

Destruction of Steganography

Targeting the Least Significant Bit in 24 bit
Images

William Jamieson

Abstract:

This paper looks at the use of techniques to prevent the communication of steganographic data by filtering. These filters are broken into two categories; Overwriting and Dissolving. *Overwriting* can be defined as being the effective implementation of a Steganographic message over the original message. *Dissolving* can be defined as being modifications to the picture in order to prevent the decoding of the image. Both methods do little damage to the cover image, while destroying the hidden data and, ideally, making it unrecoverable. This paper also reviews exactly which type of steganography this works for, and how that steganography is hidden.

Introduction:

The exchange of information is greater today than at any other time in history, and with the way technology is improving, this exchange will only become greater. The threat of a group or individual misusing standard channels of communication to send stolen information, or communicate a plan against a nation, is more likely now than ever (Goel, 2007). Due to the nature of these communications, the need for a means to hide them is ever present; this is where steganography comes into play. Through the use of steganography, information can be transmitted while being disguised within another piece of data. Significant amounts of data can be moved through common means of electronic communication, with little threat of detection. This data can be transmitted with the hidden information included, and travel across networks looking like normal traffic. Any third party that intercepts the data will not expect it to contain such a secret. Implementation of steganography is not hard to achieve, and there are multiple variations of programs that will encode and decode information (Johnson, 1998). Hiding data through steganographic means has become easier due to the availability of free programs online. In addition, the number of technologically adept individuals is increasing on a daily basis. The ability to deal with steganography will become a more crucial skill in future information flows.

Effort has been made to establish ways of detecting whether or not a piece of data contains a steganographic element. Unfortunately, not many steganographic messages can be detected (Goel, 2007). In addition to the difficulties related to detection, the message is still encoded, and extra time is needed to translate the message into an understandable format. The message can even be encrypted while hidden in the data via steganographic means, allowing the data to be sent without there appearing to be an encrypted file, but still having the benefits of being encrypted. This process of detection and analysis can be time consuming, meaning that the attack against steganography is not a real-time attack.

The interception and analysis of steganography is not always necessarily needed. What *is* important is preventing the intended party from receiving the data that is being sent. This could be accomplished by simply stopping data that is detected as steganographic, however, in the course of detection, there is still a significant amount of false positives, which would result in data being stopped that has no steganographic content. In addition, not all data that has steganographic content will be detected and stopped. A need for a better form of security against the transmission of steganography must be implemented, one which can be achieved in real time.

This end could be accomplished through alterations to the picture. These alterations can either be made within the cover image, or directly to the bits that store the hidden message. Both methods result in the destruction of the hidden data with little damage to the image it is embedded in, also known as the cover image. Destruction of steganography on a mass scale will serve as a means to protect information, and prevent hidden communication.

What is Steganography?

To understand methods of detecting and/or destroying steganography, it is important to understand just what steganography is. Steganography is a general term for a process of hiding information, most commonly messages. The word *Steganography* has its origin in the Greek language, meaning “covered writing” (Johnson, 1998). These messages, and more importantly the methods for hiding them, have evolved over the course of history, and present day steganography is concerned with the digital environment. Modern steganography is carried out in such a way that data is hidden within other data. The hidden data is not retrievable unless the recipient knows exactly how the data was hidden, or analyzes the data, and figures out the method by which it was hidden (Johnson, 1998).

Steganographic data is most commonly hidden within a picture file. This data is either stored as a picture hidden within the main picture, or in the data that makes up the bits of the picture file. Other common file types utilized by steganography include document files and audio files, both of which are much harder to create than a steganographic picture file, but offer increased protection against detection (Artz, 2001). The focus of this paper is on data hidden within the bits of a picture file, through the least significant bits.

Technical Overview:

In *Least Significant Bit Steganography*, the last bit in each color byte is manipulated (Fig 1), and allows for the storage of extra information. This is the easiest way to create a piece of steganography, and the majority of free use programs use this technique as well. This type of

steganography is best used with 24-bit pictures because three pixels can hold the binary data for one character (Johnson, 1998).

```
Blue 10101101  
Red 11010110  
Green 11111101  
  
Message 01110101  
  
Modified Colors  
Blue 10101100  
Red 11010111  
Green 11111101
```

Figure 1: Steganography in a 24 bit image

The above example requires a bit of a breakdown, as many individuals are not versed on how a pixel of a picture is defined. In a 24-bit picture, the image is broken down into pixels, each with a red, blue, and green value. These pixels are each one byte (8 bits) long, and together add up to be 24 bits long. Each color has 256 variations, based on the binary value of the byte. The variations are different shades of the color in question, with variations in the start of each byte resulting in a more drastic color difference than with those at the end. This means that changes to the last bit in each color byte results in the least amount of color change. This bit is referred to as the Least Significant Bit. Each pixel can store 3 bits of the hidden data (Venkatraman, 2004).

