A New Approach for Hiding Data within Executable Computer Program Files Using an Improvement Cover Region

Dr. AbeerTariq[1] EkhlasFalih[1] Eman Shaker[1]

[1]Computer Science Department, University of Technology, Baghdad

email: abeer282003@yahoo.com ekhlas_uot1975@yahoo.com eman_uot1974@yahoo.com.edu.iq

Received: 27/1/2013

Accepted: 22/7/2013

Abstract – The process of hiding is considered to be one of the important security branches which are used to ensure the safe transfer of vital data and its protection against theft or editing. Recently, the need for developed hiding methods have greatly increased as the programming technologies that are capable of detecting hidden data, which are hidden using conventional methods, have undergone a great leap in progress. This paper aims to introduce a novel method of hiding a text inside an executable (.exe) file. This is considered one of the recent, more developed types. The suggested system has been tested on two types of executable files. The first type was written using (C++.exe) whereas the second was written using (VB.exe). The text was hidden using the suggest method (Cover Region and Parity bits) through which the executable file will be segmented into (regions) afterwards, each group of values will be tested alongside the value of the secret message that needs to be hidden. The suggested approach was applied on several texts and the result was that hiding via the executable files of the (VB) types is faster and it occupies less size than the (C++) type..

Keywords –C++ execution file,VB execution file and Cover Region and Parity Bits.

1. Introduction

Internet communication has become an integral part of the infrastructure of today's world. The information communicated comes in numerous forms and is used in many applications. In a large number of these applications, it is desired that the communication be done in secret. Such secret communication ranges from the obvious cases of bank transfers, corporate communications, and credit card purchases, on down to a large percentage of everyday email. With email, many people wrongly assume that their communication is safe because it is just a small piece of an enormous amount of data being sent worldwide. After all, the Internet is not a secure medium, and there are programs "out there" which just sit and watch messages go by for interesting information [1].

1. Steganography and encryption are both used to ensure data confidentiality. The main difference between them is that with encryption anybody can see that both parties are communicating in secret. Steganography hides the existence of a secret message and in the best case nobody can see that both parties are communicating in secret. This makes steganography suitable for some tasks, such as copyright marking, for which encryption isn't. Adding encrypted copyright information to a file could be easy to remove but embedding it within the contents of the file itself can prevent it from being easily identified and removed [2].
2. There is one group of files that vary enormously in size and are usually rather difficult to examine in detail because they are comprised of compiled computer codes which are executable, or exe, files. These files tend to contain lots of what might be described as "junk data" of their own as well as internal programmer notes and identifiers, redundant sections of code and infuriatingly in some senses coding "bloat." All of this adds up to large and essentially random file sizes for exe files. As such, it might be possible to embed and hide large amounts of data in encoded form in an exe file without disrupting the file's ability to be executed, or run, as a program but crucially without anyone discovering that the exe file has a dual function [3].

2. Theoretical Background

Steganography can be split into two types: Fragile and Robust. The following section describes the definition of these two different types of steganography.

2.1 Pure steganography: -A steganographic system which does not require prior exchange of some secret information (like a stego key) is pure steganography. The embedding process can be described as a mapping

$$E: C \times M \rightarrow S$$

where **C** is the set of possible covers and **M** is the set of possible messages.

The extracting process consist of a mapping

$$D: S \rightarrow M$$

extracting the secret message out of a cover .

2.2 Secret key steganography: -In secret key steganography, the sender chooses cover **C** and embeds the secret message into **C** using secret key **K**. If the key in the embedding process is known to the receiver, he can reverse the process and extract the secret message. The mapping of the process if K is the set of secret keys.

$$E_k: C \times M \times K \rightarrow S$$

And

$D_k: S \times K \rightarrow M$

with the property that $D_k(E_k(c,m,k),k)=m$ for all $m \in M, c \in C$ and $k \in K$ is called a secret key steganography system

2.3 Public key steganography:- Public key steganography system require the use of two keys, one is private and the other one is public.The public key is stored in a public database whereas the public key is used in the embedding process. The private key is used to reconstruct the secret message [4].

