International Journal of Computer and Electrical Engineering, Vol. 1, No. 5 December, 2009
1793-8163

New Technique of Hidden Data in PE-File with in Unused Area One

A.A.Zaidan, B.B.Zaidan, Fazidah Othman

Abstract---**The strength of the combination between hiding and encryption science is due to the non-existence of standard algorithms to be used in hiding and encrypting secret messages. Also there are many ways in hiding methods such as combining several media (covers) with different methods to pass a secret message. Furthermore, there is no formal method to be followed to discover a hidden data.For this reason, the task of this paper becomes difficult. In this paper proposed a new system of information hiding is presented. The proposed system aim to hide information (data file) in unused area 1 of any execution file (exe.file), to make sure changes made to the exe.file will not be detected by anti-virus and the functionality of the exe.file is still functioning. The system includes two main functions; first is the hiding of the information in the unused area 1 of PE-file (exe.file), through the execution of four process (specify the cover file, specify the information file, encryption of the information, and hiding the information) and the second function is the extraction of the hiding information through three process (specify the steno file, extract the information, and decryption of the information). The testing result shows; the result file does not make any conflict with anti-virus software and the exe.file still function as usual after the hiding process. The proposed system is implemented by using Java.**

***Keyword*-Cryptography, Steganography, portable Executable File, Advance Encryption Standard, Statistical Technique**

I. STEGANOGRAPHY

Steganography is the art of concealing the presence of information within an innocuous container. Steganography has been used throughout History to protect important information from being discovered by

Enemies. A very early example of Steganography comes from the story of Demartus of Greece. He wished to inform Sparta that Xerces the King of Persia was planning to invade. In ancient Greece wax covered wooden tablets were used to record written text [1].In order to avoid detection by the Persians, Demartus scraped the wax from a tablet, etched the message into the underlying wood, then recovered the tabled with wax. This concealed the underlying message from the sentries who inspected the tablets as they left Persia by courier for Greece [1].Other historical examples of Steganography are the use of invisible inks. A common experiment conducted by young kids everywhere is to use a toothpick dipped in vinegar to write a message on a piece of paper. Once the vinegar dries, the presence of the message is not obvious to a casual inspector (aside from the smell). Upon slight heating of the paper, a chemical reaction occurs which darkens the vinegar and makes the message readable. Other, less smelly, invisible inks have been used throughout history similarly even up until World War II.A more recently developed Steganography technique was invented by the Germans in World War II, the use of microdots. Microdots were very small photographs, the size of a printed period, which contain very clear text when magnified. These microdots contained important information about German war plans and were placed in completely unrelated letters as periods. Although Steganography is related to Cryptography, the two are fundamentally different [1], [2].The quick development of multimedia and internet allows for wide distribution of digital media data. It becomes much easier to edit, modify and duplicate digital information. In additional, digital document is also easy to copy and distribute, therefore it may face many threats. It becomes necessary to find an appropriate protection due to the significance, accuracy and sensitivity of the information. Nowadays, protection system can be classified into more specific as hiding information and encrypting information or a combination between them. Cryptography is the practice of 'scrambling' messages so that even if detected, they are very difficult to decipher. The purpose of Steganography is to conceal the message such that the very existence of the hidden is 'camouflaged'. However, the two techniques are not mutually exclusive [2]. Steganography and Cryptography are in fact complementary techniques. No matter how strong algorithm, if an encrypted message is discovered, it will be subject to cryptanalysis. Likewise, no matter how well concealed a message is, it is always possible that it will be discovered [2]. By combining Steganography with Cryptography we can conceal the existence of an encrypted message. In doing this, we make it far less likely that an encrypted message will be found. Also, if a message concealed through Steganography is discovered, the discoverer is still faced with the formidable task of deciphering it [2].

II. PORTABLE EXECUTABLE FILE (PE-FILE)

The proposed system uses a portable executable file as a cover to embed an executable program as an example for the proposed system. This section is divided into five parts [3], [4],[5]:

Manuscript received June 20, 2009.

A. A. Zaidan – PhD candidate, Department of Computer Science & Information Technology, University Malaya, Kuala Lumpur, Malaysia, phone: +60172452457, Postcode: 50603 and Email: awsalaa@perdana.um.edu.my or aws.alaa@yahoo.com.

B. B. Zaidan – PhD candidate, Department of Computer Science & Information Technology, University Malaya, Kuala Lumpur, Malaysia, bilal@perdana.um.edu.my

F. Othman- Mrs. Fazidah Othman - Lecturer, Department of Computer Science & Information Technology, University Malaya, Kuala Lumpur, Malaysia, fazidah@um.edu.my

A. Executable File Types

The number of different executable file types is as many and varied as the number of different image and sound file formats. Every operating system seems to have several executable file types unique to it. These types are [5]:

- EXE (DOS"MZ")
- EXE (win 3.xx "NE")
- EXE (OS/2 "LE")
- EXE (win 9x/NT "PE")
- ELF

