****

**An Example of Steganography Tools**

**Year 3 Computer & network Forensics**

**Submitted by:**

**James Finglas – B00094138**

**Word count: 2054**

**Submission date**

**11/11/2018**

Contents

[Abstract 3](#_Toc529702181)

[Introduction 3](#_Toc529702182)

[Steganography Programs 4](#_Toc529702183)

[Xiao Steganography 2.6.1 4](#_Toc529702184)

[Hide’N’Send 9](#_Toc529702185)

[rSteg 14](#_Toc529702186)

[OpenStego 17](#_Toc529702187)

[Conclusions 20](#_Toc529702188)

# Abstract

The purpose of this report is to examine four types of steganographic programs to highlight the methods used in imbedding and retrieving hidden messages. Also, the report examines how any signatures might be used by an investigator to steganalyze such files to identify the presence of the hidden messages.

# Introduction

For this report I chose to avoid some of the more common programs in favor of some that might be less known but offered some interesting features. They are as follows:

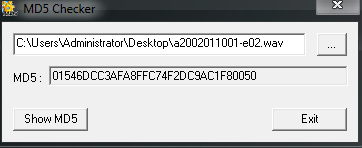
* Xiao Steganography 2.6.1
* Hide’N’Send
* rSteg
* OpenStego

These programs where found to be user friendly as opposed to some that required compilation in c++ IDE (Integrated development environment) programs, or indeed the use of the Linux Operating system.  
  
The purpose of examining these programs was to gain familiarity with how these programs hide files, the encryption methods employed and such methods might alter the file to hide the existence of message.  
  
Furthermore, we could then use these methods and our newly gained understanding of them to attempt to identify signatures that might help if a blind steganalysis where to be attempted on these files.

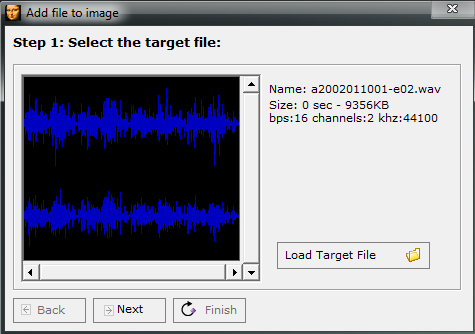
# Steganography Programs

## Xiao Steganography 2.6.1

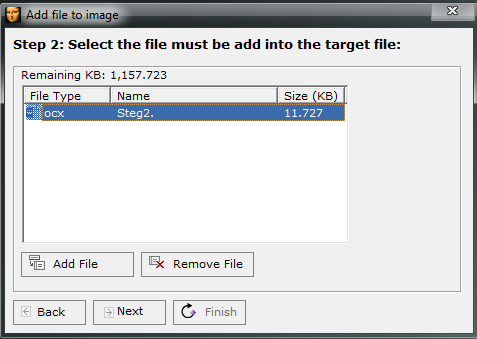
This is a popular program used to embed steganography into image and wave files. The program uses only the least significant bit(LSB) method as other audio formats apply only to audio. For the sake of file type cross compatibility between .img and .wave files is maintained through the restriction to the LSB injection method.



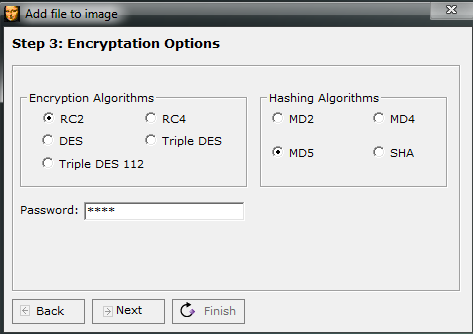
*here we have the MD5 HASH of our original wave file (a2002011001-e02.wave)*



*Here we have selected our target (carrier/virgin) file*

**

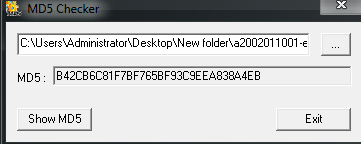
*Here we have selected our target (Stego Object) file.*