The Least Significant Bit has the lowest impact on the overall color of each pixel and very little impact on the overall picture. The color of a modified picture is not visibly changed to the human eye, but a program that retrieves the last bit from each color byte can construct the original message. The way a program hides and retrieves this information is rather simple. When writing the information, the program takes the binary data for each character in the message and writes it bit by bit over the last bit of each color byte. For example in figure one, the first bit of the message is 0 and the last bit in the color red is 1, but the program will replace the 1 in Least Significant Bit for red with the message value of 0. The program will continue replacing the last bit of each color byte until the entire message is hidden. When finished, the picture still looks identical to the original, but has within it a payload of information.

Retrieval is easy in the simple case, as all one needs to do is know which color byte was first in the writing process and starting reading from that Least Significant Bit and continue with each consecutive Least Significant Bit until the messages binary data is retrieved in whole. This data can then be assembled and the message constructed out of it. This is not always as easy as it sounds, however.

A message that is being hidden through steganographic means is normally also encrypted when being embedded in the cover image (Artz, 2001). In Least Significant Bit Steganography this means that after the binary data is pulled from the Least Significant Bit, it is actually an encrypted message. This adds a layer of protection to the message, making it much harder to gain access if you were not the one it was intended for. In the most common Least Significant Bit Steganography programs, this encryption is done by means of a Feedback Cipher.

The use of a Feedback Cipher in encryption means that every byte in the hidden message is reliant on the preceding byte to determine the cipher used to decode that byte. This means that even if you were to collect all the data and try to look for similarities to help decrypt the message, it would not be useful, since each byte has a different cipher key. For example, 10010011 in one place is not the same message as 10010011 in another location within the same encrypted message. There is, however, a drawback to this method; any point where the data is modified after encryption causes the entire message to be indecipherable (Venkatraman, 2004).

Targeting the hidden message:

The objective is to effectively destroy steganography by targeting the hidden message. In an ideal situation, the data that holds the message can be targeted directly and the message can be destroyed. This is not the case, though, as any time the message is altered, the data is as well. However, if the message can be destroyed while only causing minor alterations to the cover media, then it is still successful enough to be used as an effective counter measure. The main goal of targeting the hidden message is to change the picture enough so that degradation of quality is not an issue, while damaging the hidden data within to a state beyond recovery. This would allow for images that do not contain a hidden message to be attacked, and not be drastically altered. The benefit of this method is that it can be done covertly, and on a scale that can prevent all steganographic traffic of the targeted type.

How to implement such a technique is the issue at hand, coming down to a single question; how do we do the most damage to the hidden message, while leaving the cover image intact? Looking for an answer several possibilities were found which could be divided into two categories, the previously mentioned Overwriting and Dissolving. *Overwriting* is writing over the Steganographic message within the data, while *Dissolving* is modifying the cover image in order to damage the hidden message indirectly.

Overwriting:

Overwriting is essentially just writing over the hidden message with another message. If limited in scope to target exactly the bits that have a message, the process causes effectively no change within the picture. In Least Significant Bit steganography, this would of course be the last bit of each color byte (Fig 2).

Hidden Message 01110101

Modified Colors

Blue 10101100

Red 11010111

Green 11111101

Overwrite 10101010

Blue 10101101

Red 11010110

Green 11111101

Figure 2: Overwrite of 24 bit Least Significant Bit Steganography

The message above has been overwritten with a new message, causing the old message to be removed in the process. This is the main idea of overwriting, to write over the current data with other data in order to remove the steganographic content from the image.

Implementation of overwriting:

Overwriting can be implemented easily, with the simplest form being another application of Least Significant Bit steganography applied to the picture. Watermarking is an effective way of doing this, as most watermarks are also stored in the Least Significant Bit. Watermarking is another form of steganography, its intent being that the mark will not be found, except by someone who knows where and how to look for it. They can now check the watermark for the creator of the image. By applying a watermark that takes up the entire image, you overwrite the whole of the hidden data, effectively destroying it. This would be the same as running the original tool that wrote the message in the first place. Both would overwrite the message with a different message, leaving the old one unrecoverable by the original tool. Another way of utilizing the method Overwriting is to implement a wipe of the data in the Least Significant Bits. This could be done by writing ones (1) or zeros (0) over every last bit, without the need of hiding another message.

