

STEGANALYSIS TECHNIQUES FOR JPEG IMAGES

Introduction

Steganography is the art and science of hiding the fact that communication is taking place. Steganographic systems can hide messages inside of images or other digital objects. To a casual observer inspecting these images, the messages are invisible.

1. Steganography Background

The term "Information Hiding" relates to both watermarking and steganography. Watermarking usually refers to methods that hide information in a data object so that the information is robust to modifications. That means, it should be impossible to remove a watermark without degrading the quality of the data object.

On the other hand, steganography refers to hidden information that is fragile. Modifications to the cover medium may destroy it.

Watermarking and steganography differ in another important way: while steganographic information must never be apparent to a viewer unaware of its presence, this feature is optional for a watermark.

Modern steganography should be detectable only if secret information is known, namely, a secret key. This is very similar to "Kerckhoffs' Principle" in cryptography, which holds that the security of a cryptographic system should rely only on the key material .

Because of their invasive nature, steganographic systems leave detectable traces within a medium's characteristics. This allows an eavesdropper to detect modified

media, revealing that secret communication is taking place. Although the secret content is not exposed, its existence is revealed, which defeats the main purpose of steganography.

In general, the information hiding process consists of the following steps:

1. Identification of redundant bits in a cover medium. Redundant bits are those bits that can be modified without degrading the quality of the cover medium.
2. Selection of a subset of the redundant bits to be replaced with data from a secret message. The stego medium is created by replacing the selected redundant bits with message bits.

The modification of redundant bits can change the statistical properties of the cover medium. As a result, statistical analysis may reveal the hidden content.

2. Information Hiding in JPEG Images

This section briefly explains the JPEG format and how it can be used for information hiding.

The JPEG image format uses a discrete cosine transform (DCT) to transform successive 8×8-pixel blocks of the image into 64 DCT coefficients each. The least-significant bits of the quantized DCT coefficients are used as redundant bits into which the hidden message is embedded.

In some image formats, e.g. GIF, the visual structure of an image exists to some degree in all bitlayers of the image. Steganographic systems that modify least-significant bits of

these image formats are often susceptible to visual attacks.

This is not true for the JPEG format. The modification of a single DCT coefficient affects all 64 image pixels. For that reason, there are no known visual attacks against the JPEG image format.

3. Statistical Analysis

We can observe that embedding encrypted data into a GIF image changes the histogram of its color frequencies. One property of encrypted data is that the one and the zero bit are equally likely. When using the least-significant bit method to embed encrypted data into an image that contains the color two more often than the color three, the color two is changed more often to the color three than the other way around. As a result, the difference in color frequency between two and three has been reduced by the embedding.

The same is true for JPEG images. Instead of measuring the color frequencies, we analyze the frequency of the DCT coefficients. We use a chi-square test to determine whether an image shows distortion from embedding hidden data. Because the test uses only the stego medium, the expected distribution for the chi-square test has to be computed from the image. Let n_i be the frequency of DCT coefficients in the image. We assume that an image with hidden data embedded has similar frequency for adjacent DCT coefficients.

4. Steganographic Systems in Use

In this section, we present several steganographic systems that embed hidden messages into JPEG images. We show that the statistical distortions depend on the steganographic system that inserted the message into the image. Because the distortions are characteristic for each system, we develop signatures that allow us to identify which system has been used.

There are three popular steganographic systems available on the Internet that hide information in JPEG images:

- JSteg, JSteg-Shell
- JPHide
- OutGuess

All of these systems use some form of least significant bit embedding and are detectable by statistical analysis except the latest release of Out-Guess. In the following, we present the specific characteristics of these systems and show how to detect them.

4.1 JSteg and JSteg-Shell

The data of the message is prepended with a variable size header. The first five bits of the header express the size of the length field in bits. The following bits contain the length field that expresses the size of the embedded content.

JSteg-Shell is very simple. Because, it is just a user interface to JSteg, it does not encrypt the length of the embedded message. Instead it adds a signature at the end of the message. The signature is either "korejwa", "cMk4" or "cMk5". We get at least 32 bits of certainty that we guessed the right password. However, because the key size is restricted to 40 bits, it is feasible to search the whole key space.

JSteg-Shell is a Windows user interface to JSteg. It has been developed by Korejwa and supports encryption and compression of the content before embedding. The data with JSteg. JSteg-Shell uses the stream cipher RC4 for encryption. However, the RC4 key space is restricted to 40 bits. When encryption is being employed, we expect the probability of embedding to be high at the beginning of the image. There should be no exception.

4.2 JPHide

The DCT coefficients are not selected continuously from the beginning, JPHide is slightly more difficult to detect. The program

uses a fixed table that determines which coefficient to modify next. The coefficients are selected by the table in such a way that coefficients that are likely to be numerically high are used first. A pseudorandom number generator determines if coefficients are skipped. The probability of skipping bits depends on the length of the hidden message and how many bits have been embedded already.

JPHide not only modifies the least-significant bits of the DCT coefficients, it can also switch to a mode where the second-least-significant bits are modified.

4.3 Outguess

OutGuess is different from the systems described in the previous sections in that it chooses the DCT

coefficients with a pseudo-random number generator.

A user-supplied pass phrase initializes a stream cipher and a pseudo-random number generator, both based on RC4. The stream cipher is used to encrypt the content. Because the modifications are distributed randomly over the DCT coefficients, the chi-square test can not be applied on a continuously increasing sample of the image. Instead, we slide the position where we take the samples across the image. For OutGuess 0.13b, we do not find any clear signatures.