**

*Here, see the available algorithmic options, as well as the various hash option followed by a password option.*

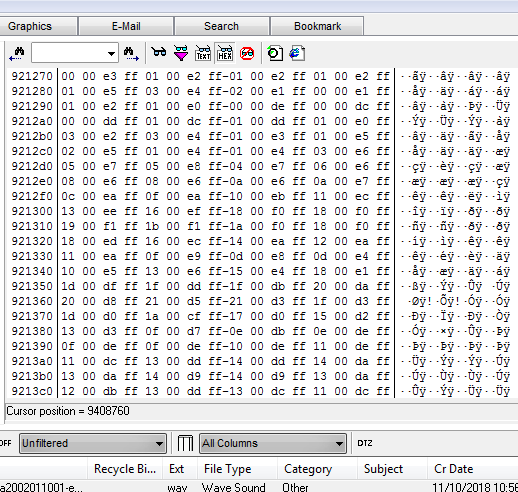
Clicking next at this stage completes the process and provides us with a completed Stego Object, which I have placed in a new folder and given the same name as the original file. Now begins our attempt to analyse the file.

Our step is to compare MD5 Hashes. If we do not see a matching MD5 this tells us immediately we have a suspect file.

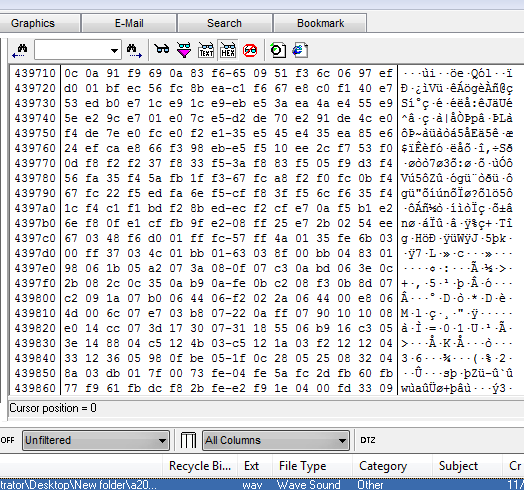


*MD5 HASH of suspect file.*

As seen above, our MD5 HASH values do not match, so we have identified this file warrants further investigation. Our next step is to compare hex values of the files, starting first with our original wave file.

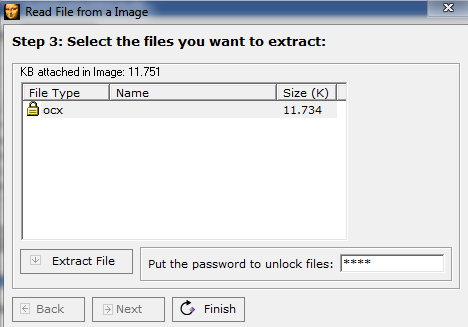


*Here we see the hex values of Original file, and as you can see, the hex structure of the file is very specific.*



*Here we see the hex values of the stego object inserted into the carrier file.*

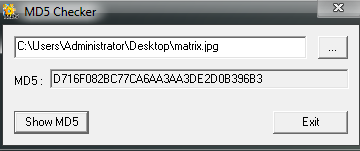
So, what we see here, is that we have two .wav files of the same name. Both work fine. However, the MD5 Hash values do not match, nor do the hex values. This would be considered a successful steganalysis as we have identified that a message likely hides inside this message. However, while we have identified that a message is hidden, we have not identified the exact method of steganography. This would need to be done during the retrieval process, if an active investigation was taking place.  
  
In the case of this example i was aware of the Steganography program used. Therefore, I simply reversed the process. One thing was interested. The file was embed using a ocx file type. When extracting the image, I needed to know the original file name and file extension. I also needed to apply the password. In this case the file name/extension was ‘steg1.txt’ and the password was ‘steg’.  
This led to the file being successfully retrieved.

