Steganography in digital media

Pricope Stefan-Cristian
psir2384@scs.ubbcluj.ro

Suciu Mihai
mihai-suciu@cs.ubbcluj.ro

Abstract

Steganography is the science of concealing a piece of information within another piece of information without affecting the latter in a noticeable way and therefore alerting any intruders of the existence of the former. This paper presents both existing and new ways of embedding computer files and data into different digital multimedia formats as covert as possible while still allowing the encoded information to be retrieved at a later time without any significant losses.

Contents

Chapter 1

Introduction to computer steganography

The practice of hiding and securing information and messages between different parties has played a major part alongside human history, especially during times of war when it was vital that the enemy didn't intercept the strategies and even if they did, they would have no idea what to do with them and would have to dedicate a lot of time, money and personnel to decode the communications. Two of the most famous manifestations of this practice are cryptography and steganography[1].

Trying to differentiate between cryptography and steganography is not difficult, the only thing they have in common is their end goal - making sure that a piece of information sent from one place to another is secured and that only the right recipient will be able to read and understand the received information. The difference lies in the methods they use and the time it takes to reach that goal.

Cryptography focuses on altering the information itself, encrypting it using a key only known by the receiver and sender[1], making it harder for any possible meddlers to alter the meaning of the message or even just understand it. Steganography is more concerned on hiding the fact that there even is any information being transmitted usually by embedding it in something else (hereby referred to as a cover), thus if any interlopers were to actually look at the cover, they wouldn't even be aware of the fact that it contained secrets.

In other words, both methods are meant to be used over an open and unsafe environment, and while cryptography tries to hide the contents of the message but not the fact that there is a message being sent, steganography tries to hide the communication altogether. The main advantage of these 2 methods is that they are not exclusive, they can be used together for maximum safety of the information.[2]

With the evolution of the Internet and the increasing usage of personal computers in day to day activities we needed to secure the data sent over the network between the users. The efforts were lead by cryptographers who developed thousands of algorithms for encrypting the information and the very few remarkable ones are still being used[3]. But that doesn't mean steganography was left behind, researchers developed plenty of new algorithms for hiding information using digital files as a cover and the Internet as the transmission environment.

In theory all types of digital files can be used as cover - shared libraries and executables can be edited to include the hidden data and then be recompiled/rebuilt, text files and documents could be modified to enclose the information in secret places/paragraphs, and multimedia such as images, music and video files can be changed to carry the digital data into their metadata, pixels, audio frequencies, motion vectors and much more. This paper will go into details about some of the most popular formats used for multimedia files and showcase their internal structure and document techniques used in steganography.

[1]This is only true in symmetric cryptography and is a gross oversimplification of cryptography as a science, but it is just meant to get a point across and help differentiate between the 2.

[2]The speed of encoding/decoding the information is greatly decreased when these 2 are combined which is the reason that nobody merges them, preferring to juse use encryption to keep the information safe.

[3]The reason cryptography is in the spotlight is because it is much faster, mathematically proven, and it hides all the data without leaving anything in plain sight(like steganography does with the cover).

Chapter 2

Steganography methods

2.1 Least Significant Bit(LSB)

2.1.1 Sequential

Least Significant Bit or LSB is by far the most used method when talking about any type of steganography. Given that the smallest unit a computer can understand and process is usually a byte, altering only the least significant bit will not change the transmitted information in a noticeable way to any external parties. It is much easier to showcase what a byte contains and what the LSB change implies and how it works. A byte contains 8 bits, so this means that the values a byte can take range anywhere from 0 to 255 (inclusive)[1]. Let's assume that we have an array of 4 random values in consecutive memory : 217, 127, 100, 62 (all values are in decimal), each stored on exactly one byte, and that we want to hide our grade in Numerical Analysis from our parents (in this case a 3) using a LSB substitution. The process would be something like this :

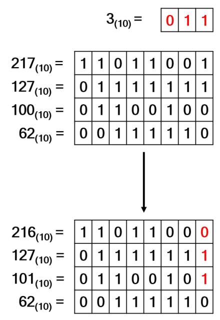


Figure 2.1: How the sequential Least Significant Bit change works

As we can see from Figure 2.1, we have successfully altered the least significant bit of the first 3 bytes of the stream in order to hide our grade : 217 became 216 when we changed the last bit from 1 to 0, 127 was unchanged because it already had the last bit set, and the third byte became 101 after toggling the final bit. Furthermore, the rest of the stream (the fourth byte, 62) was not affected because we already hid the entirety of our secret message. While this is great because we only hide exactly as much as we need and not a byte more, we have a high risk of corrupting the hidden message in case our cover image gets compressed or loses even a single byte when sent over a network. Basically, we are trading data redundancy in order to get simplicity and efficiency.