5. Stegdetect

Stegdetect detects images that have content hidden with JSteg, JPHide and OutGuess 0.13b. For each system that we want to detect, we select the DCT coefficients in the order that they are modified and apply a chi-square test.

The output from Stegdetect lists the steganographic systems found in each image or "negative" if no steganographic content could be detected. Stegdetect

expresses the level of confidence of the detection with one to three stars.

Figure 1 shows some sample output.

```
misc/0003-wonder-2.jpg : jphide(*)
misc/dscf0001.jpg : outguess(old)(***)
misc/dscf0002.jpg : negative
misc/dscf0003.jpg : jsteg(***)
```

Figure 1: The output from Stegdetect contains an estimate of the detection confidence.

5.1 JSteg Detection

JSteg does not modify the DCT coefficients zero and

one. For that reason, they are ignored in the chi-square test. We sample the DCT coefficients starting from the beginning of the image and compute the probability of embedding. This process is repeated with increasing sample size until all DCT coefficients are contained in the sample. As a performance optimization, we stop computing the probability of embedding once it falls below a certain threshold. To improve the detection accuracy, we estimate the size of the hidden content from the calculated graph and compare it with the size stored in the JSteg embedding header as described in Section.

5.2 JPHide Detection

Because JPHide modifies the DCT coefficients in a fixed order determined by a table, we rearrange the coefficients in that order before computing the probability of embedding.

However, there are two exceptions that influence the detection. JPHide modifies the DCT coefficients -1, 0 and 1 in a special way. As a result, the modifications to these coefficients cannot be detected by the chi-square test. However, simply ignoring these coefficients still allows us to detect content embedded with JPHide. We also ignore modifications to the second-least-significant

bits, which are not as frequent as modifications to the least-significant bits.

Similar to JSteg, we stop computing the probability of embedding once it falls below a certain threshold.

5.3 OutGuess Detection

Detecting content embedded with OutGuess 0.13b is complicated by the fact that the coefficients are selected pseudo-randomly, there is no fixed order in which to apply the chi-square test. However, Provos has shown that the chi-square test can be extended to detect content hidden with OutGuess 0.13b.

Instead of increasing the sample size and applying the test at a constant position, we use a constant sample size but slide the position where the samples are taken over the entire range of the image.

The test starts at the beginning of the image, and the position is incremented by one percent for every application of the chi-square test. The extended test does not react to an unmodified image, but detects the embedding in some areas of the stego image.

A binary search on the sample size is used to find a value for which the extended chi-square test does not show a correlation to the expected distribution derived from unrelated coefficients.

6. Verifying Hidden Content

The statistical tests used to find steganographic content in images indicate nothing more than a likelihood that content has been embedded. Because of that, Stegdetect can not guarantee the existence of a hidden message.

To verify that the detected images have hidden content, it is necessary to launch a dictionary attack against the JPEG files.

Stegbreak does just that for content hidden with JSteg-Shell, JPHide or Outguess 0.13b.

Because all the presented steganographic systems

hide content based on a user supplied password, an attacker can try to guess the password to determine what content has been hidden. Instead of trying all possible passwords, it is much faster to try only words from dictionary, i.e. a dictionary attack.

For a dictionary attack to work, it is necessary that the user of the steganographic system selects a weak password, i.e. he selects the password from a small subset of the full password space.

Key attacks on cryptographic systems often have the benefit that properties of the underlying plaintext are known to the attacker. Given these properties, it is possible to verify statistically if the correct decryption key has been found. All the steganographic systems presented in this paper embed header information in addition to a message into the images. The header information contains, among other things, the length of the hidden message. We can use this information to verify the correctness of the guessed password.

Conclusion

Steganography can be used for hidden communication. Instead of measuring the color frequencies, we analyze the frequency of the DCT coefficients. We use a chi-square test to determine whether an image shows distortion from embedding hidden data. Stegdetect techniques including JSteg detection, JPHide detection and OutGuess detection allow us to automatically detect steganographic content in JPEG images.

Steganography Demo

To try this demo you need a Windows 98/Me/NT/2000/XP/Vista computer. Please follow the instructions below.

1. Download and save the zip file containing the [steganography tools and image files](#)
2. Unzip the file *steg.zip* into an empty directory. If you do not have the *unzip* program already, you can get one.
3. Put a shortcut to *S-tools.exe* on your desktop by dragging it from Windows Explorer.
4. Drag the *zebras.bmp* file to your desktop. Do not make a shortcut. The file itself must be moved there.
5. Start *S-tools.exe* by double clicking on the icon on the desktop. A window will appear.
6. Drag the *zebras.bmp* file to the S-tools window.
7. Right click on the zebras pictures and select Reveal from the menu.
8. Fill in the 3-character pass phrase 'abc' (without the quotes) in two places. Leave IDEA as the encryption algorithm. Click on OK.
9. Wait until the Revealed Archive dialog box appears. This may take a minute or two.
10. Right click on any item and select Save As to save the file. Repeat for the other ones. These are the hidden files.
11. The file *original-zebras.bmp* is the file before the steganography was done, in case you wish to compare the 'before' and 'after' images.



Note: The image at the left has been compressed for quick downloading of this page. It does not contain any secret messages so do not download it unless you really like zebras