World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:3, No:6, 2009

# High Securing Cover-File of Hidden Data Using Statistical Technique and AES Encryption Algorithm

A. A. Zaidan, Anas Majeed, and B. B. Zaidan

Open Science Index, Computer and Information Engineering Vol:3, No:6, 2009 publications.waset.org/1349/pdf

***Abstract*—**Nowadays, the rapid development of multimedia and internet allows for wide distribution of digital media data. It becomes much easier to edit, modify and duplicate digital information Besides that, digital documents are also easy to copy and distribute, therefore it will be faced by many threatens. It's a big security and privacy issue with the large flood of information and the development of the digital format, it become necessary to find appropriate protection because of the significance, accuracy and sensitivity of the information. Nowadays protection system classified with more specific as hiding information, encryption information, and combination between hiding and encryption to increase information security, the strength of the information hiding science is due to the non-existence of standard algorithms to be used in hiding secret messages. Also there is randomness in hiding methods such as combining several media (covers) with different methods to pass a secret message. In addition, there are no formal methods to be followed to discover the hidden data. For this reason, the task of this research becomes difficult. In this paper, a new system of information hiding is presented. The proposed system aim to hidden information (data file) in any execution file (EXE) and to detect the hidden file and we will see implementation of steganography system which embeds information in an execution file. (EXE) files have been investigated. The system tries to find a solution to the size of the cover file and making it undetectable by anti-virus software. The system includes two main functions; first is the hiding of the information in a Portable Executable File (EXE), 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 system has achieved the main goals, such as make the relation of the size of the cover file and the size of information independent and the result file does not make any conflict with anti-virus software.

***Keywords*—**Cryptography, Steganography, Portable Executable File.

## I. Introduction

WITH the emergence and development of computer science and informatics emerged the urgent need to find ways to avoid Muggers and computer hackers from stealing or disclosure of data and sensitive task information. It was learned that arose organization (Cryptography) optimal way to achieve this end, this science has evolved steadily emerged the systems and very efficient techniques. But with the advent of information and communications networks global information network (Internet) has become the issue of complexity of the privacy and unattainable due to achieve this process and to ensure access to the data required was necessary to be accessible to everyone online common, and here is the problem of inefficient organization, vision statements as loose enough to push the spam or the attacker to believe that important or sensitive data lies in these random or encrypt text, some techniques comes using anti-encrypted to attempt to dismember the symbols and creating content, even if unable to do so, it might tampering or distorts or used some means available to prevent access to its goal.Elsewhere Governments began losing control of the encrypted messages exchanged between the institutions, companies and the possibility of these texts contain encrypted information may be against the security and the public interest, and therefore resorted to some governments to prevent the use of the organization for users of communications networks for personal purposes [1],[2].From here emerged the urgent need to find new techniques alternative organization to overcome these weaknesses, giving rise to conceal information technology (Information Hiding), which are based on a different principle to the idea of organization, where they are buried information (Information Embedding) within other media carrier, and making them aware (Imperceptible) by hackers and attackers, and so are the public domain of information to users of the network, while the content monopoly "on the relevant agencies, which alone knows how to extract content[3],[4]. One of the latest techniques that have been used in this area by researchers at the Mount Sinai School MOUNT SINAI Medical in New York New York in 2007, as they managed to hide the secret texts in sentences Strand human DNA (Human DNA) by using a technique called genetic system coverage (Genomic Steganography), and by placing signs resolution to be agreed upon in the nuclei chromosomes and then integrate these with millions sentences and sent to the other end. To extract the secret message is soaking get special distinction sentences used on the other and then placed under the microscope to extract the required text [5]. The oldest Authentications on Steganography taken from the legendary stories Greeks Herodotus and then back to the fifth century BC, these sources indicate that they felt they fly head of the Messenger and then write the letter secret in the head, leaving hair to grow then be sent to the required which is a re-extraction letter [5],[6].Authentications and other writing secret messages on the wood panels and then covered wax and will be hid those

Aos Alaa Zaidan is with Department of Computer Science & Information Technology, University Malaya, Kuala Lumpur, 50603 Malaysia (phone: +60172452457; e-mail: awsalaa@perdana.um.edu.my).

Anas Majeed Hamid is with Faculty Computer System & Information Technology, University of Malaya, Kuala Lumpur, Malaysia (e-mail: hart2hartes@perdana.um.edu.my).

Bilal Bahaa Zaidan is with Department of Computer Science & Information Technology, University Malaya, Kuala Lumpur, Malaysia (e-mail: bilal@perdana.um.edu.my).

Open Science Index, Computer and Information Engineering Vol:3, No:6, 2009 publications.waset.org/1349/pdf

writing panels appear free of anything. And they were killing their animals as rabbit example corner confidential letter inside it [7].Other means that the common use since the first century AD, invisible inks Invisible Inks, which was able to write a confidential letter with any other non-value-confidential and usually write between lines, for example those rabbis some fruit juices Fruit Juices, milk, urine, vinegar, and all these species become dark and visible when exposed to heat the written document [7]. Then these kinds of inks evolved with the evolution of science chemical was used vehicles carrying chemical characteristics of the same old species with a more accurate and efficient have been used during the First and Second World Wars in the military secrecy of correspondence. Other technical been used during World War II is sending a message hidden within another message is not relevant, and based on the idea of a nomination letters every word of the letter counterfeit representation of characters from the characters letter requested confidentiality.Moreover, there are numerous ideas for the same method is used to be more than characters, or take certain words or phrases within the text fake and leaving the rest. Finally, it should be noted that the senior researcher in the area of concealment and science-based organization itself is German Johannes Trithemius ((between (1462-1526), and the oldest books in the area of coverage Posted by Gaspari Schotti)) in1665 in the name of (Steganographyica) and (400) contains a page where all the ideas included (Trithemius) [7][8].