B. Concepts Related With PE

The addition of the Microsoft® windows NT™ operating system to the family of windows™ operating systems brought many changes to the development environment and more than a few changes to applications themselves. One of the more significant changes is the introduction of the Portable Executable (PE) file format. The name "Portable Executable" refers to the fact that the format is not architecture specific [6].In other words, the term "Portable Executable" was chosen because the intent was to have a common file format for all versions of Windows, on all supported CPUs [5].The PE files formats drawn primarily from the Common Object File Format (COFF) specification that is common to UNIX® operating systems. Yet, to remain compatible with previous versions of the MS-DOS® and windows operating systems, the PE file format also retains the old familiar MZ header from MS-DOS [6].The PE file format for Windows NT introduced a completely new structure to developers familiar with the windows and MS-DOS environments. Yet developers familiar with the UNIX environment will find that the PE file format is similar to, if not based on, the COFF specification [6].The entire format consists of an MS-DOS MZ header, followed by a real-mode stub program, the PE file signature, the PE file header, the PE optional header, all of the section headers, and finally, all of the section bodies [4].

C. Techniques Related with PE

Before looking inside the PE file, we should know special techniques some of which are [6]:

(i) General view of PE files sections

A PE file section represents code or data of some sort. While code is just code, there are multiple types of data. Besides read/write program data (such as global variables), other types of data in sections include application program interface (API) import and export tables, resources, and relocations. Each section has its own set of in-memory attributes, including whether the section contains code, whether it's read-only or read/write, and whether the data in the section is shared between all processes using the executable file. Sections have two alignment values, one within the desk file and the other in memory. The PE file header specifies both of these values, which can differ. Each section starts at an offset that's some multiple of the alignment value. For instance, in the PE file, a typical alignment would be 0x200. Thus, every section begins at a file offset that's a multiple of 0x200.Once mapped into memory, sections always start on at least a page boundary. That is, when a PE section is mapped into memory, the first byte of each section corresponds to a memory page. On x86 CPUs, pages are 4KB aligned, while on the Intel Architecture IA-64, they're 8KB aligned.

(ii) Relative Virtual Addresses (RVA)

In an executable file, there are many places where an in-memory address needs to be specified. For instance, the address of a global variable is needed when referencing it. PE files can load just about anywhere in the process address space. While they do have a preferred load address, you can't rely on the executable file actually loading there. For this reason, it's important to have some way of specifying addresses that are independent of where the executable file loads. To avoid having hard coded memory addresses in PE files, RVAs are used. An RVA is simply an offset in memory, relative to where the PE file was loaded. For instance, consider an .EXE file loaded at address 0x400000, with its code section at address 0x401000. The RVA of the code section would be:

(Target address) 0x401000 – (load address) 0x400000 = (RAV) (1)

To convert an RVA to an actual address, simply reverse the process: add the RVA to the actual load address to find the actual memory address. Incidentally, the actual memory address is called a Virtual Address (VA) in PE parlance. Another way to think of a VA is that it's an RVA with the preferred load address added in.

(iii) Importing Functions

When we use code or data from another DLL, we're importing it. When any PE files loads, one of the jobs of the windows loader is to locate all the imported functions and data and make those addressees available to the file being loaded.

D. PE File Layout

The PE file layout is shown in Figure 1. There are two unused spaces in PE file layout [7], and these unused spaces are suggested to hide a watermark. The size of the second unused space is different from one file to another [7].The most important reason behind the idea of this system is that the programmers always need to create a back door for all of their developed applications, as a solution to many problems such that forgetting the password[7]. This idea leads the customers to feel that all programmers have the ability to hack their system any time. At the end of this discussion all customers always are used to employ trusted programmers to build their own application. Programmers want their application to be safe anywhere without the need to build ethic relations with their customers. In this system a solution is suggested for this problem [6],[8].The solution is to hide the password in the executable file of the same system and then other application to be retracted by the customer himself. Steganography needs to know all files format to find a way for hiding information in those files. This technique is difficult because there are always large numbers of the file format and some of them have no way to hide information in them [8].

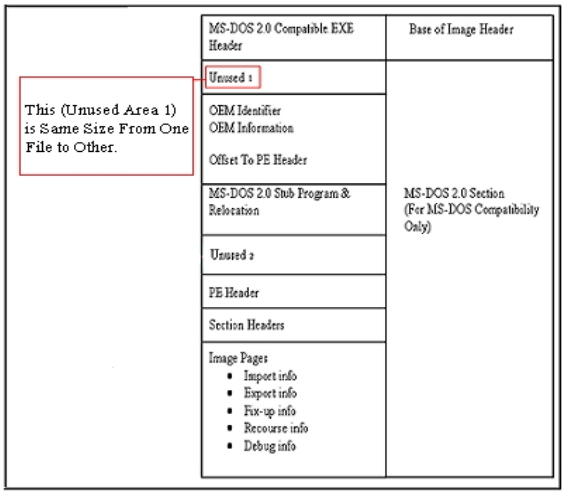

Figure 1.Typical 32-bit Portable .EXE File Layout.

E. Characteristics of the Executable

The characteristics of the Executable file does not have a standard size, like other files, for example the image file (BMP) the size of this file is between (2-10 MB), Other example is the text file (TEXT) the size often is less than 2 MB.Through our study the characteristics of files have been used as a cover, it found that lacks sufficient size to serve as a cover for information to be hidden. For these features of the Executable file, it has unspecified size; it can be 650 MB like window setup File or 12 MB such as installation file of multi-media players. Taking advantage of this feature (disparity size) make it a suitable environment for concealing information without detect the file from attacker and discover hidden information in this file[4],[6].