Dissolving:

Dissolving is harder to implement as efficiently, when compared to Overwriting. In Dissolving, the cover image is actually being edited, which results in the destruction of the hidden data. This destruction causes pieces of the hidden data to be changed, dissolving the hidden message. The theory behind the Steganographic message being dissolved is that, as changes are applied to the cover image, the binary data that composes the message is altered. In the simplest of cases, this means that each pixel that is altered results in a character loss to the message. If enough alterations are made to the cover image, the data that is hidden within becomes unreadable. Overall, this process should attempt to cause little change to the picture, and maximum change to the hidden message. Again, just like Overwriting, if the message is encrypted by means of a Feedback Cipher, any alteration to the hidden message alters the cipher and prevents the decryption of the message.

Implementation of Dissolving:

To dissolve a hidden message within an image, the picture must be edited within a limited scope. While it would be simple to black out the picture as a whole to get rid of the data within, this level of modification is not acceptable, because of the overall damage to the image. Dissolving is a fragile balance between the quantities of hidden data damaged vs. cover data damaged. To start, let's look again at the way a 24 bit image is stored (Fig 1). The 8 bits each color has in a 24 bit picture changes when the picture is changed to a 16 bit picture. In a 16 bit picture, the amount each color receives for pixel space is 5 bits or 32 colors. These colors are much more diverse in spectrum than those present in 24 bit format. Due to this increase in color diversity, colors that are converted from 24 bit to 16 bit end up becoming more simplified and lose their hidden message (Johnson, 1998). Afterwards, the image can be converted back to a 24 bit image, which results in all of the hidden data being lost. Other methods of Dissolving can target the way the hidden message is protected. If the steganography uses a Cipher Feedback, it can be stopped by the modification of any byte of the hidden data (3 pixels or 72 bits of the picture), resulting in a change of the cipher for all subsequent bytes of the message. This means that the message cannot be interpreted by the same process that encoded it. Furthermore, data stored in Least Significant Bit can be attacked by modifying the color values of the picture. If all green in the image is edited, resulting in a value change for the green data, then all hidden data stored in the "green" of the data will be lost. This results in destruction of one-third of the hidden messages bits. These modifications can be done with little alteration to the overall picture, making the listed procedures go unnoticed to anyone viewing to cover image.

Testing:

In order to achieve validation of the results in each of the different methods, it is necessary to establish a testing procedure. The following tests were conducted with S-tools, a free-to-download Least Significant Bit steganography program which not only hides the data, but encrypts the hidden data as well. This encryption method is a Cipher Feedback encryption, so each byte is dependent on the byte before it for decryption methodology. Also, detection of a method such as a Cipher Feedback is easier, due to the statistical average of the data. This makes it easier to analyze the final results with a Chi-Square test. The Chi-Square test looks at the distribution of the data, and analyzes it for mathematical dependence. A high P-percentage means that the data being analyzed has a mathematical correlation, in this case, the prime values shared throughout the cipher as part of the encryption methodology. In addition to the Chi-Square analysis, the population density of the Least Significant Bits in the picture is also tested. Higher population densities mean a greater chance that hidden information is inside the image, indicating that the data is most likely Steganographic. The main goal is to prevent the hidden data from being recovered, and by analyzing the Chi-Square results and population density, it is possible to interpret how much of the hidden message is destroyed.

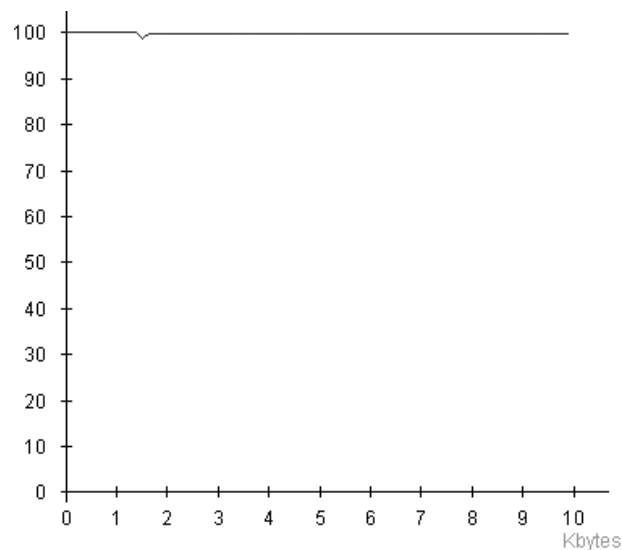


Figure 3: Chi-Square Analysis of Image with Steganographic content.

