método `LSBE` está descartado, per el enunciado del TPE.

Por ende, asumiendo que el archivo contiene información esteganografiada en modo `LSB4`, el tamaño del archivo embebido debería ser de 2758568170, es decir, aproximadamente 2,76 *gigabytes*. Siendo el tamaño del archivo 25 *megabytes*, claramente se puede descartar `LSB4` como método de inserción. Por ende, este archivo debe estar esteganografiado en `LSB1`.

Se probó extraer el contenido en dicho modo asumiendo que fuese plano, extrayéndose *exitosamente* un archivo en formato `.wmv`. Al intentar abrirlo, todo reproductor de vídeos indicaba que la información estaba corrupta. Más adelante se verá que, en realidad, el contenido esteganografiado estaba encriptado (pero en modo `LSB1`). Evidentemente, la Cátedra colocó, además, en dicho archivo, después del *payload* encriptado, la extensión del texto plano, haciendo que el programa `stegobmp` no falle si se le dice que la información no viene encriptada.

5.4. titanic.bmp

Habiendo descartado todos los otros métodos, el único que queda es `LSB4`. Se probó obtener el archivo embebido asumiendo que fuese texto plano, obteniéndose exitosamente una imagen en formato `.png`. En dicho archivo, se observa un tablero de un juego de *buscaminas*. Según el primer `.pdf`, este sería el archivo que habría que cambiarle la extensión a `.zip` e intentar descomprimir.

Se probó hacer esto, pero ningún intento dió resultados. Investigando técnicas mediante las cuales podría hacerse que un archivo de imagen contenga, a su vez, a otro, se encontró [1] que una técnica consiste en simplemente concatenar un archivo *huésped* con el *oculto*, de tal forma que el *huésped* se pueda abrir correctamente y no se evidencie a simple vista contenido oculto.

Considerando esta técnica, se analizó el archivo `.png` en búsqueda de información anómala al final del mismo. Se encontró 2 veces un *string* `sol1.txt`. Sabiendo que la información podría ser un `.zip`, se buscó el inicio del mismo, marcado por la secuencia de *bytes* `50 4b 03 04`, es decir `0x04034b50` leído en *little endian*. Efectivamente, en el *offset* 43036 se detectó dicha secuencia. Se extrayó a partir de dicha posición hasta el final el archivo `.zip` concatenado, pudiéndose así obtenerse las pistas para resolver el mensaje oculto en el juego de buscaminas.

En dicho mensaje, se afirmaba una cosa ya conocida: *la password está en otro archivo*. Dicho archivo es `lima.bmp`, y la *password* es `camaleon`.

Además, se afirma que, con el *string* obtenido del buscaminas, se obtendría algoritmo y modo de encripción del archivo `.wmv`. Conociendo este dato, y el hecho que ya se había *obtenido* un archivo `.wmv` *corrupto* de `secreto1.bmp`, se cayó en la cuenta de que, efectivamente, `secreto1.bmp` no contenía información en texto plano, sino que habría que utilizar la información obtenida del buscaminas para desencriptarlo.

Sabiendo que las minas representan 1s, el *string* resultante fue el siguiente:

- (01)000100 - D
- (01)100101 - e
- (01)110011 - s
- (01)000101 - E
- (01)100011 - c
- (01)100010 - b

Luego, como conclusión de toda la información hasta ahora:

- **Algoritmo:** DES
- **Modo:** ECB
- **Password:** camaleon
- **Modo de esteganografiado:** LSB1
- **Archivo:** secreto1.bmp

[1] `http://wiki.linuxquestions.org/wiki/Embed_a_zip_file_into_an_image`

Con estos datos, el vídeo secreto pudo desencriptarse.

6. Ej. 6

¿Qué se encontró en cada archivo?

Ya se ha discutido en la respuesta anterior, pero como resumen:

6.1. hugo.bmp

Archivo `.pdf`, cuyo contenido indica que se debe cambiar la extensión *del archivo .png a .zip y descomprimirlo.*

6.2. lima.bmp

La *password* de encripción del `.wmv` oculto en `secreto1.bmp`.