Sequential LSB insertion is the simplest and most common way of embedding any kind of information into a byte stream that is then shared. It has been thoroughly discussed by a great deal of researchers and has been the subject of many papers where it was analyzed and benchmarked[6][8] . Being the most popular technique also means that any flaws the method has are widely documented and showcased. Steganalysis[2] performed on outputs created using this algorithm has shown that it is unreliable to stay undetected if an outside party intercepted the message containing the cover file[9]. Furthermore, doing a simple reverse engineering on the algorithm reveals even more issues with this naive encoding : if a single bit that is part of the secret message was flipped from the cover file byte stream, the message would become corrupted and the orig-

[1]This is the case for unsigned bytes, but given that we are talking about a method that only deals with the least significant bit, we can safely ignore the most significant bit, also known as the sign bit.

[2]Steganalysis is the study of steganographic methods, including but not limited to : differences in file sizes or in color histograms, secret message redundancy, embedding capacity and performance etc.

inal secret would be lost forever. This means that sequential LSB insertion is not resistant at all to any form of lossy compression where some of the original data may be lost in order to reduce the used disk space because it would lose most, if not all, of the embedded file information.

Furthermore, it is extremely easy to compute the carrier storage, i.e. how many bytes we are able to hide into the cover file or in other words, the maximum size of the secret message that we can succesfully embed without losing anything while still keeping a covert profile. Assuming CDS or Cover Data Size (how many bytes are actually used to store the pixel information, no metadata information or chunk headers or anything like that), then the MMS or Maximum Message Size would be equal to

$$MMS = [CDS \ / \ 8] \ bytes$$

or the integer part of CDS divided by 8. This should come as no surprise since sequential LSB is altering 1/8 (an eighth) of each byte when embedding a bit of the secret information so the maximum capacity makes sense to also be 1/8.

2.1.2 Scrambling

As documented earlier, sequential LSB insertion algorithm has a decent number of flaws so it was needed to develop some new techniques that are not relying as much on the cover file not losing a few bytes or undergoing a compression algorithm. In other words, it needed to introduce a few redundancies to ensure that the secret message wouldn't be lost as easily and that the message was not written in a sequential and direct order. They achieved this by not just changing the least significant bit in a sequential order, but by writing in an apparently random order (in simpler terms, scrambled) that could be reproduced by having the right key or by using the same algorithm in order to retrieve the embedded information. In this subsection we will discuss a few of the most common scrambling techniques and introduce a new one as well.

The most popular methods used for scrambling secret messages into various covers usually choose to simply ignore the entire data stream and only focus on a specific subsection and choose that as the carrier environment. After a smaller subsection is chosen (it can still be the entire actual data part of the cover, it's not an actual rule), we will have to generate the order in which we will write the message information. As mentioned before, this is derived using a key known only to the sender and the receiver that can be shared between the 2 parties using another transmission environment, preferably one that is encrypted and safe. The main logic is to use that passkey as a seed in a valid pseudo random engine when generating the order so that there are no collisions and that only the right key will produce the right order.

