Exploring Steganography: Seeing the Unseen

Steganography is an ancient art of hiding information. Digital technology gives us new ways to apply steganographic techniques, including one of the most intriguing—that of hiding information in digital images.

Neil F. Johnson
Sushil Jajodia
George Mason University

Steganography is the art of hiding information in ways that prevent the detection of hidden messages. *Steganography*, derived from Greek, literally means "covered writing." It includes a vast array of secret communications methods that conceal the message's very existence. These methods include invisible inks, microdots, character arrangement, digital signatures, covert channels, and spread spectrum communications.

Steganography and cryptography are cousins in the spycraft family. Cryptography scrambles a message so it cannot be understood. Steganography hides the message so it cannot be seen. A message in ciphertext, for instance, might arouse suspicion on the part of the recipient while an "invisible" message created with steganographic methods will not.

In this article we discuss image files and how to hide information in them, and we discuss results obtained from evaluating available steganographic software. For a brief look at how steganography evolved, see the "Steganography: Some History" sidebar.

IMAGE FILES

To a computer, an *image* is an array of numbers that represent light intensities at various points (pixels). These pixels make up the image's *raster data*. A common image size is 640 × 480 pixels and 256 colors (or 8 bits per pixel). Such an image could contain about 300 kilobits of data.

Digital images are typically stored in either 24-bit or 8-bit files. A 24-bit image provides the most space for hiding information; however, it can be quite large (with the exception of JPEG images). All color variations for the pixels are derived from three primary colors: red, green, and blue. Each primary color is represented by 1 byte; 24-bit images use 3 bytes per pixel to represent a color value. These 3 bytes can be represented as hexadecimal, decimal, and binary values. In many Web pages, the background color is represented by a six-digit hexadecimal number—actually three pairs representing red, green, and blue. A white background would have the value FFFFFF: 100 percent red (FF), 100 percent green (FF), and 100 percent blue (FF). Its decimal value is 255, 255, 255, and its binary value is 11111111, 11111111, 11111111, which are the three bytes making up white.

This definition of a white background is analogous to the color definition of a single pixel in an image. Pixel representation contributes to file size. For example, suppose we have a 24-bit image 1,024 pixels wide by 768 pixels high—a common resolution for high-resolution graphics. Such an image has more than two million pixels, each having such a definition, which would produce a file exceeding 2 Mbytes. Because such 24-bit images are still relatively uncommon on the Internet, their size would attract attention during transmission. File compression would thus be beneficial, if not necessary, to transmit such a file.

File compression

Two kinds of compression are *lossless* and *lossy*.[1] Both methods save storage space but have different results, interfering with the hidden information, when the information is uncompressed. Lossless compression lets us reconstruct the original message exactly;

therefore, it is preferred when the original information must remain intact (as with steganographic images). Lossless compression is typical of images saved as GIF (Graphic Interchange Format) and 8-bit BMP (a Microsoft Windows and OS/2 bitmap file).

Lossy compression, on the other hand, saves space but may not maintain the original image's integrity. This method typifies images saved as JPEG (Joint Photographic Experts Group). Due to the lossy compression algorithm, which we discuss later, the JPEG formats provide close approximations to high-quality digital photographs but not an exact duplicate. Hence the term "lossy" compression.

Embedding data

Embedding data, which is to be hidden, into an image requires two files. The first is the innocent-looking image that will hold the hidden information, called the *cover image*. The second file is the message—the information to be hidden. A message may be plain text, ciphertext, other images, or anything that can be embedded in a bit stream. When combined, the cover image and the embedded message make a *stego-image*.[2] A stego-key (a type of password) may also be used to hide, then later decode, the message.

Most steganography software neither supports nor recommends using JPEG images, but recommends instead the use of lossless 24-bit images such as BMP. The next-best alternative to 24-bit images is 256-color or gray-scale images. The most common of these found on the Internet are GIF files.