The sample data used here is a 2.2 MB, 24 bit picture, containing 5 text documents. The population density is 96.14%, and has a Chi-Square analysis showing nearly a hundred percent (Fig 3). This is the basis data for all other tests, showing both the presence of the Steganography in population density, and in relation to the bit before it.

Testing overwriting:

Both tests for Overwriting resulted in the hidden data becoming unrecoverable by the same tool that conducted the embedding. The result of overwriting the file with the same tool yielded an expected outcome, in that the previous data was no longer attainable, and both the Chi-Square analysis and the population density indicated hidden data. The hidden data that was retrieved is the new message which overwrote the original message. The tool is able to recover the second quantity of hidden data, but is unable to recover the first hidden data set. The graph for this is not shown, as it is identical to that of the sample. Both tests on the data will indicate steganography is present; however, this is to be expected, because the data that has replaced the previous method will show up the same on both tests.

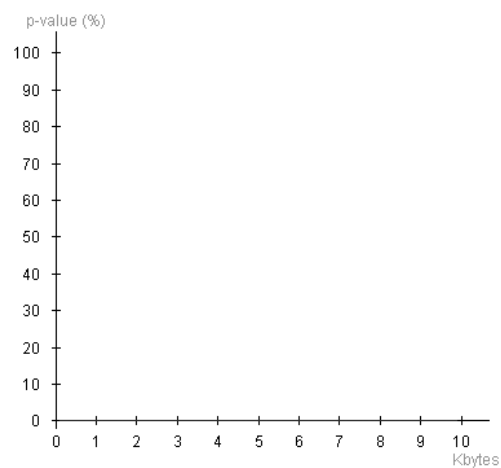


Figure 4: Chi-Square Analysis of watermarking

The second test also yielded predictable results, as again the data was not able to be recovered with the tool. However, unlike a second application of the steganography tool, the results of a Chi-Square analysis (Fig 4) yielded a flat value of 0%. When the population density was examined, the final result was 31.4%. This indicates that the watermark that was used was not encrypted, and overwrote the value in the Least Significant Bit of the pixels, compromising the picture. The watermark also wrote over the entire picture, meaning that this method would work well against non-Feedback Cipher encrypted steganography.

Testing Dissolving:

The method of Dissolving is more varied in testing possibilities, largely because it is only limited by the amount of editing tools that one has access to. Any edit that can change the values of the Least Significant Bits in the image can be used; however the most valuable tools

are those that can change the Least Significant Bits, while not causing any visible change to the appearance of the cover image.

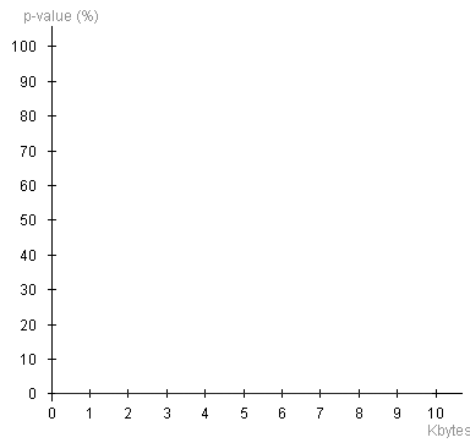


Figure 5: Chi-Square Analysis of quality reduction

To begin, the process of reducing the quality of the picture is executed. The original image is reduced in quality from a 24 bit .BMP to a 16 bit .BMP, then saved and returned to original quality. This change results in loss of data to the pixels that compose the hidden message. Visually, there is no noticeable change; the picture only seems a shade lighter. The Chi-Square analysis is flat at zero percent once more, indicating no presence of the Least Significant Bits being mathematically dependent, since most of the original Least Significant Bits have been lost in the change. The population density (Fig 5) shows an interesting result, with only 15.36% of the Least Significant Bits being used. This percentage is broken down as approximately 14% red, 3% green and 28% blue. This uneven proportion further serves to demonstrate that the hidden data is destroyed. The original sample was much more uniform in how the independent values were distributed with, at most, a 3% difference between any two colors. The reduction in density, combined with the huge difference between how much each color is used, means that the hidden data is destroyed. This is further confirmed when the tool that was used to recover the data was unable to find any hidden message within.