There is also an option for when there is no safe method of transmitting a key and that is to scramble the secret message within a certain order that is not sequential. But as long as the receiver and transmitter have no way of communicating the algorithm used in embedding the message (can also assume this because they have no way of sharing the key), this option is useless but is still interesting to look into because they are a variation of the other mentioned option. Having no key to generate the order makes this option easier to showcase(since we don't have to also simulate a pseudo-random engine and a seed). Let's assume we have a byte stream of random data[3] and we want to again hide a grade from our parents, in this case a 6 (or 110 in binary). Instead of hiding the grade sequentially, the bits will be hidden into the last bit of every third byte and it will end up looking something like this :

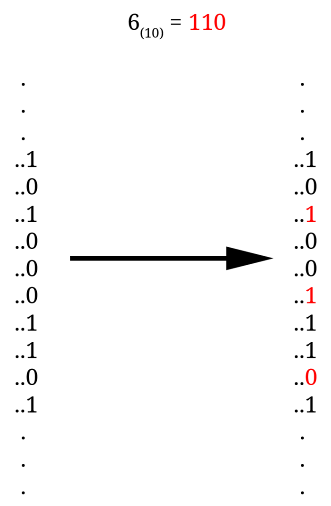


Figure 2.2: An example of scrambling LSB insertion

The key difference from sequential LSB insertion is that as long as the sender and receiver are aware of the used algorithm, any external parties will not be aware of the secret embedded message. Further-

[3]Now the actual data meaning can be safely ignored because we only work on the LSB and we have seen in the previous chapter it does not alter the data in a significant way such that an intruder might notice something is wrong.

more, this method has proven very useful because there are infinite ways of scrambling a message into the cover file without alerting any possible intruders and it ends up being an extremely hard guessing game for the attackers in their goal to extract the information.

The carrier capacity appears initially to be the same, but it is very important to remember that the scrambling algorithm used will usually work on a subset of the cover data bytes, not the entirety of it (just like in figure 2.2 we used only each third byte to store the information). Using the same notations as in the previous chapter we get that

$$MMS \leq [CDS \ / \ 8] \ bytes$$

so in most cases, the scrambled MMS will be smaller than the sequential MMS. However it is very important to note that this decrease in size comes with a great increase in data security and message safety.

This paper also introduces a new type of scrambling algorithm created by the authors that only works on lossless image formats that do not use any kind of interlacing when rendering the picture. It relies on scrambling the secret message into the image sub-blocks in a specific order that can only be deduced by having the right passkey. More information on this method is presented in chapter 3.2.1 after the introduction of image basics.

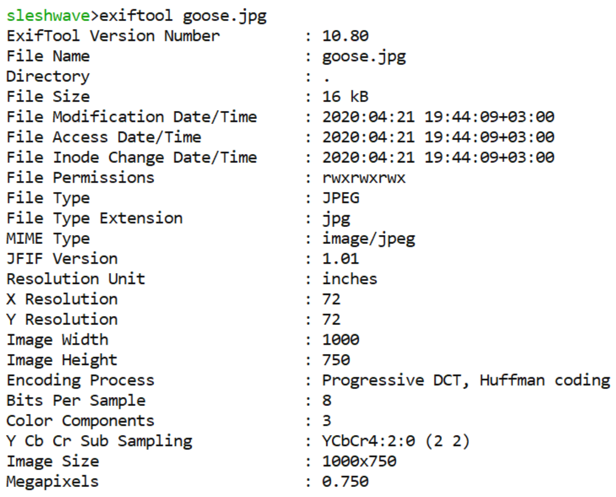
2.2 Metadata encoding

The word metadata was formed from combining the word "data" with the prefix "meta-" and is used to describe a special type of data that has information about other types of data[3]. In simpler terms, it means "data about data" and it keeps the same meaning in the digital world. It is much easier to visualize and understand the concept of metadata using a simple example : let's take a picture stored on a computer.

Figure 2.3: A simple image file

As noted in the introductory chapter, every file can be seen as a byte stream. However it is very important to keep in mind that those bytes don't represent only the image data, the pixels seen on the screen, they are much more. They also contain information about the camera used to take the photo, the location where it was taken, when the file was created, when it was modified, etc. All of the aforementioned information forms the metadata. It varies from file format to file format where they store this information in the byte stream, how many bytes are allocated for each piece of information or if it even has any effects on the actual file data. Most of the time metadata fields are only parsed by the renderer software and are mostly hidden to the user, but there are a few ways of viewing the information:

- using a hex editor to view the raw bytes of the file and then mapping those bytes to the publicly available file format specifier - an international approved paper which specifies the meaning of the bytes in the file binary stream
- using the operating system to view more properties about the file, not just the actual data interpreted and displayed to the end user
- using a third party tool which already knows the mapping and meaning of each sequence of bytes in the file binary stream and can successfully parse the metadata