3. Media Used for Hiding

There are different media for hiding data:-

3.1 Data hiding in images

In a computer, images are represented as arrays of values. These values represent the i3.ntensities of the three colors R(ed) G(reen) and B(lue), where a value for each of the three colors describes a pixel.

Through varying the intensity of the RGB values, a finite set of colors spanning the full visible spectrum can be created. In an 8-bit gif image, there can be 2^8 = 256 colors and in a 24-bit bitmap, there can be 2^{24} = 16777216 colors.

Information can be hidden in many different ways in images. Straight message insertion can be done, which will simply encode every bit of information in the image. More complex encoding can be done to embed the message only in noisy areas of the image that will attract less attention. The message may also be scattered randomly throughout the cover image [4].

3.2 Hiding in Audio

General principles of data hiding technology, as well as terminology adopted at the First International Workshop on Information Hiding, Cambridge, U.K. are illustrated in Figure (3.1).

A data message is hidden within a cover signal (object) in the block called embeddor using a stego key, which is a secret set of parameters of a known hiding algorithm. The output of the embedded is called stego signal (object). After transmission, recording, and other signal processing which may contaminate and bend the stego signal, the embedded message is retrieved using the appropriate stego key in the block called extractor [5].

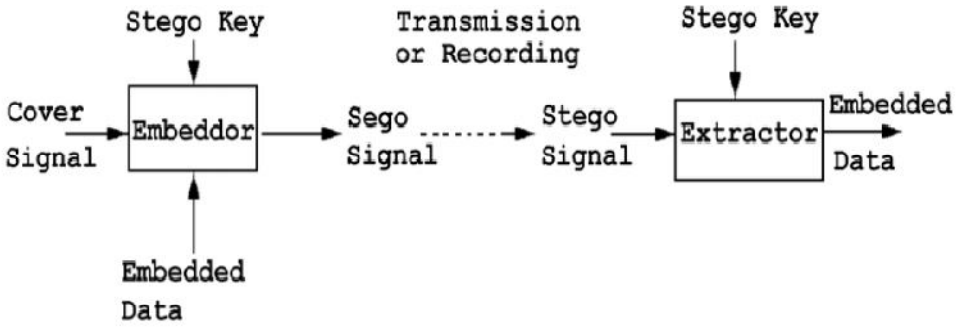


Figure (1) Block diagram of data hiding and retrieval.

A number of different cover objects (signals) can be used to carry hidden messages. Data hiding in audio signals exploits the imperfection of the human auditory system known as audio masking. In thepresence of a loud signal (masker), another weaker signal may be inaudible, depending on spectral and temporal characteristics of both masked signal and masker [5].Masking models is extensively studied for perceptual compression of audio signals. In the case of perceptual compression the quantization noise is hidden below the masking threshold, while in a data hiding application the embedded signal is hidden there. Data hiding in audio signals is especially challenging, because the human auditory system operates over a wide dynamic range. The human auditory system perceives over a range of power greater than one billion to one and a range of frequencies greater than one thousand to one.

Sensitivity to additive random noise is also acute. The perturbations in a sound file can be detected as low as one part in ten million (80 dB below ambient level).However, there are some “holes” available. While the human auditory system has a large dynamic range, it has a fairly small differential range. As a result, loud sounds tend to mask out quiet sounds.

Additionally, the human auditory system is unable to perceive absolute phase, only relative phase. Finally, there are some environmental distortions so common as to be ignored by the listener in most cases [6]. Now we will discuss many of these methods of audio data hiding technology.

4. Basic Media (Cover) and Techniques Used In Proposal

The execution file used as a cover, and Cover Region and Parity Bits technique used as technique in our proposal:

4.1 Executable File

An executable file is a file that is used to perform various functions or operations on a computer. Unlike a data file, an executable file cannot be read because it has been compiled. On an IBM compatible computer, common executable files are .BAT, .COM, .EXE, and .BIN. Depending on the operating system and its setup, there can also be several other executable files [7].