In 8-bit color images such as GIF files, each pixel is represented as a single byte, and each pixel merely points to a color index table (a palette) with 256 possible colors. The pixel's value, then, is between 0 and 255. The software simply paints the indicated color on the screen at the selected pixel position. Figure 1a, a red palette, illustrates subtle changes in color varia-

Steganography: Some History

Throughout history, people have hidden information by a multitude of methods and variations.[1,2] For example, ancient Greeks wrote text on wax-covered tablets. To pass a hidden message, a person would scrape wax off a tablet, write a message on the underlying wood and again cover the tablet with wax to make it appear blank and unused. Another ingenious method was to shave the head of a messenger and tattoo a message or image on the messenger's head. After the hair grew back, the message would be undetected until the head was shaved again.

Invisible inks offered a common form of invisible writing. Early in World War II, steganographic technology consisted almost exclusively of these inks.[1] With invisible ink, a seemingly innocent letter could contain a very different message written between the lines.[3]

Documents themselves can hide information: document text can conceal a hidden message through the use of null ciphers (unencrypted messages), which camouflage the real message in an innocent-sounding missive. Open coded messages, which are plain text passages, "sound" innocent because they purport to be about ordinary occurrences. Because many open-coded messages don't seem to be cause for suspicion, and therefore "sound" normal and innocent, the suspect communications can be detected by mail filters while "innocent" messages are allowed to flow through.[1] For example, the following null-cipher message was actually sent by a German spy in WWII[1]:

> Apparently neutral's protest is thoroughly discounted and ignored. Isman hard hit. Blockade issue affects pretext for embargo on by-products, ejecting suets and vegetable oils.

Decoding this message by extracting the second letter in each word reveals the following, hidden message:

> Pershing sails from NY June 1.

Document layout may also reveal information. Documents can be marked and identified by modulating the position of lines and words.[4]

Message detection improved with the development of new technologies that could pass more information and be even less conspicuous. The Germans developed microdot technology, which FBI Director J. Edgar Hoover referred to as "the enemy's masterpiece of espionage."[1] Microdots are photographs the size of a printed period having the clarity of standard-sized typewritten pages, which permits the transmission of large amounts of data, including drawings and photographs.[1]

With every discovery of a message hidden with an existing application, a new steganographic application is being devised. Old methods are given new twists. While drawings have often been used to conceal or reveal information, computer technology has, in fact, sparked a revolution in such methods for hiding messages.

Space limitations prevent further discussion here. For more information on techniques for hiding information, see Peter Wayner's *Disappearing Cryptography*.[5]

References

1. D. Kahn, *The Codebreakers*, Macmillan, New York, 1967.
2. B. Norman, *Secret Warfare*, Acropolis Books, Washington, D.C., 1973.
3. H.S. Zim, *Codes and Secret Writing*, William Morrow, New York, 1948.
4. J. Brassilet et al., "Document Marking and Identification using Both Line and Word Shifting," *Proc. Infocom95*, IEEE CS Press, Los Alamitos, Calif., 1995.
5. P. Wayner, *Disappearing Cryptography*, AP Professional, Chestnut Hill, Mass., 1996.

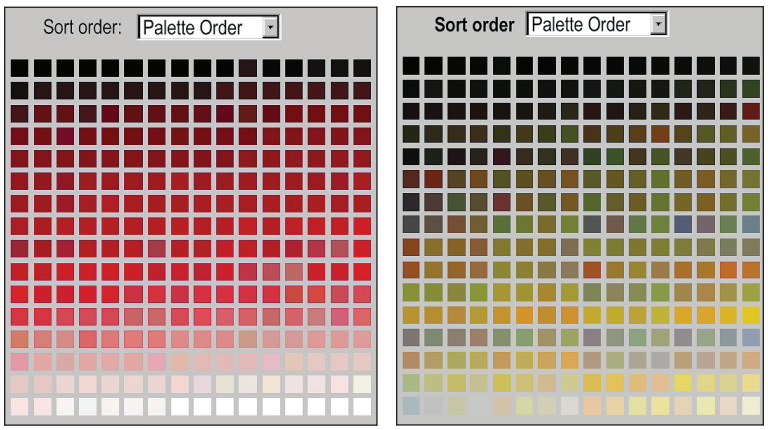

Figure 1. Representative color palettes. (a) A 256-color red palette and (b) a 256-color Renoir palette. The Renoir palette is so named because it comes from a 256-color version of Pierre-Auguste Renoir's "Le Moulin de la Galette."

tions: visually differentiating between many of these colors is difficult. Figure 1b shows subtle color changes as well as those that seem drastic.