III. Statistical Technique

Statistical Steganography techniques utilize the existence of "1-bits" Steganography schemes, which embed one bit of information in a digital carrier. This is done by modifying the cover in such a way that some statistical characteristics change significantly if a "1" is transmitted. Otherwise, the cover is left UN changed. So the receiver must be able to distinguish unmodified covers from modified ones. A cover is divided into l (m) disjoint blocks B1...B l (m). A secret bit, mi is inserted into the ith block by placing "1" in to Bi if mi=1.Otherwise, the block is not changed in the embedding process [5].

Table 1: Weakness of Steganography Techniques

Steganography Techniques	Weakness
Substitution Systems	Low robustness: filtering, lossy compression attacks, format file dependand.
Transform Domain Techniques	An attacker can simply apply signal processing techniques in order to destroy the secret information. In many cases even the small changes resulting out of loose compression systems yield total information loss.
Spread Spectrum (SS) Techniques	There are increases in the complexity, higher costs and more stringent timing requirements. a) Direct-Sequence Scheme:The circuitry required to produce the spectrum is complex, it requires a large bandwidth channel with relatively small phase distortions and requires a long acquisition time since the PN codes are long. b) Frequency–Hopping Scheme: Weakness with both slow and fast hopping. With slow hopping, coherent data detection is possible, but data can be lost if a single frequency hop channel is jammed. To overcome this, it is necessary to use error correcting codes. Fast hopping disposes of the need for error codes since one bit of data is spread over a number of hops. However, fast hopping has the disadvantage that due to phase discontinuities, coherent data detection is not possible.
Distortion Techniques	In many applications, such systems are not useful, since the receiver must have access to the original cover.It is weakness point.So if the attacker also has access to them, he/she can easily detect the cover modification and has evidence for a secret communication.If the embedding and extraction functions are public and do not depend on a stego-key, it is also possible for the attacker to reconstruct secret message entirely.
Cover Generation Techniques	They have heavy and complexity process for algorithems compersion with other techniques. This point due to dealy time for finished (hiding or extract) process operation .Example: Automated Generation of English Text. Use a large dictionary of words categorised by different types, and a style source which describes how words of different types can be used to form a meaningful sentence. Transform message bits into sentences by selecting words out of the dictionary which conforms to a sentence structure given in the style source.

From the above table, most of the techniques are very complex and not suitable to be used with exe.file. In order to use exe.file. Thus, we choose to apply statistical technique because it is not complex and suitable to be implemented with the structure and characteristic of the exe.file.

IV. Advance Encryption Standard

In 1997, the NIST called for submissions for a new standard to replace the aging DES. The contest terminated in November 2001with the selection of the Rijndael cryptosystem as the Advanced Encryption Standard (AES) [1],[2]. The Rijndael cryptosystem operates on 128-bit blocks, arranged as 4 × 4 matrices with 8-bit entries. The algorithm can use a variable block length and key length. The latest specification allowed any combination of keys lengths of 128, 192, or 256 bits and blocks of length 128, 192, or 256 bits.AES may, as all algorithms, be used in different ways to perform encryption. Different methods are suitable for different situations. It is vital that the correct method is applied in the correct manner to each and every situation, or the result may well be insecure even if AES as such is secure. It is very easy to implement a system using AES as its encryption algorithm, but much more skill and experience are required to do it in the right way for a given situation. To describe exactly how to apply AES for varying purposes is very much out of scope for this short introduction.

A. Strong keys

Encryption with AES is based on a secret key with 128, 192 or 256 bits. But if the key is easy to guess it doesn't matter if AES is secure, so it is as critically vital to use good and strong keys as it is to apply AES properly. Creating good and strong keys is a surprisingly difficult problem and requires careful design when done with a computer. The challenge is that computers are notoriously deterministic, but what is required of a good and strong key is the opposite unpredictability and randomness. Keys derived into a fixed length suitable for the encryption algorithm from passwords or pass phrases typed by a human will seldom correspond to 128 bits much less 256. To even approach 128- bit equivalence in a pass phrase, at least 10 typical passwords of the kind frequently used in day-to-day work are needed. Weak keys can be somewhat strengthened

by special techniques by adding computationally intensive steps which increase the amount of computation necessary to break it. The risks of incorrect usage, implementation and weak keys are in no way unique for AES; these are shared by all encryption algorithms. Provided that the implementation is correct, the security provided reduces to a relatively simple question about how many bits the chosen key, password or pass phrase really corresponds to. Unfortunately this estimate is somewhat difficult to calculate, when the key is not generated by a true random generator [3].

B. The Round Transformations

There are four transformations:

(i). Add Round Key

Add Round Key is an XOR between the state and the round key. This transformation is its own inverse [4].

(ii). Sub Bytes

Sub Bytes is a substitution of each byte in the block independent of the position in the state. This is an S-box. It is bisection on all possible byte values and therefore invertible (the inverse S-box can easily be constructed from the S-box). This is the non-linear transformation. The S-box used is proved to be optimal with regards to non-linearity. The S-box is based on arithmetic in GF (2^8).