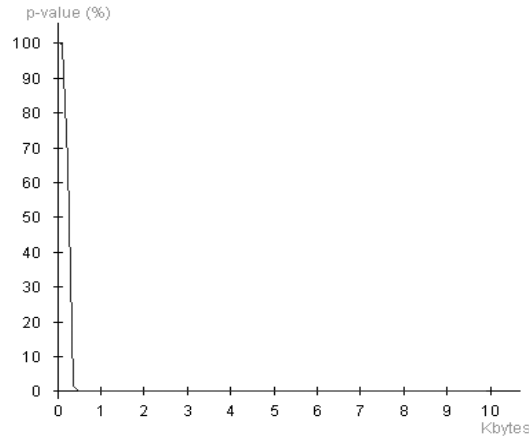


Figure 6: Chi-Square Analysis of color shift

Next we consider the modification of one color, and its impact on the steganographic data. Like the other methods, this resulted in the failure of the tool to recover the hidden message. The color shift was tested on the green channel for the picture, and yielded interesting results. The Chi-Square analysis (Fig 6) shows a breakdown in the mathematical dependence from start to finish. This indicates that the encrypted data is present, but is increasingly damaged throughout the picture. The population density remains high with 84% of the Least Significant Bits being used. The tool, when set to recovery, seemed to still find the information for the hidden beginning of the file. Without finishing, the tool crashed, because the data it was receiving was not able to be interpreted by the program. This method works well on data that is encrypted by a Feedback Cipher, but on plain text messages, this method may not be able to eliminate the whole message. Mathematically this change can be expressed as a fifty percent chance of change in every third value that composes the image. An unencrypted byte contains eight pixels worth of Least Significant Bits. Since nine pixels are present in the three pixels needed to compose a byte of data, the amount each color weighs on each byte is changing between two and three. This means that on the low end, $3/4$ of the message is lost and on the high end, $7/8$ of the message is lost. The loss is a character-by-character loss, so simple words become no longer understandable (i.e. if the message were "I went to the store", there is a good chance that any single character would be a completely different character).

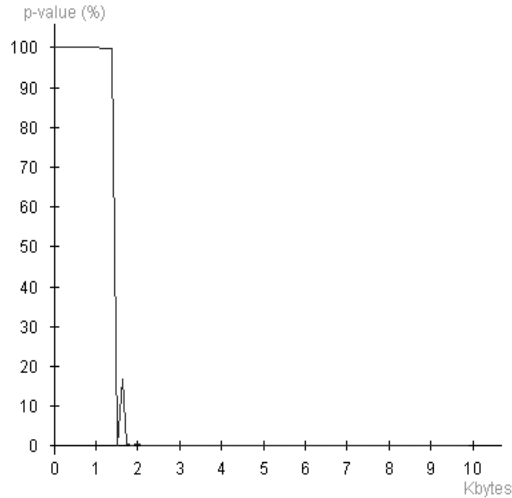


Figure 7: Chi-Square Analysis of corner coloration.

Manipulation in which the encryption is key to failure was also tested. In the first test, a small black dot was colored in each corner of the picture. This process aims to prevent reading of the steganographic message by breaking the Cipher Feedback algorithm. The result was a plummet in the mathematical relationship between the Least Significant Bits. The file was unable to be read by the tool, showing that the encryption method was not robust enough to survive such an attack. The population density (Fig 7) results were 94.21%, meaning that even though the mechanism to read the message failed, the message is still present. This sort of attack would only work on a message that uses a Feedback Cipher; an unencrypted message would still be able to be recovered, and it would make sense.

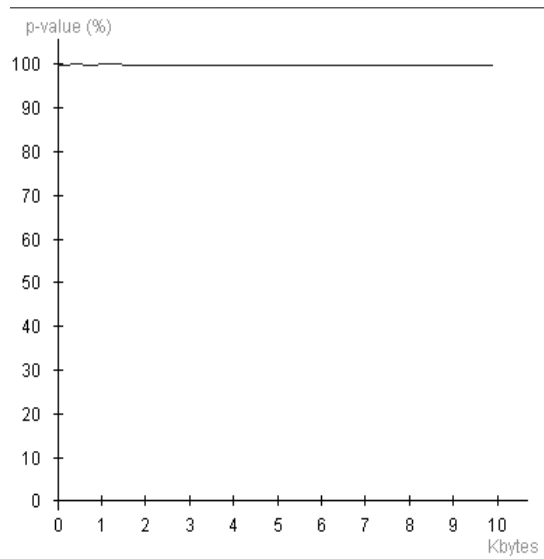


Figure 8: Chi-Square Analysis of pixel shaving.