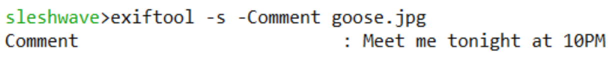
```
sleshwave>exiftool goose.jpg
ExifTool Version Number         : 10.80
File Name                       : goose.jpg
Directory                       : .
File Size                       : 16 kB
File Modification Date/Time     : 2020:04:21 19:44:09+03:00
File Access Date/Time           : 2020:04:21 19:44:09+03:00
File Inode Change Date/Time     : 2020:04:21 19:44:09+03:00
File Permissions                : rwxrwxrwx
File Type                       : JPEG
File Type Extension             : jpg
MIME Type                       : image/jpeg
JFIF Version                    : 1.01
Resolution Unit                 : inches
X Resolution                    : 72
Y Resolution                    : 72
Image Width                     : 1000
Image Height                    : 750
Encoding Process                : Progressive DCT, Huffman coding
Bits Per Sample                 : 8
Color Components                : 3
Y Cb Cr Sub Sampling            : YCbCr4:2:0 (2 2)
Image Size                      : 1000x750
Megapixels                      : 0.750
```

Figure 2.4: Example of using a tool to read file metadata

The focus of this sub-chapter will be on the metadata fields that don't necessarily have any important effect on the data representation and theoretically could be altered, such as any comments from the author or any contact information. In the case of an image, changing important metadata fields such as the width or height of the picture are not very covert methods of embedding any information at all, proving that not all fields are equally important. We are left with the more *useless* metadata fields, but the good news is that most file formats allow these fields to have a variable size which is perfect for any steganographic purposes because it removes the size constraint of the secret message. While in theory this allows for ∞ MMS, there are a few limitations in place that significantly reduce that number:

- computers have a limited amount of storage space, in most modern day computers that would be about one terabyte. This means that the MMS will certainly be smaller than that.
- most file formats that allow metadata fields to be embedded in the byte stream of the file by an external party also have a sequence of bytes that indicates how long that metadata field is (how many bytes it contains). In most cases that sequence is 4 bytes long so this usually results in a MMS of $2^{4*8} = 2^{32} \approx 4.29\ gigabytes$.
- in the pursuit of keeping the existence of a secret inside the cover file as covert as possible, the MMS is again limited by the CDS. This happens because the size of the cover file is almost always displayed to any parties without any interactions and it needs to not raise any suspicions or attract unwanted attention to the actual contents of the file. To give an example, it would be very weird to see a simple image in FULL HD resolution have a size of two gigabytes and will almost certainly alert any intruders. It is extremely hard to say an exact MMS based on this limitation because it depends on the original cover file size and it should be relative to that number. It is recommended that $MMS \leq 50\% * CDS$.

In the end it is important to note that metadata secret encoding is the most trivial method of embedding secret information into different kinds of cover files. However, this method comes with a great cost: almost every single tool that specializes in extracting metadata will be able to detect and identify our message without too much hassle because the final cover file still has to respect the format specification.



```
sleshwave>exiftool -s -Comment goose.jpg
Comment                         : Meet me tonight at 10PM
```

Figure 2.5: Example of message hidden in a file in the metadata section

2.3 Unused space embedding

Usually most file formats and even internationally approved and used protocols such as the TCP/IP have unused bits or bytes in their composition, either for future proofing the format in the form of reserved space or simply to serve as padding space mainly present to help the associated parsers.

One well known example of this is found in the BMP file format which appears only in the cases where the width of the image is not a multiple of 4. The standards specify that in this situation there needs to be added the right amount of bytes to fill so that the width will be divisible by 4. So if an image has a resolution of 1920x1080 (the width is the first number), there won't be a need for any padding bytes because $1920\%4 = 0$. If it had a resolution of 125x125, since $125\%4 \neq 0$ there would be $4 - (125\%4) = 4 - 1 = 3$ bytes to serve as a padding for each pixel line, ending up in a total of $125*3 = 375$ bytes of simple, raw, unused space that most BMP parsers will ignore because they know that data is meaningless.