(iii). Shift Rows

Shift Rows is a cyclic shift of the bytes in the rows in the state and is clearly invertible (by a shift in the opposite direction by the same amount) [5].

(iv). Mix Columns

Each column in the state is considered a polynomial with the byte values as coefficients. The columns are transformed independently by multiplication with a special polynomial c(x). c(x) has an inverse d(x) that is used to reverse the multiplication by c (x) [1].

C. The Rounds

A round transformation is composed of four different transformations as shown in fig.2. The Round keys are made by expanding the encryption key into an array holding the Round Keys one after another. The expansion works on words of four bytes. Nk is a constant defined as the number of four bytes words in the key. The encryption key is filled into the first Nk words and the rest of the key material is defined recursively from preceding words. The word in position i, W[i], except the first word of a Round Key, is defined as the XOR between the preceding word, W[i-1], and W[i-Nk]. The first word of each Round Key, W[i] (where i mod Nk == 0), is defined as the XOR of a transformation on the preceding word, T (W [i - 1]) and W [i - Nk]. The transformation T on a word, w, is w rotated to the left by one byte, XOR'ed by a round constant and with each byte substituted by the S-box [5].

```
Round (State, RoundKey) {
SubBytes(State);
ShiftRows(State);
MixColumns(State);
AddRoundKey(State, RoundKey);
}
```

Figure 2. Four Different Transformations.

The final round is like a regular round, but without the mix columns transformation as shown in fig. 3:

```
FinalRound(State, RoundKey) {
SubBytes(State);
ShiftRows(State);
AddRoundKey(State, RoundKey);
}
```

Figure 3. Final Round.

V. METHODOLOGY

A. System Concept

Concept of this system can be summarized as hiding the password or any information beyond the end of an executable file so there is no function or routine (open-file, read, write, and close-file) in the operating system to extract it. This operation can be performed in two alternative methods: Building the file handling procedure independently of the operating system file handling routines. In this case we need canceling the existing file handling routines and developing a new function which can perform our need, with the same names. The advantage of these methods is it doesn't need any additional functions, which can be identified by the analysts. And it can be executed remotely and suitable for networks and the internet applications .The disadvantage of these methods is it needs to be installed (can not be operated remotely). So we choose this concept to implementation in this paper.

B. System Features

This system has the following features*:*

- The cover file can be executed normally after hiding operation. Because the hidden information already hide in the unused area 1 within exe.file and thus cannot be manipulated as the exe.file, therefore, the cover file still natural, working normally and not effected, such as if the cover is exe.file (WINDOWES XP SETUP) after hiding operation it'll continued working, In other words, the exe.file can be installed of windows.
- It's very difficult to extract the hidden information it's difficult to find out the information hiding , that is because of three reasons:
  - The information hiding was encrypted before hiding of the information by AES method; this method very strong, 128-bit key would be in theory being in range of a military budget within 30-40 years. An illustration of the current status for AES is given by the following example, where we assume an attacker with the capability to build or purchase a system that tries keys at the rate of one billion keys per second. This is at least 1 000 times faster than the fastest personal computer in 2004. Under this

assumption, the attacker will need about 10 000 000 000 000 000 000 000 000 years to try all possible keys for the weakest version.

- The attacker impossible guessing the information hiding inside the exe.file because of couldn't guessing the real size of (exe.file and information hiding).
- The information hiding should be decrypted after retract of the information.

- Virus detection programmers' can't detect such as files, the principle of antivirus check are checking from beginning to end. When checking the exe.files by antivirus, will checked it from beginning to end of it, since the principle of information hiding for that system within unused area 1 of exe.file .The information hiding will be encrypt after that it will be hidden, the antivirus discontinue checking in the unused area 1of exe.file after hiding process because the unused area 1 still empty so didn't mention to anything inside the exe.file while doing scanning.

C. The Proposed System Structure

To protect the hidden information from retraction the system encrypts the information by the built-in encryption algorithm provided by the Java. The following algorithm for hiding operation procedure as shown in Figure 4. The following algorithm for Retract operation procedure as shown in Figure 5

The following algorithm the hiding operation procedure:

Procedure: Hide operation.
Input: Hidden file name, cover file name.
Output: Stego-File.

- Begin.
- Opens the cover file (EXE file).
- Assign a pointer to the end of (MS-DOS 2.0 Compatible EXE), which before the unused space 1 of the cover file
- Write the hidden file name in the unused space 1 to the cover file
- Assign a pointer to (EXE file) after hidden file name.
- Encrypt the hidden file.
- Write the encrypt contact to the file cover (EXE file).
- End.

Figure 4. Shows Algorithm for Hiding Operation.

The following algorithm retraction operation procedure:

Procedure: Retract operation.
Input: Stego-File.
Output: hidden information.

- Begin(1)
- Select the cover file.
- Get the end of (MS-DOS 2.0 Compatible EXE) of EXE File.
- If end of (MS-DOS 2.0 Compatible EXE) Pointer exists.
- Begin (2) read the name of hidden file.
- Read the hiding data.
- Decrypt the data using the file name as a key.
- Create a file using hiding file name.
- Write in to the create file the decrypt data.
- End (2).