## 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 four parts:

- Executable file types.
- Concept related with PE-file.
- Techniques related with PE-file.
- PE-file Layout.

### *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 [3]:

- EXE (DOS"MZ")

DOS-MZ was introduced with MS-DOS (not DOS v1 though) as a companion to the simplified DOS COM file format. DOS-MZ was designed to be run in real mode and having a relocation table of SEGMENT: OFFSET pairing. A very simple format that can be run at any offset, it does not distinguish between TEXT, DATA and BSS.
The maximum file size of (code + data + bss) is one-mega bytes in size.
Operating systems that use are: DOS, Win*, Linux DOS.

- EXE (win 3.xx "NE"):

The WIN-NE executable formatted designed for windows 3.x is the "NE" new-executable. Again, a 16-bit format, it alleviates the maximum size restrictions that the DOZ-MZ has[4].
Operating system that uses it is: windows 3.xx

- EXE (OS/2 "LE"):

The "LE" linear executable format was designed for IBM's OS/2 operating system by Microsoft. Supporting both 16 and 32-bit segments Operating systems that are used in: OS/2, DOS [5].

- EXE (win 9x/NT "PE"):

With windows 95/NT a new executable file type is required, thus was born the "PE" portable executable. Unlike its predecessors, the WIN-PE is a true 32-bit file format, supporting relocatable code. It does distinguish between TEXT, DATA, and BSS. It is in fact, a bastardized version of the common object file format (COFF) format.
Operating systems that use it are: windows 95/98/NT/2000/ME/CE/XP [5].

- ELF:

The ELF, Executable Linkable Format was designed by SUN for use in their UNIX clone. A very versatile file format, it was later picked up by many other operating systems for use as both executable files and as shared library files [7].
It does distinguish between TEXT, DATA and BSS.
TEXT: the actual executable code area.
DATA: "initialized" data, (Global Variables).
BSS : "un- initialized" data, (Local Variables).

### *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 [9]. 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 [11].

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 [11].

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 [10].

The entire format consists of an MS-DOS MZ header, followed by a real-mode stub program, the PE file signature,

Open Science Index, Computer and Information Engineering Vol:3, No:6, 2009 publications.waset.org/1349/pdf

the PE file header, the PE optional header, all of the section headers, and finally, all of the section bodies [12].

### C. Techniques Related with PE

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

- 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[8].

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.

- 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 [12].

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.

- 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 [12], and these unused spaces are suggested to hide a watermark.The size of the second unused space is different from one file to another [12].

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. 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 [12].

Programmers want their application to be safe any where without the need to build ethic relations with their customers. In this system a solution is suggested for this problem. 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.

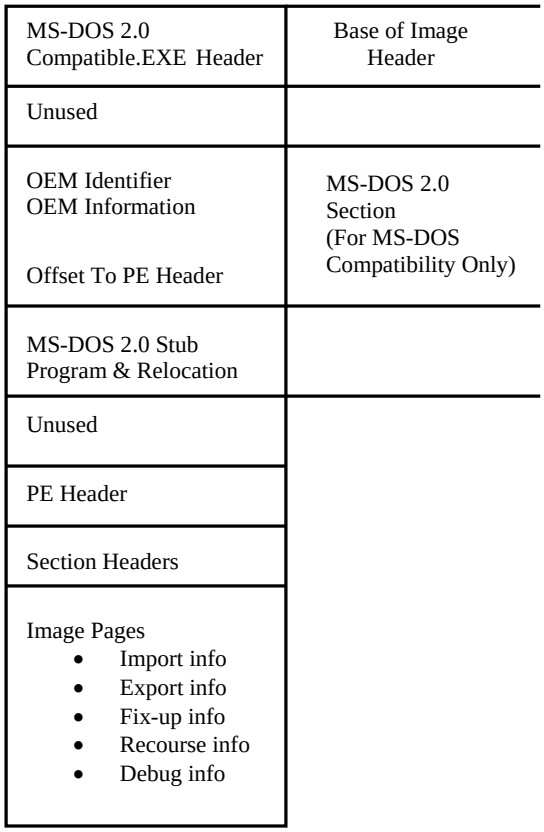



Fig. 1 Typical 32-bit Portable File EXE File Layout

Open Science Index, Computer and Information Engineering Vol:3, No:6, 2009 publications.waset.org/1349/pdf

## III. 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. This way needs the customer to install the system application manually as shown in Fig. 2.
- Developing the file handling functions depending on the existing file handling routines. This way can be performed remotely as shown in Fig. 3. The advantage of the first method is it doesn't need any additional functions, which can be identified by the analysts.
  The disadvantage of this method is it needs to be installed (can not be operated remotely). The advantage of the second method is it can be executed remotely and suitable for networks and the internet applications. So we choose this concept to implementation in this research.