In the second test, which targeted the encryption method of the data, 1 pixel worth of dimension was cut off of the width and height of the image. This dimension was removed off of the left and bottom of the picture. The Chi-Square analysis (Fig 8) resulted in nearly 100% mathematical certainty of encryption being present. The population density of the modified picture came out to be 98.61%, indicating the presence of a hidden message. The files were able to be read by the recovery tool, but the text in each file was garbage data. This means that the image was altered just enough to throw off the encryption mechanism, but not significantly enough to impair the mechanism from operating. This method is dependent on the data being encrypted with a Feedback Cipher.

Conclusion:

The results of the different tests show that both Overwriting and Dissolving are viable methods in destroying hidden messages within steganography, more specifically Least Significant Bit applications in images. The damage done to the cover image in both methods is limited in scope, and can stop the interpretation of the hidden message. Even with these shared successes, there is a difference between how each method excels.

Overwriting is much more effective in guaranteeing that the data cannot be recovered. Through the modification of the Least Significant Bits, the appearance of the cover image is not noticeably changed. This destroys the entire hidden message, as long as the Overwriting data is the same length or greater than the original hidden message. This method provides the most effective destruction of the hidden data, and is the most covert.

Unfortunately, this method is very target specific. In application of Least Significant Bit steganography, it is easy to find an overriding method, but this is not the case with other forms of steganography. If the method in which the data was hidden does not match the overwriting method, then the overwriting will have no effect on the hidden message. For simple steganographic programs, like most of the ones that are free, overwriting the Least Significant Bit will work, but overwriting is hard to implement against more complex methods of steganography.

In addition, Overwriting applies a set number sequence to the cover image, and if this sequence is known, some data may be retrieved. The whole hidden payload, however, would not be able to be recovered due to overlap of the same bits. For example if you wrote over the byte 01110110 with 10001000 one might try to recover the original by implementing the reverse, resulting in 01110111. This is close to the original, and it may be possible to figure out the original data from it.

Dissolving is a more diverse method for dealing with steganography. The process of Dissolving is varied, and one technique of Dissolving can work on more than just Least Significant Bit steganography. Reducing the quality of the image to one that has less memory, then

restoring the image back, is the most successful method of Dissolving that was tested, and would work well with other forms of image steganography.

The biggest drawback to a Dissolving method, however, is the fact that the cover image is damaged in the process of making the hidden data unrecoverable. This damage limits the usefulness of such a method in regards to covert application. Some of the hidden data will remain untouched in some dissolving approaches, as well. If the method is not trying to be implemented covertly, and the loss of some picture quality is acceptable, Dissolving is a better choice, as it does not have to be a targeted attack.

Dissolving is also a better choice over Overwriting if the data has been heavily encrypted, as any small change within the data has a larger effect on the hidden message. Any minor alteration in the cover image is then possible, resulting in the hidden message being unrecoverable. The message must have a type of feedback encryption in place for this to work with only small alterations.

Final Remarks:

Destroying steganography is possible, but targeting the specific form is problematic. This paper has shown that Least Bit Steganography can be destroyed efficiently, but other ways of implementing steganography exist, and even more will be created. Through analyses of how these methods hide data, it is likely similar methods can be developed. The benefit of such techniques is the interruption of the flow in hidden information. The key to defeating steganography by these means is the full understanding of how it is being implemented.

References:

1. Goel, R. , Garuba, M. , Liu, C. , Nguyen, T. (2007). The Security Threat Posed by Steganographic Content on the Internet. Information Technology, 2007. ITNG '07. Fourth International Conference on. 794-798. DOI: 10.1109/ITNG.2007.192
2. Johnson, N. F. , Jajodia, S. (1998) Exploring Steganography: Seeing the Unseen, IEEE Computer, February 1998, vol. 31, no. 2, pp.26-34
3. Venkatraman, S. , Abraham, A. , Paprzycki, M. (2004) Significance of Steganography on Data Security. Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004. International Conference on. 347 – 351. DOI: 10.1109/ITCC.2004.1286660
4. Artz, D. (2001) Steganography: Hiding Data within Data , Internet Computing, IEEE, May/Jun 2001 Volume 5, Issue 3, pp. 75-80 DOI: 10.1109/4236.935180