Another very good example is in the case of the IPv4 packet header which has a reserved bit in the flags subsection, or to be more precise, the 49th bit. According to RFC3514[4] it is meant to be a security bit representing the true intent of the packet (GOOD or EVIL), however it is important to note that it was an April Fools informational memo and it is most likely that altering this bit will still go undetected by any kind of firewall or intrusion prevention system.

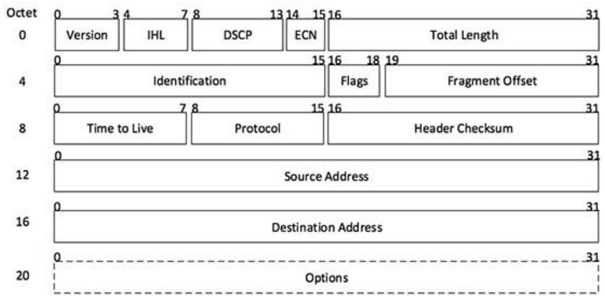
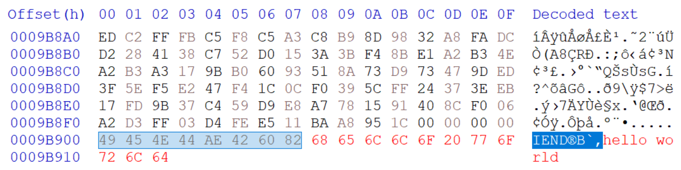


Figure 2.6: The structure of an IP packet

Given the IP packet header specifier represented in Figure 2.6 it becomes trivial to alter the first bit of the flags section to the current bit of a secret and then over multiple sent packets to actually send the message in one of the most covert ways possible. Furthermore, it is very important to note the contribution done by Kamran Ahsan and Deepa Kundur in their 2002 article entitled "Practical Data Hiding in TCP/IP"[5] because they also achieve the ability to send covert information by taking it one step further: using IP packet headers that may actually even be in use and by using Chaos Theory in the values of the Identification field sent over.

However, there is still one very important spot left in a file's byte stream where the secret message could easily be hidden: at the end of it. Most file formats either have specially crafted headers that announce the end of the file[4] or have a few bytes reserved at the beginning of the file that mark the length of the data[5]. It is important to note that both of these methods basically produce the same result technically speaking: a superior margin, a maximum amount of bytes that are to be read and interpreted by the parsers that contain the data relevant to the current file. After that region has ended, it is a free-for-all memory territory for secrets to be embedded. The same limitations from the previous chapter still stand, it is recommended that $MMS \leq 50\% * CDS$ in order to remain as covert as possible while still exfiltrating data.



```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text

0009B8A0  ED C2 FF FB C5 F8 C5 A3 C8 B9 8D 98 32 A8 FA DC  íÂÿûÅøÅ£È¹.˜2¨úÜ
0009B8B0  D2 28 41 38 C7 52 D0 15 3A 3B F4 8B E1 A2 B3 4E  Ò(A8ÇRÐ.:;ô‹á¢³N
0009B8C0  A2 B3 A3 17 9B B0 60 93 51 8A 73 D9 73 47 9D ED  ¢³£.›°`"QŠsÙsG.í
0009B8D0  3F 5E F5 E2 47 F4 1C 0C F0 39 5C FF 24 37 3E EB  ?^õâGô..ð9\ÿ$7>ë
0009B8E0  17 FD 9B 37 C4 59 D9 E8 A7 78 15 91 40 8C F0 06  .ý›7ÄYÙè§x.'@Œð.
0009B8F0  A2 D3 FF 03 D4 FE E5 11 BA A8 95 1C 00 00 00 00  ¢Óÿ.Ôþå.º¨•.....
0009B900  49 45 4E 44 AE 42 60 82 68 65 6C 6C 6F 20 77 6F  IEND®B`‚hello wo
0009B910  72 6C 64                                         rld
```

Figure 2.7: Simple example of hiding a note after a file has ended

[4]One good example is the PNG format which has an IEND header and a couple of other information to mark that the image data has ended and that there is nothing left that is relevant to the parser. More information regarding this format is available in chapter 3.4

[5]This method is found in BMP or WAV file formats usually, it is very simple and easy to work with for parsers. More details in later chapters

Chapter 3

Image file formats and steganography techniques

3.1 Introduction

An image is a two-dimensional representation depicting any possible subject conceivable by human imagination, captured using an optical device (such as a camera or a telescope) or a natural object (human eyes). The image can then be rendered and displayed for other people to see either manually (by painting, carving etc.) or automatically (by using a computer). In this chapter we will focus on images captured using digital optical devices that are rendered automatically. The correct term for them is digital images, but throughout the rest of the paper they will be reffered to as images for convenience.