Else

- Display a message (no hiding file).
- End (1).

Figure 5. Shows Algorithm for Retract Operation.

VI. Testing Of The System

A. Testing Approach

There are two fundamental approaches to identifying test cases, these are known as functional and structure testing, each of these approaches has several distinct test case identification methods, more commonly called testing methods, functional testing is based on the view that any program can be considered to be a function that maps values from its input domain to values in its output range. (Function, domain and range) this notion is commonly used in engineering. There are two distinct advantages to functional test cases: they are independent of how the software is implemented, so if the implementation changes, the test cases are still useful and test case development can occur in parallel with the implementation, thereby reducing overall research development interval, on other side, functional test cases frequently suffer from two problems: there can be significant redundancies among test cases, and this is compounded by the possibility of gaps of untested software. As shown in figure 6.

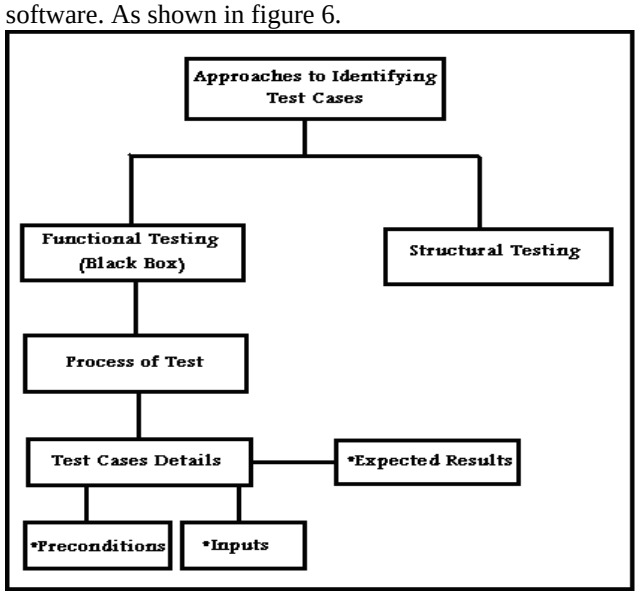

Figure 6: Approaches to Identifying Test Cases

When systems are considered to be "black boxes" test cases are generated and executed from the specification of the required functionality at defined interfaces, this leads to

the function of the black box is understood completely in terms of its inputs and outputs, as shown in figure 7. Black-box testing has some important advantages:

- It does not require that the code is seen, it is testing. Sometimes code will not be available in source code form, yet it can still construct useful test cases without it. The person writing the test cases does not need to understand the implementation.
- The test cases do not depend on the implementation. They can be written in parallel with or before the implementation. Further, good black-box test cases do not need to be changed. Even if the implementation is completely rewritten.
- Constructing black-box test cases causes the programmer to think carefully about the specification and its implications. Many specification errors are caught this way.

The disadvantage of black box testing is that its coverage may not be as high as like, because it has to work without the implementation. But it is a good place to start when writing test cases, with the functional approach to test case identification; the only information that is used is the specification of the software.

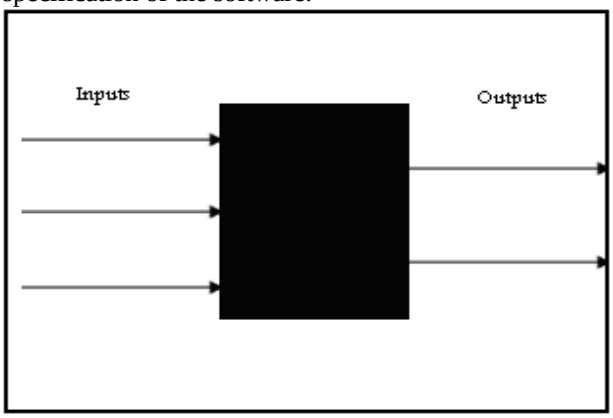

Figure 7: Black Box.

B. Process of the Test

(i). Test Case One:

In this case making test for the usage of exe.files after the hiding operation done:

- Four pictures approve the cover (exe.files) usage after the hiding operation and these pictures divide to: (First picture of text, Second picture of image, Third picture of video. and fourth picture of audio).

(ii). Test Case Two:

Testing for Scanning Result (undetectable by antivirus software):

- Four pictures approve the cover (exe files) undetectable from antivirus software after the hiding operation and these pictures divide to: (First picture of text, Second picture of image, Third picture of video, and fourth picture of audio.

C. Test Cases Details

Test cases are known preconditions, inputs and expected results, which is worked out before the test is executed. The definition of software installation needed for test an (Preconditions) and the definition inputs should needed for test an (Inputs) and the definition predictable results for outputs an (Except Results).

(i). Preconditions:

1. Installation (Microsoft Windows XP for Any Version or Vista).
2. Installation (Jcreators and JDK or Net beans Editor).
3. Installation (Microsoft Office Word Document 2003 or 2007).
4. Installation (Software Antivirus).
5. Installation (Real Player Programme).
6. Installation (Jet Audio Programme).
7. Installation (ACDSEE Programme).
8. System application for this research.