To execute a file in MS-DOS and numerous other command line operating systems, type the name of the executable file and press enter. For example, the file myfile.exe is executed by typing myfile at the prompt.

Other command line operating systems such as Linux or Unix may require the user to type a period and a forward slash in front of the file name, for example, ./myfile would execute the executable file named myfile.

To execute a file in Microsoft Windows double-click the file.To execute a file in other GUI operating systems, a single or double-click will execute the file [7].

4.2 EXE file formats

There are several main executable file formats [9]:

4.2.1 DOS

16-bit DOS MZ executable: Being the original DOS executable file format, these can be identified by the letters "MZ" at the beginning of the file in ASCII.

16-bit New Executable: Introduced with Multitasking MS-DOS 4.0, these can be identified by the "NE" in ASCII. They never became popular or useful for DOS and cannot be run by any other version of DOS, but can usually be run by 16/32-bit Windows and OS/2 versions.

4.2.2 OS/2

32-bit Linear Executable: Introduced with OS/2 2.0, these can be identified by the "LX" in ASCII. These can only be run by OS/2 2.0 and higher. They are also used by some DOS extenders [8].

Mixed 16/32-bit Linear Executable: Introduced with OS/2 2.0, these can be identified by the "LE" in ASCII. This format is not used for OS/2 applications anymore, but instead for VxD drivers under Windows 3.x, Windows 9x, and by some DOS extender [8].

4.2.3 Windows

32-bit Portable Executable: Introduced with Windows NT, these are the most complex and can be identified by the "PE" in ASCII (although not at the beginning; these files also begin with "MZ"). These can be run by all versions of Windows and DOS (DOS runs the MZ section; Windows runs the NE or PE section). Using HX DOS Extender, DOS can load the NE and PE sections. They are also used in BeOS R3, although the format used by BeOS somewhat violates the PE specification as it doesn't specify a correct subsystem. These can also be used on React OS[8].

64-bit Portable Executable (PE32+): Introduced by 64-bit versions of Windows, this is a PE file with wider fields. In most cases, you can write a code that simply works as both a 32 and 64-bit PE file [8].

5. Proposed method to hide data in exe file

The proposed method used to hide data in exe file format is bits as illustrated bellow:

5.1. Cover Region and Parity Bits

A cover-region is any non-empty subset of the cover c = {c1, …, c*l(c)*}.The idea is to generate a pseudorandom sequence of disjoint cover regions, using a stego-key as the seed, and store only one bit of the secret message in a whole cover-region rather than in a single element. The secret bit to be hidden inside a cover-region is embedded as the parity-bit *p* (*I*) for the cover-region *I* chosen.

$$p(I) = \sum_{j \in I} LSB(c_j) \bmod 2 \quad \ldots (1)$$

During the embedding step, *l(m)* disjoint cover-regions *Ii* ($1 \leq i \leq l(m)$) are selected, each encoding one secret bit *mi* in the parity bit *p*(*Ii*).

If the parity bit of the cover-region *Ii* does not match with the secret bit *mi* to encode, one LSB of a randomly chosen cover-element in *Ii* is flipped. This will result in *p*(*Ii*) = *mi* .During the extraction process at the receiver, the parity bits of all the selected cover-regions (generated according to the pseudo randomsequence) are calculated and lined up to reconstruct the message .

Example

Suppose the LSB of the execution block C_i is as follows (suppose the block size is **9** bits)

C1								
1	0	0	1	1	1	0	1	0

C2								
0	0	1	0	1	1	1	1	1

C3								
1	1	1	0	0	0	1	1	0

To embed 011=s1 s2 s3

$$S1 = \sum_{i=1}^{9} LSB\ (C\ 1) \bmod 2 \quad \ldots.2$$

5 Mod 2 = 1 = s1

S1 does not match equation 2.Select the middle location (5) in C1 the LSB of c1 is 1,and so complement it to 0 and it becomes