Many steganography experts recommend using images featuring 256 shades of gray.[3] Gray-scale images are preferred because the shades change very gradually from byte to byte, and the less the value changes between palette entries, the better they can hide information. Figure 2 shows a gray-scale palette of 256 shades. Some images are 4-bit, created with 16 shades of gray; obviously these images offer many fewer variations.

While gray-scale images may render the best results for steganography, images with subtle color variations are also highly effective, as Figure 1 showed.

When considering an image in which to hide information, you must consider the image as well as the palette. Obviously, an image with large areas of solid colors is a poor choice, as variances created from the embedded message will be noticeable in the solid areas. We will see that Figure 1b, the palette for the Renoir cover image, makes a very good cover for holding data.

Once you've selected a cover image, you must decide on a technique to hide the information you want to embed.

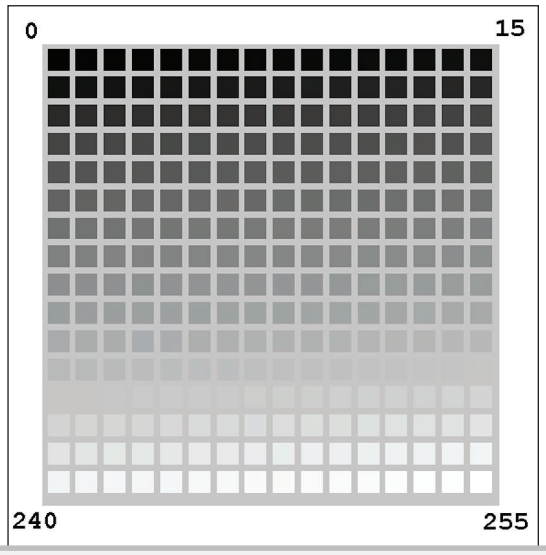

Figure 2. Representative gray-scale palette of 256 shades.

CONCEALMENT IN DIGITAL IMAGES

Information can be hidden many different ways in images. To hide information, straight message insertion may encode every bit of information in the image or selectively embed the message in "noisy" areas that draw less attention—those areas where there is a great deal of natural color variation. The message may also be scattered randomly throughout the image. Redundant pattern encoding "wallpapers" the cover image with the message.

A number of ways exist to hide information in digital images. Common approaches include

- least significant bit insertion,
- masking and filtering, and
- algorithms and transformations.

Each of these techniques can be applied, with varying degrees of success, to different image files.

Least significant bit insertion

Least significant bit (LSB) insertion[4] is a common, simple approach to embedding information in a cover file. Unfortunately, it is vulnerable to even a slight image manipulation. Converting an image from a format like GIF or BMP, which reconstructs the original message exactly (*lossless compression*) to a JPEG, which does not (*lossy compression*), and then back could destroy the information hidden in the LSBs.

24-bit images. To hide an image in the LSBs of each byte of a 24-bit image, you can store 3 bits in each pixel. A 1,024 × 768 image has the potential to hide a total of 2,359,296 bits (294,912 bytes) of information. If you compress the message to be hidden before you embed it, you can hide a large amount of information. To the human eye, the resulting stego-image will look identical to the cover image.