(ii). Inputs:

The system has two types of inputs:

- Inputs for cover (exe.files):
  - Four types of the cover (exe.files)
- Inputs for information hidden:
  - One text file.
  - One image file
  - One video file
  - One audio file

Table 2: Inputs for Test Cases.

Name of Inputs	Type of Inputs
Cover 1	VMware Player Setup
Cover 2	SSH
Cover 3	JCreator Editor Setup
Cover 4	JDK Setup.
Text 1	Word Document
Video 1	Real Player
Audio 1	Jet Audio
Image 1	JPEG

(iii). Expected Results:

1. These covers (exe.files) usage after the hiding operation.
2. These covers (exe.files) undetectable from antivirus software after the hiding operation.

D. Subscript the Test Cases

(i). Test Case One

In this test case shows picture of the cover files after hiding operation of all types of multimedia files (text, image, audio and video), which related with this system, approve these cover (exe.files) usage after the hiding operation.

Table 3: Inputs and Outputs for Test Case One.

Before Hiding Operation		After Hiding Operation
No of Cover	Information Hidden	Usage the EXE Covers
1	Text1	Figure 8
2	Image1	Figure 9
3	Video 1	Figure 10
4	Audio 1	Figure 11

1. Text

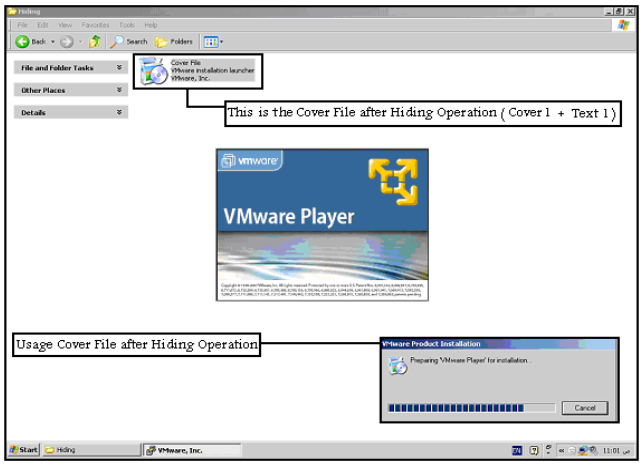

Figure 8: After Hiding Operation Inside the (Hiding Folder), Executable File (Cover 1) Still Working.

2. Image

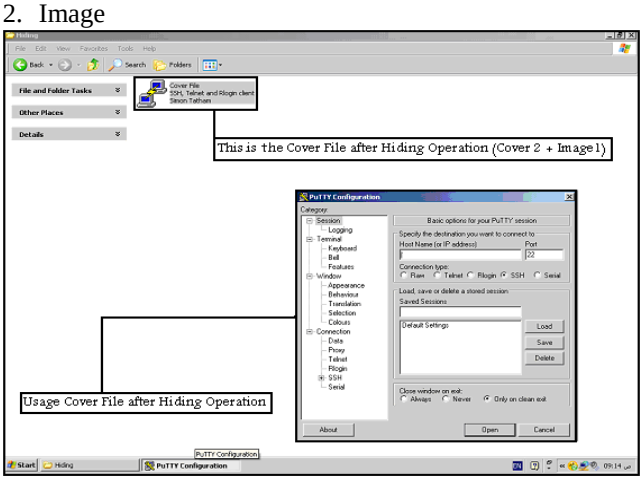

Figure 9: After Hiding Operation Inside the (Hiding Folder), Executable File (Cover 2) Still Working.

3. Video

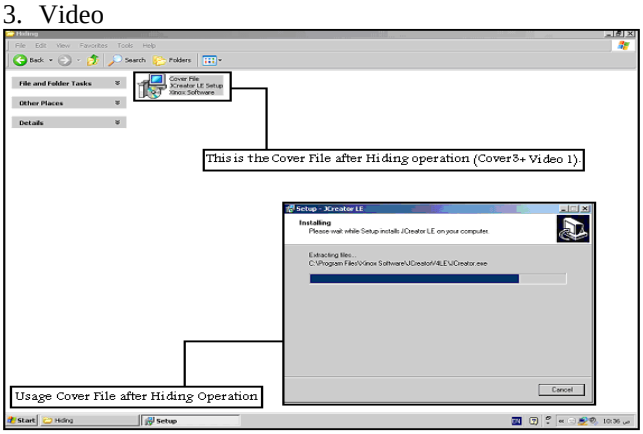

Figure 10: After Hiding Operation Inside the (Hiding Folder), Executable File (Cover 3) Still Working.

4. Audio

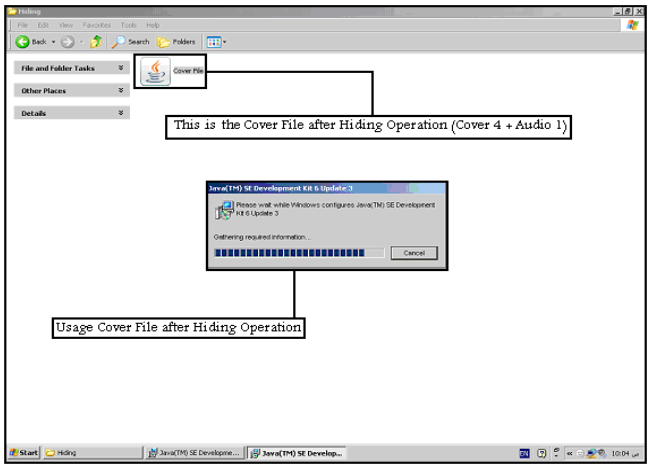

Figure 11: After Hiding Operation Inside the (Hiding Folder), Executable File (Cover 4) Still Working.