C1								
1	0	0	1	0	1	0	1	0

$$S2 = \sum_{i=1}^{9} LSB\,(C\,2) \bmod 2 \quad \ldots.3$$

6 Mod 2 = 0 = s2

S2 does not match equation3. Select the middle location (5) in C2 the LSB of c2 is1,and so complement it to 0 and it becomes

C2								
0	0	1	0	1	1	1	1	1

$$S3 = \sum_{i=1}^{9} LSB\ (C\ 3) \bmod 2 \quad \ldots 4$$

Since s3 matches equation 4, the values of C3 are not affected

Algorithm (1) Embedding Process

Input: - Execution cover file (E), binary secret message S.
Output :- Stego execution file

Step1:-
1) Convert the execution cover file (E) into binary data and put the result in (BE).

2) Partition binary data (BE) into blocks, each of size 9 bits contains the LSB of (BE) and put the result in (C).

3) Calculate the number of blocks and put the result in (N).

4) Set M to the length of secret message.

Step2:- if M > N then return error message" cover is small to hide the message", stop.

Step 3:- 1) set i to 1,set j to 1
2) while (i<=N) and (j<=M) do
Begin

If $S_j = \sum_{K=i}^{k=i+9} C_k \bmod 2$ then

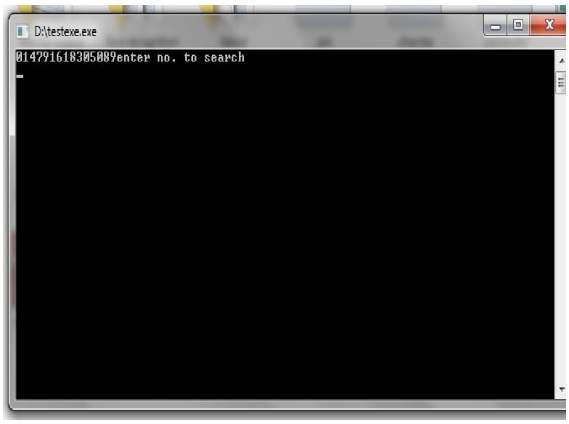
keep C_k unchanged
Else
Select location 5 in C_k ,flipped it.
End if
i=i+9
j=j+1
End while.

Step4:- End.

6. Experimental Results

In this section, to explain the implementation of the proposed system, we will focus on displaying the basic differences of .exe files for both files C++ and VB.

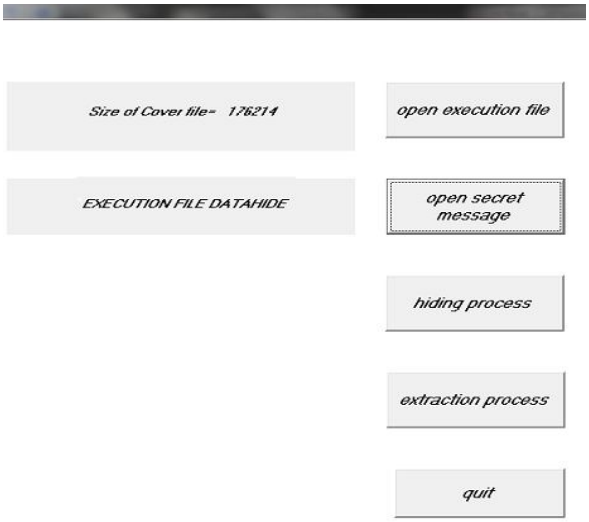
Example 1:Suppose the cover is C++execution file as shown in figure(2):



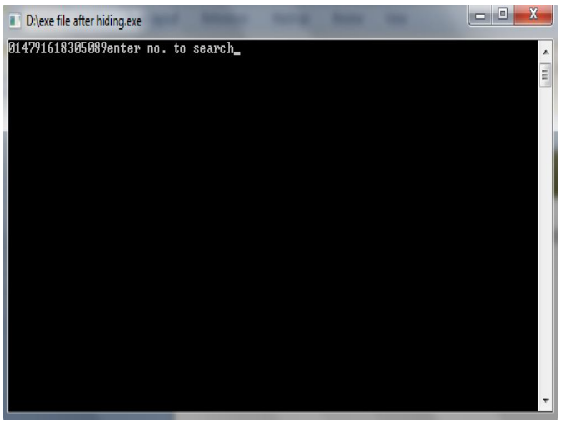
Figure(2):C++ cover file

Figure(3): size of C++ cover file

The form that displays the size of a C++ file, which had been displayed in figure (2) before the hiding operation, is displayed in figure (3) with all its cases.