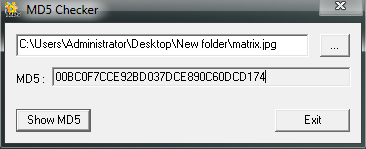


*Here we see the process being reversed*

## Hide’N’Send

Here we have a program very similar to Xiao Steganography. Again, we simply select the carrier image file in which we wish to embed our Stego object text file. Once the hide option is selected, an option to add a password is offered and the object is hidden.

  
  
*Here we see the original MD5 hash of the original carrier file.*

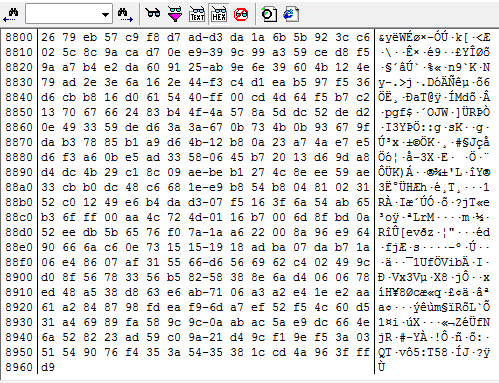


*Here we see the MD5 of the carrier file with stego object inserted.*

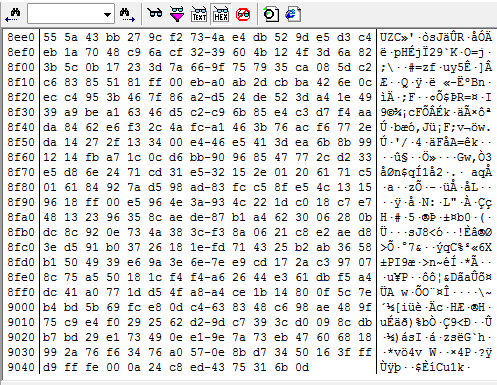


*Here we see the Stego object being imbedded into the carrier file*

In this particular case you will notice in the image above, the steganography has been encrypted. This will alter the signature of the new file thus making the detection of the hidden message more difficult. You may notice a different hash algorithm was selected. While this will of course also change the signature from that of a similar image hashed with an md5 algorithm it should be noted that the database of known signatures and hashes is not quite vast and while these kinds of variations might fool an investigator in a blind analysis they are far less likely to fool an automated tool with access to the latest hash and signature databases.

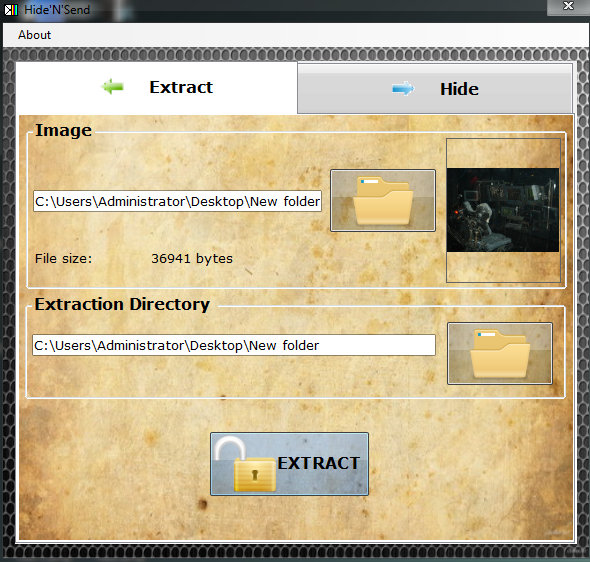


*Here we see the hex view of the original unaltered carrier file.*

**

*Here we see the hex file of the newly created stego object.*

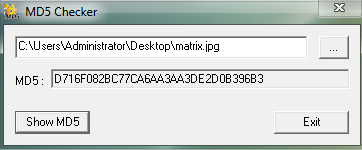
As seen above, the newly created stego object has Both a variance in the MD% hash, and the hex values when compared to the original carrier file. We have achieved another successful detection of a hidden message. Furthermore, we have begun to establish a pattern in how many common steganography programs will appear when begin analysed. However, we must not get complacent. Remember there are a vast array of uncommon programs available, and signatures will not always be easily spotted.