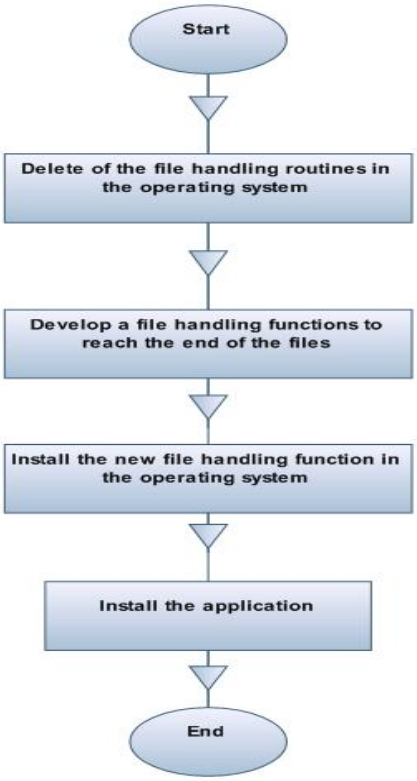



Fig. 2 First Method of the System Concept

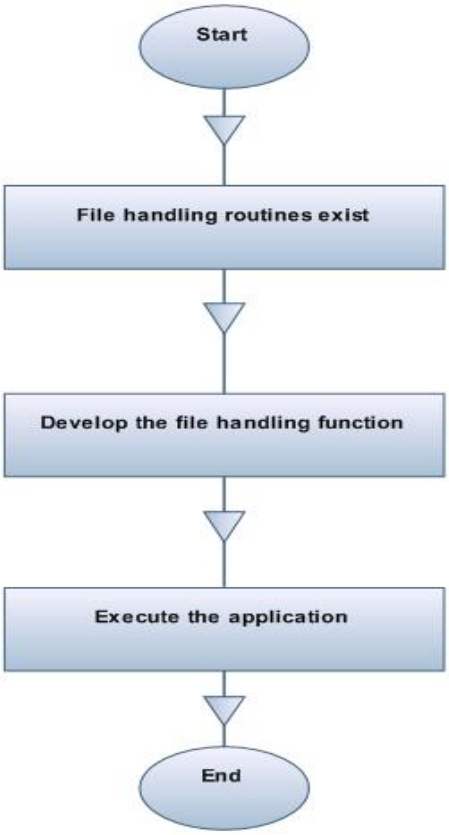



Fig. 3 Second Method of the System Concept

### 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 after the end of 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 files (WINDOWES XP SETUP) after hiding operation it'll continued working, In other words, the EXE file can be installed of windows.
- There is no limitation on the hidden file size where you can hide any file of any size regardless of the size of hidden information by structure on the property of the EXE file, so that the EXE cannot identify the size of the EXE file, so can using type of EXE file such as JDK whose contain number of different size (72MB, 77MB or 65MB), other world disparity in the size of the executable files, so can hide any size inside it without guessing the real size of the information hidden by the attacker. Furthermore, when hide after the end of EXE file, there is no limitation of the size files which must be hiding after the end of EXE file, open space of any size.
- It's very difficult to extract the hidden information it's difficult to find out the information hiding , that is because of three reasons:

Open Science Index, Computer and Information Engineering Vol:3, No:6, 2009 publications.waset.org/1349/pdf

- 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 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.

- The hidden information can be of any type of multimedia files (Text, Audio, Video or Image) of any size without limitation and also can hidden all type of multimedia files in the same time inside the same cover, so can put (Text, Image, Video and Audio) in one folder and compressed them and then choose the compressed folder as a information hiding, in that way can hidden all in the same time.

- 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 is after end of file, the antivirus discontinue checking in the end of file 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 VB.net. The hiding operation can be performed as shown in Fig. 4. The retraction operation can be performed as shown in Fig. 5.

The following algorithm is the hiding operation procedure:

Procedure: Hide Operation.
Input: Hiding file name, Cover file name.
Out put: Stego-File.

Begin:
Step (1): Open the cover file (PE-file).
Step (2): Assign a pointer to the end of file.
Step (3): Write the file name after the end of file (PE-file).
Step (4): Assign a pointer to PE-file after the hidden file name.
Step (5): Encrypt the hidden file.
Step (6): Write the encrypt content to the file cover (PE-file).
End.

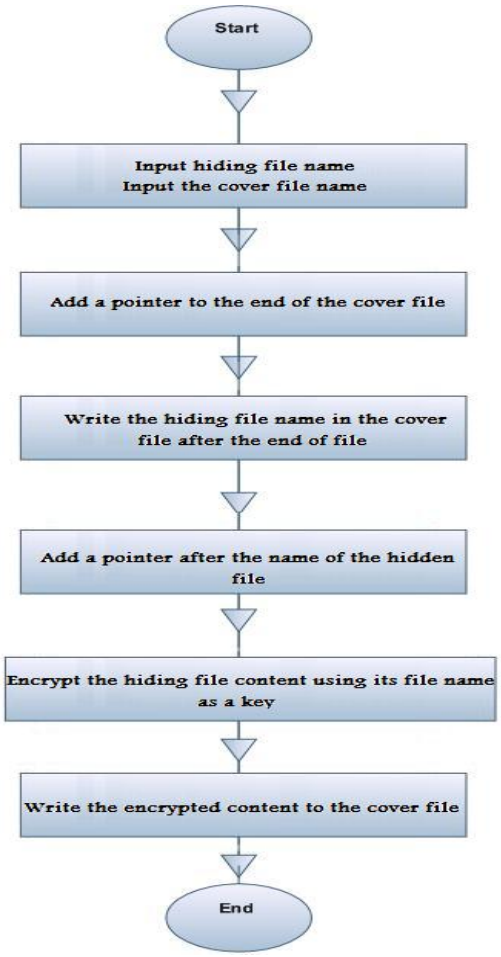



Fig. 4 Hiding Operation

Open Science Index, Computer and Information Engineering Vol:3, No:6, 2009 publications.waset.org/1349/pdf

The following algorithm is retraction operation procedure:

```
Procedure: Retraction Operation.
Input: Stego-file.
Output: Hidden information file.

Begin:
  Step (1): Select the cover file.
  Step (2): Get the end of file.
  Step (3): If end of file pointer exist.
Begin:
  Step (3-1): Read the name of hidden file.
  Step (3-2): Read the hidden data.
  Step (3-3): Decrypt the data using the file name as a key.
  Step (3-4): Create a file using hidden file name.
  Step (3-5): Write into the created file the decrypted data.
End;
  Else
  Step (4): Display a message (no hidden information).
End.
```