Figure(4): load secret message

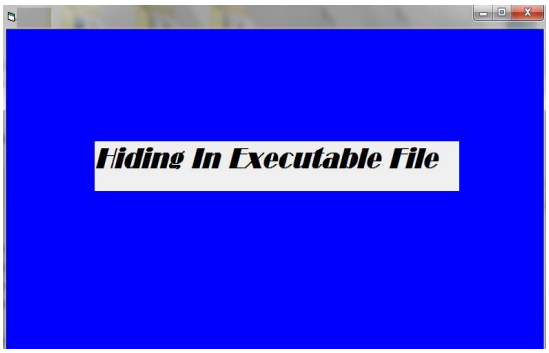


Figure(5):C++ stegonagraphy file after hiding the message

Figure (6): Size of Steganography file

Figure(7): Secret Message after extraction

Example 2: Suppose the cover isVB execution file as shown in figure(8)



Figure(8): VB cover file

Figure (9): form shows hiding and extraction in VB execution file

The following table illustrates the comparison between **C++.exe** and **VB.exe**

Table 2 describes the comparison between C++ and Visual Basic Execution file

	Visual Basic	C++
Performance	very good	Good
Size of .exe	Excellent	Excellent
Complexity	complex	More complex
Security	High security	Highest security
Speed of Extraction	High speed	Low speed
Capacity of data	Low capacity	High capacity

7. Conclusion

This research reached the following points:

1. The performance in **VB** is much more efficient than its **C++** counterpart.
2. The size of the execution file, in both languages, after adding the secret message isn't greatly changed which prevents hackers from noticing any suspicious behavior in the file.
3. **C++** is more complex than its counterpart in regards to the hiding procedure.
4. The **C++** enjoys the highest level of security due to its complex nature.
5. Due to the complex nature of **C++** in terms of security; the speed of extraction is thus lowered, taking a larger amount of time than the **VB.**
6. Capacity of data is higher in **C++** than in **VB.**

8. Recommendations

The proposed system can suggest the following re commendations

1) It can apply the proposed system to another type of execution file format such as windows files.
2) It can encrypt the text before embedding with another type of hiding technique.
3) It can mix more than one technique for hiding.

References

1] J. Johnston and K. Brandenburg, "*Wideband Coding Perceptual Consideration for Speech and Music*". Advances in Speech Signal Processing, S. Furoi and M. Sondhi, Eds. New York: Marcel Dekker, 2000.

[2] S. Katzenbeisser, And F. Peticolas ,"*Information Hiding Techniques For Steganography And Digital Watermarking*", Artech House, USA, 2000.

[3] R. Meyer and Bryan, "*Implementation And Evaluation Of The Least Significant Bit (LSB) Replacement Steganographic Technique*" , Pittsburgh, Pa 15260 USA, Department Of Computer Science, University Of Pittsburgh, April 2003.

[4] S. Kumar Bandyopadhyay, Debnath Bhattacharyya, Poulami Das, Debashis Ganguly and Swarnendu Mukherjee, **"***A tutorial review on Steganography***"**, International Conference on Contemporary Computing(IC3-2008), Noida, India, August 7-9, pp. 105-114, 2008.

[5] Rade Petrovi, Kanaan Jemili, Joseph M. Winograd, Ilija Stojanovi, Eric Metois, "*Data Hiding Within Audio Signals*" June 15, MIT Media Lab, Series: Electronics and Energetics vol. 12, No.2, pp103-122, 2002.