  
*Here we see the process being reversed and the stego object being extracted.*

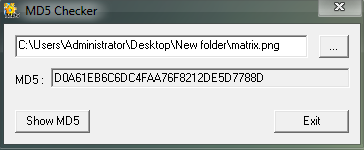
The reversal process here was also very similar, with one quality of life difference. During the extraction It was not necessary to rename the original stego object with its original file name. this was done automatically. This would be a useful feature as it removes the need for the user to know these details in advance and adds extra ease of use appeal to the program.

## rSteg

This is an interesting program which was designed by students. No information is provided about the algorithms used hashing methods or if encryption is used, however, as it is a simple .jar file, no installation is necessary. It allows for text to be directly imbedded into the LSB rather than a file. This is actually quite clever. While this will of course still change both the MD5 signature, it is likely to change it far less.

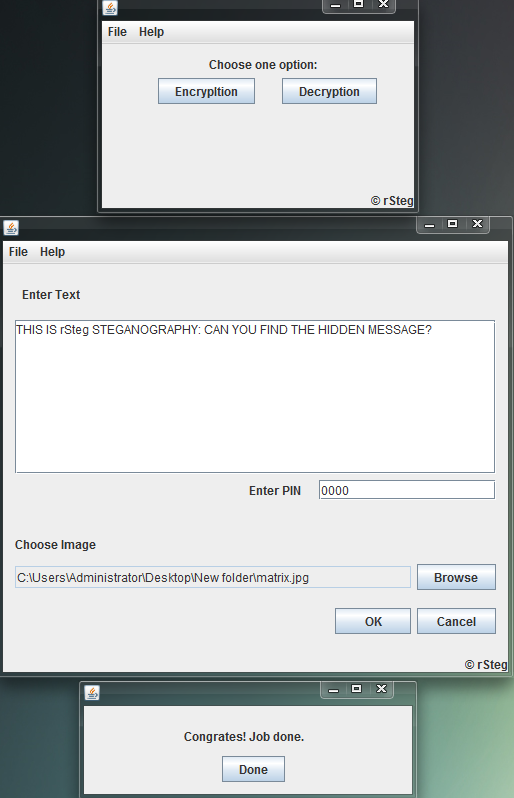


*Here we see the original MD5 hash of the carrier file.*

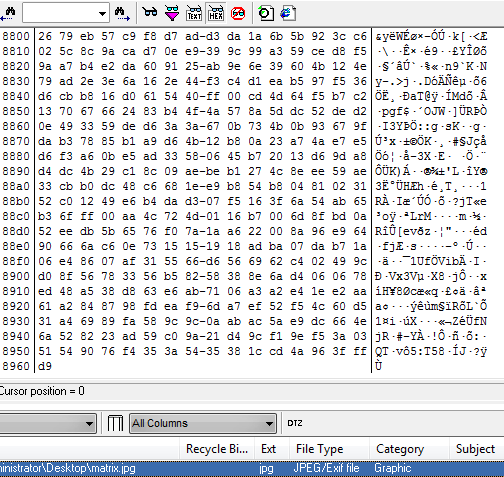
**

*Here we see the MD5 of the new Stego object file (notice the extension has changed).*

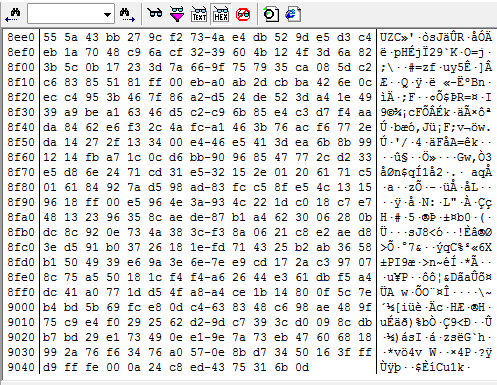
Interestingly, here we see that the procedure has follow the process we have come to expect, even we do not know the exact algorithms and hash types being used. We are also presented a with an option to rename the new stego object file. However, we see our first major deviation. The file extension has changed from .jpg to .png.   
  