6.3. secreto1.bmp

Vídeo `.wmv`, esteganografiado y encriptado con los parámetros indicados más arriba. En el fragmento de esta serie, los personajes empiezan discutiendo sobre un *email* en el que, presuntamente, uno de sus agentes habría roto con alguien. Discuten sobre qué tan *aceptable* es este protocolo para *romperle el corazón a alguien*, hasta que la agente se percata de que un archivo es *más grande de lo que debería*, lo que significaría que hay algún error de compresión o que *el agente ocultó datos en el archivo.*

6.4. titanic.bmp

Imagen `.png`, en la que se observa un juego de buscaminas. En el mismo archivo, y concatenada a la imagen huésped, se encuentra un archivo `.zip` que contiene un `.txt` describiendo cómo leer el juego de buscaminas para obtener el algoritmo y modo con el que otro archivo (el `.wmv`) está encriptado. Es decir, el `.png` es *más grande de lo que debería.*

7. Ej. 7

Algunos mensajes ocultos tenían, a su vez, otros mensajes ocultos. Indica cuál era ese mensaje y cómo se había ocultado.

Tal como se discutió hasta ahora, el `.png` contenía oculto en él un `.zip`.

8. Ej. 8

Uno de los archivos ocultos era una porción de un video, donde se ve ejemplificado una manera de ocultar información ¿cuál fue el portador?

Como se ha dicho, el portador del vídeo es `secreto1.bmp`. La manera de ocultar información es la que se puede observar en el `.png`: concatenar dos archivos, haciendo que el primero se siga leyendo como si no contuviera ningún mensaje adicional. No se ha investigado sobre el protocolo de los archivos `.png`, pero, evidentemente, dicho protocolo no obliga verificar que no haya *bytes* de sobra por sobre los declarados en su *header*. Lo mismo pasaría en un `.bmp`.

9. Ej. 9

¿De qué se trató el método de esteganografiado que no era LSB? ¿Es un método eficaz? ¿Por qué?

El método de esteganografiado encontrado en los archivos de prueba que no se trató de LSB fue simplemente concatenar al final del archivo portador el archivo huésped en texto plano, técnica utilizada tanto en `lima.bmp`, como en el archivo `.png` resultante de extraer el contenido esteganografiado en `titanic.bmp`.

Este método puede realizarse en ambos tipos de archivo dado que se especifica la cantidad de bytes a leer. En el caso del archivo `.bmp`, su header contiene información del tamaño del archivo, por lo que si agregamos más bytes al final, el programa que usemos para visualizar el `.bmp` los ignorará y los bytes ocultos pasan desapercibidos, siempre y cuando no se utilize un *hex editor*.

En cuanto al archivo `.png`, éste contiene sus *bytes* divididos en *chunks*, donde el primer chunk es un IHDR *chunk* y el último un IEND *chunk*, que son los delimitadores. Cada chunk tiene también los *bytes* a leer. Es por eso que si se agregan *bytes* al final del IEND *chunk*, ningún visualizador de archivos `.png` leerá los bytes ocultos.

En cuanto a espacio, no es un método tan eficaz como los métodos LSB, dado que agranda el tamaño del archivo final. Sin embargo, el beneficio de éste método es la simplicidad a la hora de programarlo.

10. Ej. 10

¿Qué mejoras o futuras extensiones harías al programa stegobmp?

Una primera mejora propuesta, la cuál es fácilmente implementable debido a la estructura del código, es agregar soporte para `LSBN`, siendo $1 \leq N < 7$. Otra mejora sería agregar soporte para RSA.

Sería factible hacer un programa que oculte información en un arreglo de imágenes `.bmp`. Esto sería útil para casos en los que el archivo oculto es demasiado grande como para que ocultarlo en una única imagen no levante sospechas. Debería diseñarse un protocolo sobre cómo especificar el orden de los fragmentos, siendo una opción esteganografiar $size||position||data||extension$ en texto plano, o $size_{enc}||enc(size||position||data||extension)$ sino.

Sería ideal que `stegobmp` pueda detectar al menos la presencia de contenido oculto automáticamente. Una opción sería automatizar el método de descarte de algoritmos de inserción expuesto en la respuesta 5. Luego, obtenidos los posibles métodos, se podría intentar extraer el contenido en texto plano, guardando los resultados para posterior análisis por un experto esteganógrafo. Este debería determinar si existía efectivamente un archivo esteganografiado. Una de las opciones sería que el programa extraiga exitosamente contenido en texto plano. Caso contrario, el hipotético archivo oculto estaría encriptado, por lo que se debería realizar criptoanálisis sobre los archivos resultantes para extraer más información de los mismos.