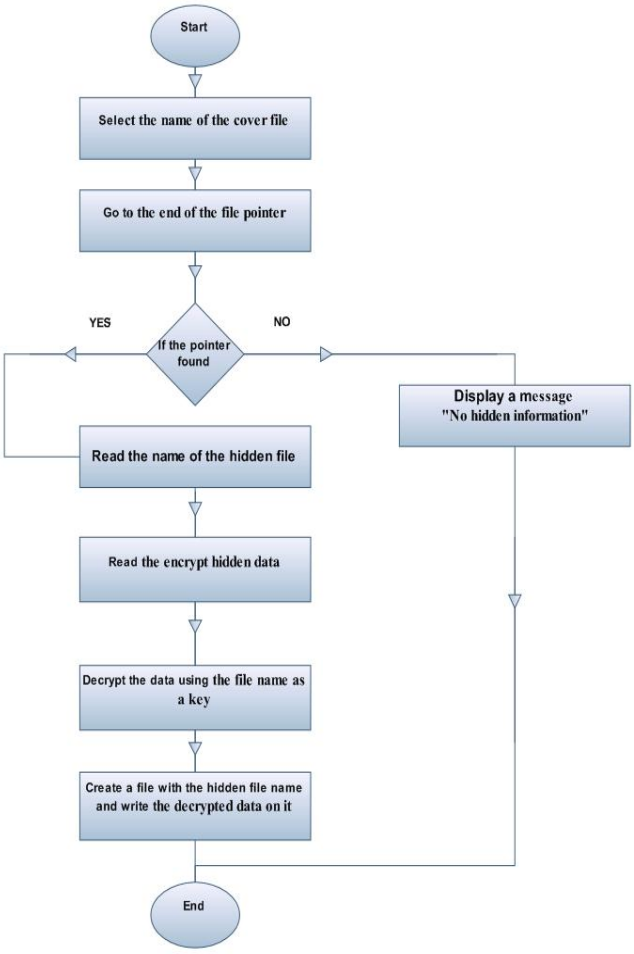



Fig. 5 Retraction Operation

## IV. Testing of the System

There are two fundamental approaches to identifying test cases, these are know 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 Fig. 6.

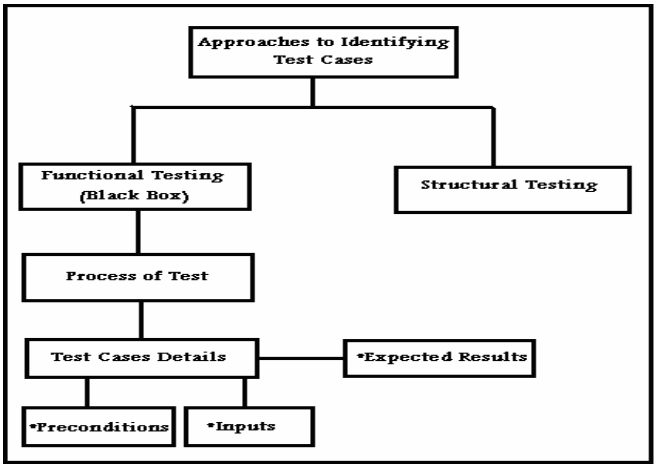



Fig. 6 Approaches to Identifying Test Cases

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 Fig. 7. Black-box testing has some important advantages:

It doesn't require that we see the code we are 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 we'd like, because it has to work without the implementation. But it's 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.

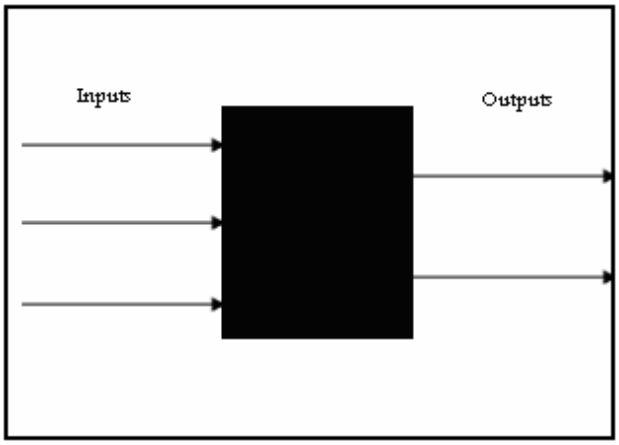



Fig. 7 Black box

*A. Process of the Test*

1. Test Case One:

Make compare between cover files size after and before hiding operation:

- Four tables to compare between size after and before hiding operation:
  - Table II: different size for cover with different type of the EXE files and same size for information of each type for multimedia files (text, video, audio and image).
  - Table III: same size for cover with same type of the EXE files and different size for information of each type for multimedia files (text, video, audio and image).
  - Table IV: different size for cover with same type of the EXE files and Same Size for information of each type for multimedia files (text, video, audio and image).
  - Table V: different size for cover with same type of the EXE files and different size for information of each type for multimedia files (text, video, audio and image).

2. Test Case Two:

Testing 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 divides to:
  i. First picture for text.
  ii. Second picture for image.
  iii. Third picture for video.
  iv. Fourth picture for audio.

3. Test Case Three:

Testing for Scanning Result (undetectable by antivirus software):

- Four pictures approve the cover (EXE Files) undetectable for antivirus software after the hiding operation and these pictures divides to:
  i. First picture for text.
  ii. Second picture for image.
  iii. Third picture for video.
  iv. Fourth picture for audio.

*B. 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).

- Preconditions**:**
  1. Installation (Microsoft Windows XP for Any Version or Vista).
  2. Installation (Microsoft Visual Studio 2005).
  3. Installation (Microsoft .NET Framework SDK v2.0).
  4. Installation (Microsoft Excel Worksheet 2003 Or 2007).
  5. Installation (Microsoft Office Word Document 2003 or 2007).
  6. Installation (Software Antivirus).
  7. Installation (Real Player Programme).
  8. Installation (Jet Audio Programme).
  9. Installation (ACDSEE Programme).
  10. System application for this research.