Already from the perspective we can consider this a valid signature, however one must consider how this might appear if we did not have access to the information we have such as program used, original file and MD5, knowledge of a hidden message and the contents of said message. We would have, a file, with a different extension, different file name, different MD5 hash, and signature. What we see here is that the change of file extension, while simple in this case, adds a very nice extra layer of obfuscation that would make a blind visual attack on the steganography very difficult indeed.



*Here we see the entire process of rSteg’s injection of text into the carrier file.*



*Here we see the hex value of the original carrier file.*

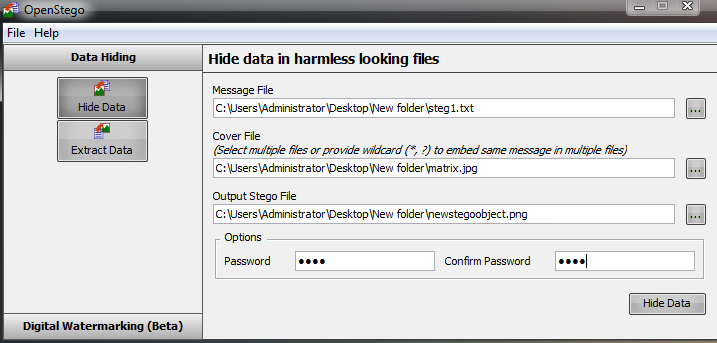
**

*Here we see the hex values of the new stego object (With file extension change).*

While we do see the expected hex value changes, we would expect to see these merely due to the different extension. Here we see the difficulty added by the file extension change. Where previously we could rely on the hex values alone when comparing two files of identical image, extension and MD5 we are now placed in a situation where we suspect a hidden message, but no singular method has provided definitive proof of a message. A blind approach would perhaps fail here, where an analytical method might succeed.

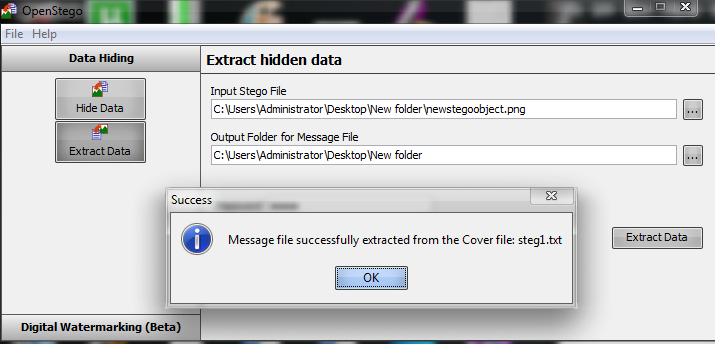
## OpenStego

This program introduces a new feature not yet encountered. The ability to inject the stego object into multiple files at once. This is rather interesting. Consider a folder of images found on a drive, where the same message has been implanted into every image altering the signatures in the exact same way. Would this make it more likely to pass a blind analysis. Or to pose the question another way, if one is looking for a tree in a forest, would one notice the entire forest is made of that exact tree? Certainly, it is an interesting extra layer of obfuscation adding multiple possibilities, even if one simple hide one thousand messages, with only one being the real message and the others being red herrings.