For example, the letter A can be hidden in three pixels (assuming no compression). The original raster data for 3 pixels (9 bytes) may be

```
(00100111 11101001 11001000)
(00100111 11001000 11101001)
(11001000 00100111 11101001)
```

The binary value for A is `10000011`. Inserting the binary value for A in the three pixels would result in

```
(00100111 11101000 11001000)
(00100110 11001000 11101000)
(11001000 00100111 11101001)
```

The underlined bits are the only three actually changed in the 8 bytes used. On average, LSB requires that only half the bits in an image be changed. You can hide data in the least and second least significant bits and still the human eye would not be able to discern it.

8-bit images. 8-bit images are not as forgiving to LSB manipulation because of color limitations. Steganography software authors have devised several approaches—some more successful than others—to hide information in 8-bit images. First, the cover image must be more carefully selected so that the stego-image will not broadcast the existence of an embedded message.

When information is inserted into the LSBs of the raster data, the pointers to the color entries in the palette are changed. In an abbreviated example, a simple four-color palette of white, red, blue, and green has corresponding palette position entries of 0 (00), 1 (01), 2 (10), and 3 (11), respectively. The raster values of four adjacent pixels of white, white, blue, and blue are 00 00 10 10. Hiding the binary value 1010 for the number 10 changes the raster data to 01 00 11 10, which is red, white, green, blue. These gross changes in the image are visible and clearly highlight the weakness of using 8-bit images. On the other hand, there is little visible difference noticed between adjacent gray values, as Figure 2 shows.

Implementing LSB. Steganography software processes LSB insertion to make the hidden information less detectable. For example, the EzStego tool arranges the palette to reduce the occurrence of adjacent index colors that contrast too much—before it inserts the message. This approach works quite well in gray-scale images and may work well in images with related colors.

S-Tools, another steganography tool, takes a different approach by closely approximating the cover image, which may mean radical palette changes. As with 24-bit images, changing the pixels' LSB may create new colors. (New colors may not be added to an 8-bit image due to the palette limit.) Instead, S-Tools reduces the number of colors while maintaining the image quality, so that the LSB changes do not drastically change color values.

For example, eight color values are required for each color if values 000 through 111 are to be stored. Reducing the number of unique colors to 32 ensures that these values can be used and that the number of colors will not exceed 256 (256/8 = 32). Each of the 32 unique colors in the palette may be expanded to eight colors having LSB values of the red, green, blue (RGB) triples ranging from 000 to 111. This results in multiple colors in the palette that look the same visually but that may vary by one bit.[5]

Figure 3. Image "painted" with the watermark: "Invisible Man" © 1997, Neil F. Johnson. Traditional steganography conceals information; watermarks extend information and become an attribute of the cover image.

These tools take a similar approach with gray-scale images. However, the resulting stego-images as applied with S-Tools are no longer gray-scale. Instead of simply going with adjacent colors as EzStego does, S-Tools manipulates the palette to produce colors that have a difference of one bit. For example, in a normal gray-scale image, white will move to black with the following RGB triples

```
(255 255 255), (254 254 254), ...,
(1 1 1), (0 0 0)
```

After processing with S-Tools, the value for white will be spread over a range of up to eight colors such as

```
(255 255 255), (255 255 254), and
(255 254 255)
```

Visually, the stego-image may look the same as the gray-scale cover image, but it has actually become an 8-bit color image.

Masking and filtering

Masking and filtering techniques, usually restricted to 24-bit and gray-scale images, hide information by marking an image, in a manner similar to paper watermarks. Watermarking techniques may be applied without fear of image destruction due to lossy compression because they are more integrated into the image.

Visible watermarks are not steganography by definition. The difference is primarily one of intent. Traditional steganography *conceals* information; watermarks *extend* information and become an attribute of the cover image. Digital watermarks may include such information as copyright, ownership, or license, as shown in Figure 3. In steganography, the object of communication is the hidden message. In digital water-

Steganography's niche in security is to supplement cryptography, not replace it. If a hidden message is encrypted, it must also be decrypted if discovered, which provides another layer of protection.

marks, the object of communication is the cover.

To create the watermarked image in Figure 3, we increased the luminance of the masked area by 15 percent. If we were to change the luminance by a smaller percentage, the mask would be undetected by the human eye. Now we can use the watermarked image to hide plaintext or encoded information.