(ii). Test Case Two

In this test case shows picture of cover files after hiding operation of all types of multimedia files (text, image, audio and video), which related with this system, approve these covers (exe.files) undetectable from antivirus software after the hiding operation.

Table 4: Inputs and Outputs for Test Case Two.

Before Hiding Operation		After Hiding Operation
No of Cover	Information Hidden	EXE Cover Undetectable for Antivirus
1	Text1	Figure 12
2	Image1	Figure 13
3	Video 1	Figure 14
4	Audio 1	Figure 15

1. Text:

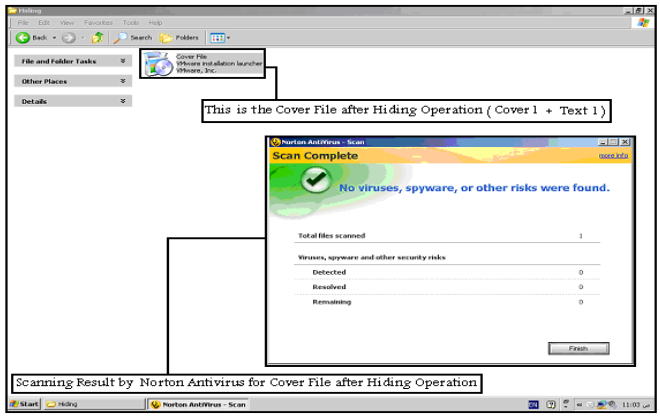

Figure 12: Shows that the Executable File (Cover 1) Inside (Hiding Folder) Immune to Anti-virus Program.

2. Image

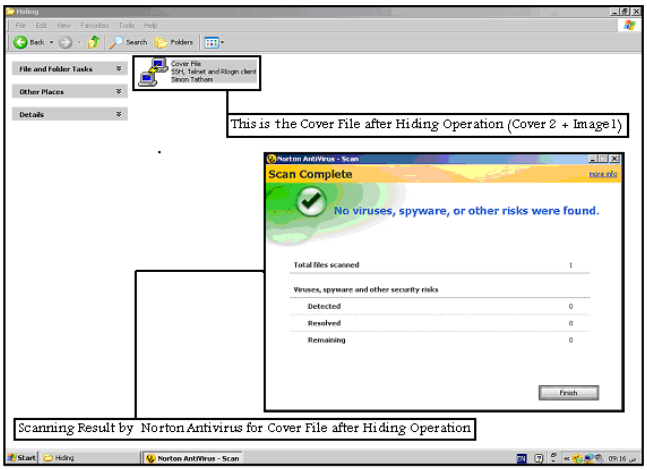

Figure 13: Shows that the Executable File (Cover 2) File Inside (Hiding Folder) Undetectable by Anti-virus program.

3. Video

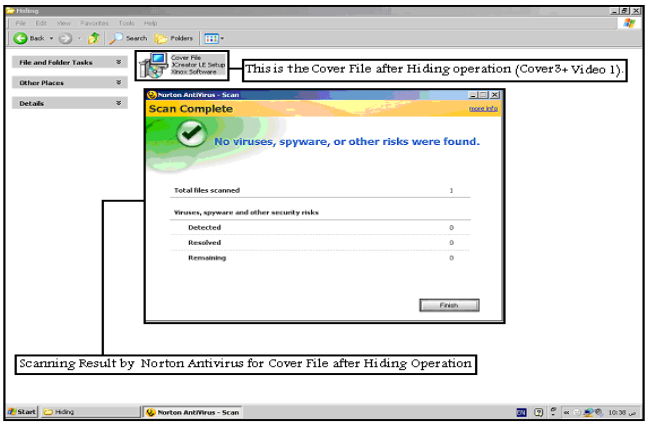

Figure 14: Shows that the Executable File (Cover 3) Inside (Hiding Folder) Immune to Anti-virus Program.

4. Audio

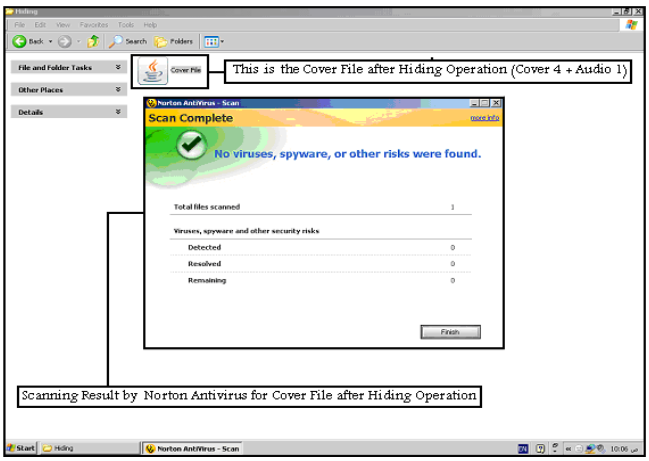

Figure 15: Shows that the Executable File (Cover 4) File Inside (Hiding Folder) Immune to Anti-virus Program.