- Inputs**:**

The system has two types of inputs:

- Inputs for cover (EXE Files):
  - Five types of cover (EXE Files) for different size.
- Inputs for information hidden:
  - Four text for different size.
  - Four image for different size.
  - Four video for different size.
  - Four audio for different size.

Open Science Index, Computer and Information Engineering Vol:3, No:6, 2009 publications.waset.org/1349/pdf

Open Science Index, Computer and Information Engineering Vol:3, No:6, 2009 publications.waset.org/1349/pdf

TABLE I
INPUTS FOR TEST CASES

| Name of Inputs | Type of Inputs | Size of Inputs |
|---|---|---|
| Cover 1 | VMware Player Setup | 3.84MB |
| Cover 2 | Windows XP Setup | 1.32MB |
| Cover 3 | SSH | 520KB |
| Cover 4 | Net beans Editor Setup | 21.7MB |
| Cover 5 | 1. JDK Setup.<br>2. JDK Setup.<br>3. JDK Setup | 65.6MB<br>72.9MB<br>77.4MB |
| Text 1 | Word Document | 10.5 KB |
| Text 2 | Word Document | 80.0 KB |
| Text 3 | Word Document | 1.69MB |
| Text 4 | Word Document | 4.06MB |
| Video 1 | Real Player | 3.55MB |
| Video 2 | Real Player | 126MB |
| Video 3 | Real Player | 181MB |
| Video 4 | Real Player | 294MB |
| Audio 1 | Jet Audio | 1.83MB |
| Audio 2 | Jet Audio | 2.35MB |
| Audio 3 | Jet Audio | 3.12MB |
| Audio 4 | Jet Audio | 4.31MB |
| Image 1 | JPEG | 156KB |
| Image 2 | JPEG | 595KB |
| Image 3 | JPEG | 1.20MB |
| Image 4 | JPEG | 2.98MB |

- Expected Results:
- Secure cover (EXE Files).
- There are no limitations on the hidden files size.
- The hidden information can be of any type for multimedia files.
- These cover (EXE Files) usage after the hiding operation.
- These cover (EXE Files) undetectable for antivirus software after the hiding operation.

### *C. Test Case One*

In this test case can be shown tables for cover files and information hidden after and before hiding operation for all types of multimedia files (text, image, audio and video),which related with this system, approve these cover (EXE Files) are secure and There are no limitations on the hidden files size.

TABLE II
DIFFERENT SIZE FOR COVER WITH DIFFERENT TYPE OF THE EXE FILES AND SAME SIZE FOR INFORMATION OF EACH TYPE FOR MULTIMEDIA FILES (TEXT, IMAGE, AUDIO AND VIDEO)

| Before Hide Operation | | | | After Hide Operation |
|---|---|---|---|---|
| Information Hidden | No of Cover | Size of IH /Bytes | Size of Cover/Bytes | Size of Cover /Bytes |
| Text 1 | 1 | 10,752 | 4,027,802 | 4,038,610 |
| Text 1 | 2 | 10,752 | 1,388,544 | 1,399,352 |
| Text 1 | 3 | 10,752 | 532,480 | 543,288 |
| Image 1 | 1 | 160,048 | 4,027,802 | 4,187,906 |
| Image 1 | 2 | 160,048 | 1,388,544 | 1,548,648 |
| Image 1 | 3 | 160,048 | 532,480 | 692,584 |
| Audio 1 | 1 | 1,929,553 | 4,027,802 | 5,957,410 |
| Audio 1 | 2 | 1,929,553 | 1,388,544 | 3,318,152 |
| Audio 1 | 3 | 1,929,553 | 532,480 | 2,462,088 |
| Video 1 | 1 | 3,727,756 | 4,027,802 | 7,755,602 |
| Video 1 | 2 | 3,727,756 | 1,388,544 | 5,116,344 |
| Video 1 | 3 | 3,727,756 | 532,480 | 4,260,280 |

TABLE III
SAME SIZE FOR COVER WITH SAME TYPE OF THE EXE FILES AND DIFFERENT SIZE FOR INFORMATION OF EACH TYPE FOR MULTIMEDIA FILES (TEXT, IMAGE, AUDIO AND VIDEO)

| Before Hide Operation | | | | After Hide Operation |
|---|---|---|---|---|
| Information Hidden | No of Cover | Size of IH /Bytes | Size of Cover/Bytes | Size of Cover /Bytes |
| Text 2 | 4 | 84,992 | 22,806,060 | 22,891,108 |
| Text 3 | 4 | 1,782,272 | 22,806,060 | 24,588,388 |
| Text 4 | 4 | 4,262,400 | 22,806,060 | 27,068,516 |
| Image 2 | 4 | 609,773 | 22,806,060 | 23,415,876 |
| Image 3 | 4 | 1,261,562 | 22,806,060 | 24,067,668 |
| Image 4 | 4 | 3,133,684 | 22,806,060 | 25,939,796 |
| Audio 2 | 4 | 2,467,697 | 22,806,060 | 25,273,812 |
| Audio 3 | 4 | 3,276,272 | 22,806,060 | 26,082,388 |
| Audio 4 | 4 | 4,528,843 | 22,806,060 | 27,334,948 |
| Video 2 | 4 | 132,398,133 | 22,806,060 | 155,204,244 |
| Video 3 | 4 | 190,454,028 | 22,806,060 | 213,260,132 |
| Video 4 | 4 | 309,314,052 | 22,806,060 | 332,120,164 |

Open Science Index, Computer and Information Engineering Vol:3, No:6, 2009 publications.waset.org/1349/pdf