Masking is more robust than LSB insertion with respect to compression, cropping, and some image processing. Masking techniques embed information in significant areas so that the hidden message is more integral to the cover image than just hiding it in the "noise" level. This makes it more suitable than LSB with, for instance, lossy JPEG images.

Algorithms and transformations. LSB manipulation is a quick and easy way to hide information but is vulnerable to small changes resulting from image processing or lossy compression. Such compression is a key advantage that JPEG images have over other formats. High color quality images can be stored in relatively small files using JPEG compression methods; thus, JPEG images are becoming more abundant on the Internet.

One steganography tool that integrates the compression algorithm for hiding information is Jpeg-Jsteg. Jpeg-Jsteg creates a JPEG stego-image from the input of a message to be hidden and a lossless cover image. According to the Independent JPEG Group, the JPEG software we tested has been modified for 1-bit steganography in JFIF output files, which are composed of lossy and nonlossy sections. The software combines the message and the cover images using the JPEG algorithm to create lossy JPEG stego-images.

JPEG images use the discrete cosine transform to achieve compression. DCT[6] is a lossy compression transform because the cosine values cannot be calculated exactly, and repeated calculations using limited precision numbers introduce rounding errors into the final result. Variances between original data values and restored data values depend on the method used to calculate DCT. For details, see Wayne Brown and Barry Shepherd's book on graphic file formats.[7]

In addition to DCT, images can be processed with fast Fourier transformation and wavelet transformation.[8] Other image properties such as luminance can also be manipulated. Patchwork[9] and similar techniques use *redundant pattern encoding* or spread spectrum methods[10] to scatter hidden information throughout the cover images ("patchwork" is a method that marks image areas, or patches). These approaches may help protect against image processing such as cropping and rotating, and they hide information more thoroughly than by simple masking. They also support image manipulation more readily than tools that rely on LSB.

In using *redundant pattern encoding*, you must trade off message size against robustness. For example, a small message may be painted many times over an image as shown in Figure 3 so that if the stego-image is cropped, there is a high probability that the watermark can still be read. A large message may be embedded only once because it would occupy a much greater portion of the image area.

Other techniques *encrypt and scatter* the hidden data throughout an image. Scattering the message makes it appear more like noise. Proponents of this approach assume that even if the message bits are extracted, they will be useless without the algorithm and stego-key to decode them. For example, the White Noise Storm tool is based on spread spectrum technology and frequency hopping, which scatters the message throughout the image. Instead of having x channels of communication that are changed with a fixed formula and passkey, White Noise Storm spreads eight channels within a random number generated by the previous window size and data channel. Each channel represents 1 bit, so each image window holds 1 byte of information and many unused bits. These channels rotate, swap, and interlace among themselves to yield a different bit permutation. For instance, bit 1 might be swapped with bit 7, or both bits may rotate one position to the right. The rules for swapping are dictated by the stego-key and by the previous window's random data (similar to DES block encryption).

Scattering and encryption helps protect against hidden message extraction but not against message destruction through image processing. A scattered message in the image's LSBs is still as vulnerable to destruction from lossy compression and image processing as is a clear-text message inserted in the LSBs.

Steganography's niche in security is to supplement cryptography, not replace it. If a hidden message is encrypted, it must also be decrypted if discovered, which provides another layer of protection.

EVALUATION EXAMPLES

To determine the limitations and flexibility of available software, we evaluated several steganographic packages. Here we discuss only three: StegoDos, White Noise Storm, and S-Tools for Windows. For details on other tools, see the sidebar "For More Information."

First, we selected message and cover files. In some cases, we had to alter the selected images to fit into the constraints of the software or had to use other cover images. In all, we tested 25 files as cover images. For purposes of this article, we discuss the results we obtained with only two message files and two cover image files.

The first message file contained this plain text:

> Steganography is the art and science of communicating in a way which hides the existence of the

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.