Figure 3.1: Lenna - Classic example of a digital image

Computers are programmed to do operations in a clear sequential way and this rule doesn't change when working with pictures. In order for a computer to be able to render an image, it needs to know some general metadata information about the photo, such as the width and height, as well as the data bytes of the image. These bytes are the actual representation of the picture which compose the two-dimensional pixel map[1]. A pixel is the smallest unit that a computer monitor can read and display. The pixel color is the result of merging the different color channels which compose the picture (such as RGB, YUV, YCbCr etc.). Here is an example of the entire process - let's assume that from the image data bytes the first 3 bytes have the decimal values 20, 127, 250 and that it uses the RGB color model. This means that when the computer will have to render the image, the first pixel will have the red component equal to 20 (0x14), the green equal to 127 (0x7F), and the blue equal to 250 (0xFA), in what will finally be interpreted as #147FFA by the monitor (variation of light blue).

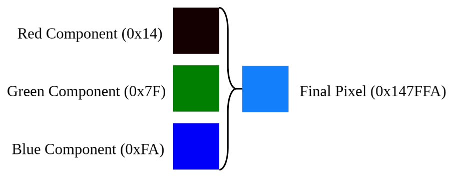


Figure 3.2: How 3 colors channels build the pixel

With all of this information in mind, we can now procede to discussing the different steganography methods possible when working with images as cover.

[1]This is true for a lossless format, where each pixel is stored in memory. It is not exactly the case for lossy formats such as JPEG where the image goes through processing before being rendered. More information later in the chapter.

3.2 Additional techniques used in image stenography

3.2.1 Image sub-block scrambling

3.3 BitMap Picture (BMP)

3.4 Portable Network Graphics (PNG)

3.5 Joint Photographic Experts Group (JPEG)

Chapter 4

Audio file formats and steganography techniques

4.1 Introduction

4.2 Additional techniques used in audio steganography

4.2.1 Embedding within certain frequencies

4.3 The MPEG-1/2 Audio Layer III (MP3)

4.4 Waveform Audio (WAV)

Chapter 5

Video file formats and steganography techniques

5.1 Introduction

5.2 Additional techniques used in video steganography

5.2.1 Motion Vectors

5.3 MPEG-4 Part 14 (MP4)

5.4 Audio Video Interleave (AVI)

Chapter 6

Conclusion

Chapter 7

Bibliography

Bibliography

[1] Petitcolas FAP, Anderson RJ and Kuhn MG. *Information Hiding - A Survey.* Proceedings of the IEEE, special issue on protection of multimedia content, 1999.

[2] Merriam-Webster dictionary.
`https://www.merriam-webster.com/dictionary/steganography`

[3] Merriam-Webster dictionary.
`https://www.merriam-webster.com/dictionary/metadata`

[4] S. Bellovin *The Security Flag in the IPv4 Header.*
`https://tools.ietf.org/html/rfc3514` AT&T Labs Research, 1 April 2003

[5] Johnson Neil and Jajodia Susil. *Exploring Steganography : Seeing the Unseen.* Los Alamitos, IEEE Computer Society, 1998.

[6] Kamran Ahsan, Deepa Kundur. *Practical Data Hiding in TCP/IP.* Workshop on Multimedia Security at ACM Multimedia, 2002.

[7] T. Morkel, J.H.P. Eloff and M.S. Olivier. *An overview of image steganography.* University of Pretoria, ICSA Research Group, 2005.

[8] Niels Provos and Peter Honeyman. *Hide and Seek: An Introduction to Steganography.* University of Michigan, IEEE Computer Society, 2003

[9] Andreas Westfeld and Andreas Pfitzmann. *Attacks on Steganographic Systems.* International workshop on information hiding, 1999