TABLE IV
DIFFERENT SIZE FOR COVER WITH SAME TYPE OF THE EXE FILES AND SAME SIZE FOR INFORMATION OF EACH TYPE FOR MULTIMEDIA FILES (TEXT, IMAGE, AUDIO AND VIDEO)

| Before Hide Operation | | | | After Hide Operation |
|---|---|---|---|---|
| Information Hidden | No of Cover | Size of IH/Bytes | Size of Cover/Bytes | Size of Cover/Bytes |
| Text 1 | 5 | 10,752 | 68,830,616 | 68,841,424 |
| Text 1 | 5 | 10,752 | 76,445,080 | 76,455,888 |
| Text 1 | 5 | 10,752 | 81,208,728 | 81,219,536 |
| Image 1 | 5 | 160,048 | 68,830,616 | 68,990,720 |
| Image 1 | 5 | 160,048 | 76,445,080 | 76,605,184 |
| Image 1 | 5 | 160,048 | 81,208,728 | 81,368,832 |
| Audio 1 | 5 | 1,929,553 | 68,830,616 | 70,760,224 |
| Audio 1 | 5 | 1,929,553 | 76,445,080 | 78,374,688 |
| Audio 1 | 5 | 1,929,553 | 81,208,728 | 83,138,336 |
| Video 1 | 5 | 3,727,756 | 68,830,616 | 72,558,416 |
| Video 1 | 5 | 3,727,756 | 76,445,080 | 80,172,880 |
| Video 1 | 5 | 3,727,756 | 81,208,728 | 84,936,528 |

TABLE V
DIFFERENT SIZE FOR COVER WITH SAME TYPE OF THE EXE FILES AND DIFFERENT SIZE FOR INFORMATION OF EACH TYPE FOR MULTIMEDIA FILES (TEXT, IMAGE, AUDIO AND VIDEO)

| Before Hide Operation | | | | After Hide Operation |
|---|---|---|---|---|
| Information Hidden | No of Cover | Size of IH/Bytes | Size of Cover/Bytes | Size of Cover /Bytes |
| Text 2 | 5 | 84,992 | 68,830,616 | 68,915,664 |
| Text 3 | 5 | 1,782,272 | 76,445,080 | 78,227,408 |
| Text 4 | 5 | 4,262,400 | 81,208,728 | 85,471,184 |
| Image 2 | 5 | 609,773 | 68,830,616 | 69,440,432 |
| Image 3 | 5 | 1,261,562 | 76,445,080 | 77,706,688 |
| Image 4 | 5 | 3,133,684 | 81,208,728 | 84,342,464 |
| Audio 2 | 5 | 2,467,697 | 68,830,616 | 71,298,368 |
| Audio 3 | 5 | 3,276,272 | 76,445,080 | 79,721,408 |
| Audio 4 | 5 | 4,528,843 | 81,208,728 | 85,737,616 |
| Video 2 | 5 | 132,398,133 | 68,830,616 | 201,228,800 |
| Video 3 | 5 | 190,454,028 | 76,445,080 | 266,899,152 |
| Video 4 | 5 | 309,314,052 | 81,208,728 | 390,522,832 |

For all tables above in test case one can be concluding:

There are no limitations on the hidden files size inside the cover files so can be hide different size inside the EXE files as shown in the Table II and Table III.

The attacker can't be attack the information hiding that's because can't guess the EXE files size because the EXE files size don't have Constant size as shown in the Table IV and Table V, where it can be different size for the same type of EXE files like cover file number 5 they have three sizes in same type of cover file.

### *D. Test Case Two*

In this test case can be shown Picture for 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 VI
INPUTS AND OUTPUTS FOR TEST CASE TWO

| Before Hiding Operation | | After Hiding Operation |
|---|---|---|
| No of Cover | Information Hidden | Usage the EXE Covers |
| 1 | Text1 | Figure 5.3 |
| 2 | Image1 | Figure 5.4 |
| 3 | Video 1 | Figure 5.5 |
| 5 | Audio 1 | Figure 5.6 |

- Text:

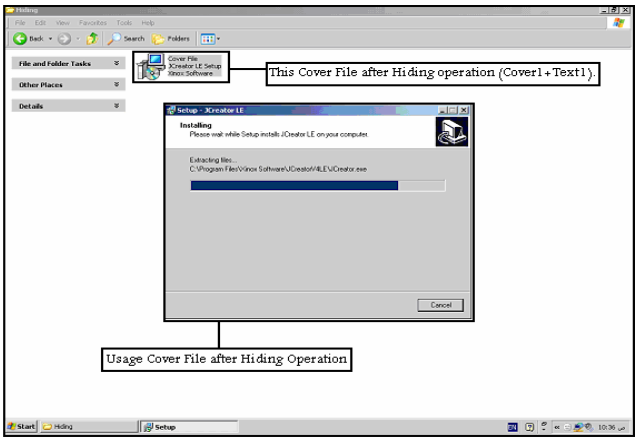



Fig. 8 After Hiding Operation inside the (Hiding Folder) that Executable File (Cover 1) Still Working

- Image:

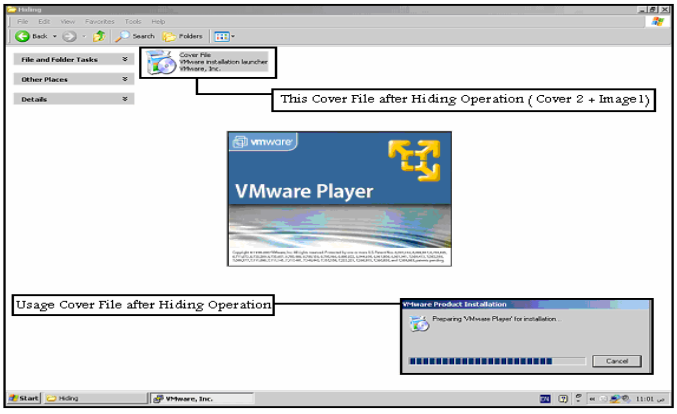



Fig. 9 After Hiding Operation inside the (Hiding Folder) that Executable File (Cover 2) Still Working