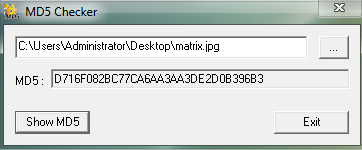


*Here we see the OpenStego injection of the message file into the carrier file. (again, please note the file extension change). Please also note the ability to select multiple cover(carrier) files.*

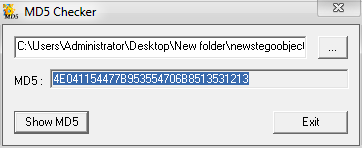
Again, we can see that the file extension has changed. We can also note that digital water marking is employed here. This is a curios choice. Digital watermarking, much as the name suggests, is a method whereby a signature, be it visual, hex or audio is added to a file to allow for identification of the file. This is our first instance of seeing a signature addition that will decrease the effectiveness of steganography once the watermark is commonly known without randomization of the watermarking methodology. In this case, no information is readily available with regard to the methodology of OpenStego’s watermarking.



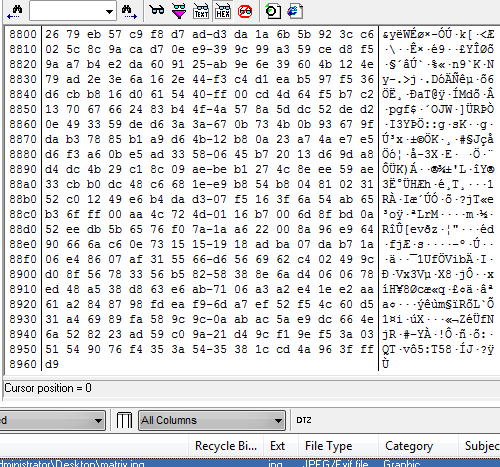
Here we see the OpenStego process reversed with the stego object extracted.



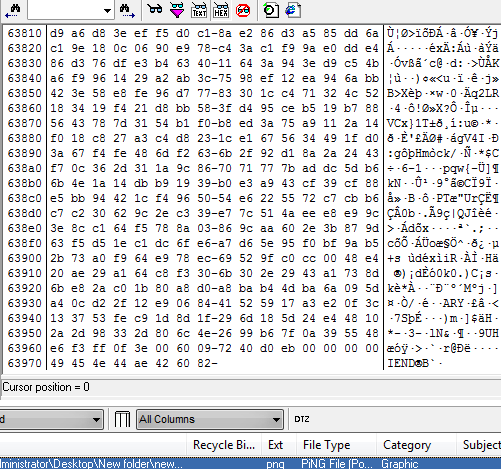
*Here we see the original MD5 hash of the carrier file.*

**

*Here we see the MD5 hash of the new stego object.*



*Here we see the hex values of the un altered carrier file.*

**

*Here we see the hex values of the new stego object*

Again, we see MD5 values, hex values and file signature have changed and we can consider this another successful signature identification and thus, a successful steganalysis.

# Conclusions

Of the four programs examine, I found “Hide’N’Steg” to be the most user friendly, coupled with the easy selection of algorithm, hash and encryption made the process of injecting and extracting when using this program, a breeze. The detection when possessed with all of the information are of course easy, however I feel that if one did not have the knowledge of the hidden message, this would be a very good method of hiding messages.

It is clear now, that the ability to undertake a reliable blind analysis of a steganographic file takes a great deal of knowledge, expertise and most importantly experience. The programs selected for this report show how just one addition feature can alter the signature vastly to the point where a human eye could easily misinterpret the data.  
  
None of the programs examined include the advanced abilities of steganographic programs which can break a file into byte sized chunks, and to imbed the file into multiple locations in one file or even multiple locations across multiple files. One can begin to develop an understanding of the vast complexity of even identifying such a message exists let alone how to reassemble such a message.  
  
What we see therefore, is the importance of establishing and maintaining a database of known signature types and hashes pertaining to as many known steganographic programs, algorithms and injection methods as possible in order to maintain the effectiveness of our automated tools. One should always be striving to develop a good eye for identifying signatures using the blind approach so that ultimately, one can use both the blind approach coupled with the analytical approach to provide the deepest screening of files and most reliable detections of steganography.