VII. Conclusion

The exe.files are one of the most important files in operating systems and in most systems designed by developers (programmers/software engineers), and then hiding information in these file is the basic goal for this paper, because most users of any system cannot alter or modify the content of these files. PE files structure is very complex because they depend on multi headers and addressing, and then insertion of data to PE files without full understanding of their structure may damage them, so the choice is to hide the information beyond the structure of these files. Most antivirus systems do not allow direct write in executable file, so the approach of the proposed system is to prevent the hidden information to observation of these systems. One of the important conclusions in implementation of the proposed system is the solving of the problems that are related to the functionality of the exe.file, so the executable files still working after its use as cover for embedding data. The encryption of the message increases the degree of security of hiding technique which is used in the proposed system. The proposed hiding technique is flexible and very useful in hiding any type of data for files message (text, image, sound or video).

ACKNOWLEDGEMENT

Our sincere thanks to all researchers who have contribute to this project. Also we would like to acknowledge and thanks the researchers in UM for their support.

References

[1] A.A.Zaidan, B.B.Zaidan, M.M.Abdulrazzaq, R.Z.Raji and S.M.Mohammed, "Implementation Stage for High Securing Cover-File of Hidden Data Using Computation between Cryptography and Steganography". International Association of Computer Science and Information Technology (IACSIT), indexing by Nielsen, Thomson ISI (ISTP), IACSIT Database, British Library and EI Compendex, Volume 20, 2009, Manila, Philippines.

[2] A.W.Naji, A.A.Zaidan, B.B.Zaidan, Shihab A, Othman O. Khalifa, " Novel Approach of Hidden Data in the (Unused Area 2 within EXE File) Using Computation Between Cryptography and Steganography ", International Journal of Computer Science and Network Security (IJCSNS) , Vol.9, No.5 , ISSN : 1738-7906, pp. 294-300, May 30 (2009), Seoul, Korea.

[3] B.B.Zaidan, A.A.Zaidan, Fazidah Othman "Enhancement of the Amount of Hidden Data and the Quality of Image", Malaysia Education Security (MyEduSec08), Grand Continental Hotel, 2008, Kuala Trengano, Malaysia

[4] Avedissian, L.Z," Image in Image Steganography System", Ph.D.Thesis, Informatics Institute for Postgraduate Studies (IIPS), University of Technology, Baghdad, Iraq, 2008.

[5] C. J. S. B," Modulation and Information Hiding in Images", of Lecture Notes in Computer Science, University of Technology, Malaya, Vol. 1174, pp.207-226, 2007.

[6] Clelland, C.T.R, V.P & Bancroft, " Hiding Messages in DNAMicroDots " , International Symposium on Industrial Electronics (ISIE) , University of Indonesia , Indonesia, Vol. 1, pp.315-327, 2007.

[7] Davern, P.S, M.G, "Steganography It History and Its Application to Computer Based Data Files", School of Computer Application (SCA), Dublin City University. Working Paper. Studies (WPS), Baghdad, Iraq, 2007.

[8] Dorothy, E.R, D.K, "Cryptography and Data Security", IEEE International Symposium on Canada Electronics (ISKE), University of Canada, Canada, Vol.6, pp.119-122, 2006,

Aos Alaa Zaidan - He obtained his 1st Class Bachelor degree in Computer Engineering from university of Technology / Baghdad followed by master in data communication and computer network from University of Malaya. He led or member for many funded research projects and He has published more than 40 papers at various international and national conferences and journals, he has done many projects on Steganography for data hidden through different multimedia carriers image, video, audio, text, and non multimedia carrier unused area within exe.file, Quantum Cryptography and Stego-Analysis systems, currently he is working on the

multi module for Steganography. He is PhD candidate on the Department of Computer System & Technology / Faculty of Computer Science and Information Technology/University of Malaya /Kuala Lumpur/Malaysia.

Bilal Bahaa – he obtained his bachelor degree in Mathematics and Computer Application from Saddam University/Baghdad followed by master from Department of Computer System & Technology Department Faculty of Computer Science and Information Technology/University of Malaya /Kuala Lumpur/Malaysia, He led or member for many funded research projects and He has published more than 40 papers at various international and national conferences and journals. His research interest on Steganography & Cryptography with his group he has published many papers on data hidden through different multimedia carriers such as image, video, audio, text, and non multimedia careers such as unused area within exe.file, he has done projects on Stego-Analysis systems, currently he is working on Quantum Key Distribution QKD and multi module for Steganography, he is PhD candidate on the Department of Computer System & Technology / Faculty of Computer Science and Information Technology/University of Malaya /Kuala Lumpur/Malaysia.

Fazidah Othman- has received her master from University Technology Malaysia, Faculty of Computer Science currently she is a lecturer, Department of Computer System & Technology/University of Malaya /Kuala Lumpur/Malaysia, her research interest on the network security, intelligent agent for Machine translation tools, she has many publication on Steganography, Agent Technology for Proxy Server, her PhD work on the multicast group security management.