Open Science Index, Computer and Information Engineering Vol:3, No:6, 2009 publications.waset.org/1349/pdf

- Video

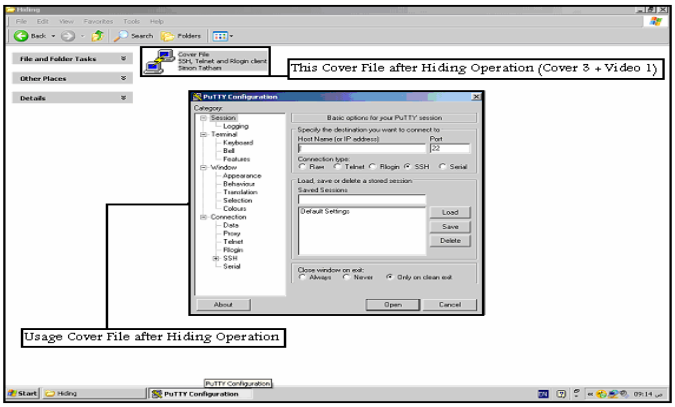



Fig. 10 After Hiding Operation inside the (Hiding Folder) that Executable File (Cover 3) Still Working

- Audio

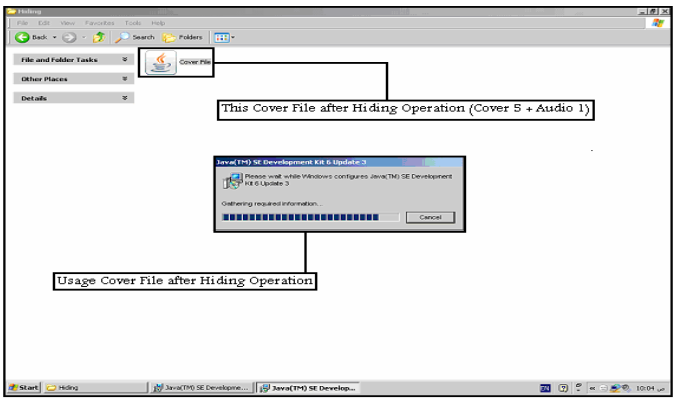



Fig. 11 After Hiding Operation inside the (Hiding Folder) that Executable File (Cover 5) Still Working

### *E. Test Case Three*

In this test case can be shown picture for cover files after hiding operation for all types of multimedia files (text, image, audio and video), which related with this system, approve these cover (EXE Files) undetectable for antivirus software after the hiding operation.

TABLE VII
INPUTS AND OUTPUTS FOR TEST CASE THREE

| Before Hiding Operation | | After Hiding Operation |
|---|---|---|
| No of Cover | Information Hidden | EXE Cover Undetectable for Antivirus |
| 1 | Text1 | Figure 5.7 |
| 2 | Image1 | Figure 5.8 |
| 3 | Video 1 | Figure 5.9 |
| 5 | Audio 1 | Figure 5.10 |

- Text:

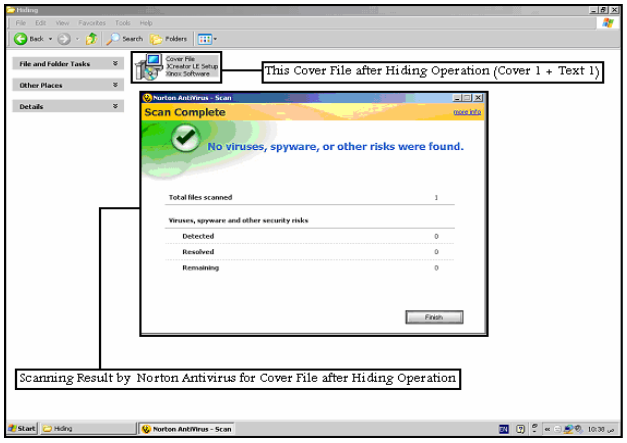



Fig. 12 The Executable File (Cover 1) Inside (Hiding Folder) Immune to Anti-virus Program

- Image:

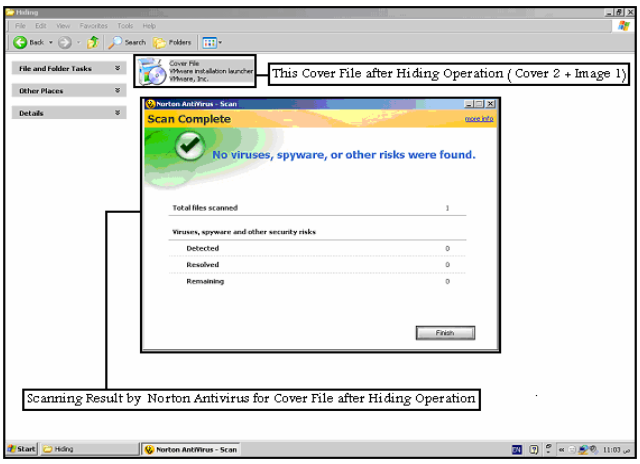



Fig. 13 The Executable File (Cover 2) Inside (Hiding Folder) Immune to Anti-virus Program

- Video

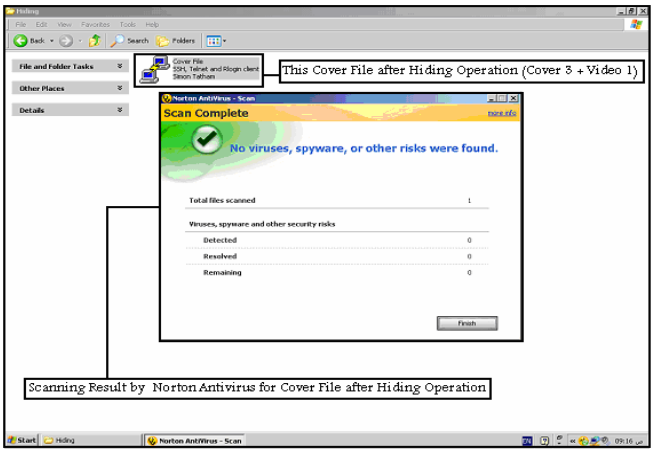



Fig. 14 The Executable (Cover 3) File Inside (Hiding Folder) Undetectable by Anti-virus program

Open Science Index, Computer and Information Engineering Vol:3, No:6, 2009 publications.waset.org/1349/pdf

- Audio

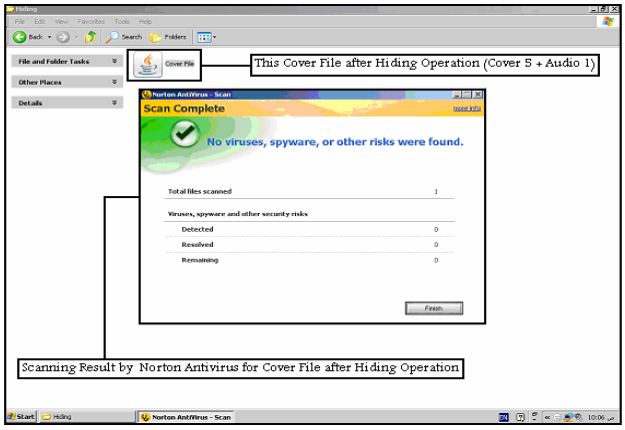



Fig. 15 The Executable (Cover 5) File Inside (Hiding Folder) Immune to Anti-virus Program

## V. Evaluation of the System

1. There are no boundaries on the hidden files size within the cover files so can be hide different size inside the EXE files.
2. The executable file still works after its use as cover for embedding data.
3. The executable file undetectable for Norton antivirus software after the hiding operation**.**
4. The size of cover files to be used in testing **(**520KB-77.4MB**),** the text (10.5KB-4.06MB), the Image (156KB-2.98MB), the audio (1.83MB-4.31MB) and the video (3.55MB-295MB), is found that when the size of cover EXE files upgrade, the secure is very height, when the information hidden less inside the cover ,the cover file in this case has been more secure, from the Information which is listed in the above tables ( Table I, Table II, Table III, Table IV and Table V), concludes that:

• The proportion of potential discovery of embedded data (Text 2%, Image 3%, Audio 4%, Video 5%).

• The percentage of success achieved by the innovative system (Text 97%, Image 97%, Audio 96%, Video 95%).

TABLE VIII
Conclusion of the Evaluation

| Information | What volume of information that can hide inside the (EXE)cover File? | Is it that executable file still works after its use as cover for Embedding data? | Undetectable Antivirus | What proportion of potential discovery of Embedding data? | What percentage of success achieved by the innovative system? |
|---|---|---|---|---|---|
| Video | Any size without limitation | Yes | Success | 5% | 95% |
| Audio | Any size without limitation | Yes | Success | 4% | 96% |
| Text | Any size without limitation | Yes | Success | 2% | 98% |
| Image | Any size without limitation | Yes | Success | 3% | 97% |

## VI. 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 research, because most users of any system cannot alter or modify the content of these files.

We get the following conclusions:

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 anti virus 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 size of cover file, so the hiding method makes the relation between the cover and the message independent. 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) because there are no limitations or restrictions on the type of the message (image, sound, text).

## VII. Suggestions for Future Work

There are many suggestions for improving the proposed system, the main suggestions are:

- Developing the method which is used in proposed system to deal with other PE files such as "dll', "sys", "cpl", and "ocx".
- Improvement of the security of hiding technique by adding compression function of the message before the hidden operation.
- Developing the proposed system to deal with other executable files created by other operating systems like (LINUX, UNIX, OS/2).
- Improvement of the security of the proposed system by changing the encryption methods for other methods such as (MD5, BLOWFISH)

## Acknowledgement

This work was supported in part by the University of Malaya, Kuala Lumpur Malaysia.

## References

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

[2] Zaidan, B. B., Zaidan A. A. and Othman F., "Enhancement of the amount of hidden data and the quality of image," Faculty of Computer Science and Information Technology, University of Malaya, Kuala Lumpur, Malaysia,2008.

[3] 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.

[4] 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.

[5] 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.

[6] 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,

[7] Johnson, N. F. S. D, Z., "Information Hiding: Steganography and Watermarking-Attacks and Countermeasures", Center for Secure Information Systems (CSIS), Boston/Dordrecht/London, George Mason University, 2006.

[8] Katzenbeisser, S. P., A. P, "Information Hiding Techniques for Steganography and Digital watermarking", available from: Artech house pub, 2005.

[9] G. Doërr and J. Dugelay, "Security pitfalls of frame –by-frame approaches to video watermarking," IEEE Transactions on Signal Processing, vol. 52, 2004, pp. 2955-2964.

[10] Mehdi Kharrazi, Husrev T. Sencar and Nasir Menon, "Image steganography: concepts and practice," Lecture Notes on Computer Science, vol. 2939, 2004, pp. 204-211.

[11] Katzenbeisser S. & Petitcolas, F. A., "Information Hiding Techniques for Steganography and Digital Watermarking", Artech House, USA, 2001.

[12] Mbaugh, S. E., "Computer vision and image processing, a practical approach using CVIP tools," Ph.D., 1998.

Open Science Index, Computer and Information Engineering Vol:3, No:6, 2009 publications.waset.org/1349/pdf