Analysis of Natural Language Steganography

Shaifizat Mansor shaifizat@kedah.uitm.edu.my
School of Computer Science
Universiti Sains Malaysia (USM)
Minden, Pulau Pinang, 11800, MALAYSIA

Roshidi Din roshidi@uum.edu.my
School of Computer Science
Universiti Sains Malaysia (USM)
Minden, Pulau Pinang, 11800, MALAYSIA

Azman Samsudin azman@cs.usm.my
School of Computer Science
Universiti Sains Malaysia (USM)
Minden, Pulau Pinang, 11800, MALAYSIA

ABSTRACT

The technology of information hiding through an open network has developed rapidly in recent years. One of the reasons why we need tools to hide message, is to keep secret message concealed from unauthorized party. Steganography is one of the techniques in sending secret message. In this paper, several software metrics were used to analyze the common criteria in steganographic tools and measure the complexity of the tools in hiding message. Several criterias have been chosen: Percent Lines with Comments (PLwC); Average Statements per Function (ASpF) and Average Block Depth (ABD) to measure the tools complexity. The analysis process has been implemented using a single Linux platform.

Keywords: Steganography, Text Steganography, Secret Message, Software Metric.

1. INTRODUCTION

Recently, millions of documents are produced and easily accessed in the Internet [1]. Thus, the information of these documents needs to be secured and protected because the activities of document analysis [2]. Steganography is one of the popular areas in information protection. The purpose of steganography is to establish communication between two parties whose existence is unknown to a possible attacker [3]. If this is done correctly, the exchanged messages should not arouse any suspicion since the communicated information has an innocent looking and the communication itself does not require any secret key as part of its information hiding process. In text, this can be done in many ways such as inclusion of line break characters, and multiple spacing that represents a hidden message.

Steganography technique is not a new technique [4]. They are some older practices in message hiding such as invisible ink, tiny pin punctures on selected characters and pencil mark on typewritten characters. In term of the key management, steganography is more secure than cryptography [5]. If The goal of steganography is to hide secret message in such a way that it does not arouse any eavesdropper's suspicion. Steganography is characterized as a process of hiding message in cover signal so that the message can be extracted unknowingly to the public at the receiving end. Steganography can be divided into two broad categories namely technical steganography and natural language steganography. Technical steganography is a technique of hiding information onto another medium such as image, audio, video or other digitally represented code invisibly [6]. On the other hand, natural language steganography is the art of using the natural language to conceal secret message [7]. Natural language steganography focuses on hiding information in text steganography, linguistic steganography and its hybrid as shown in FIGURE 1.

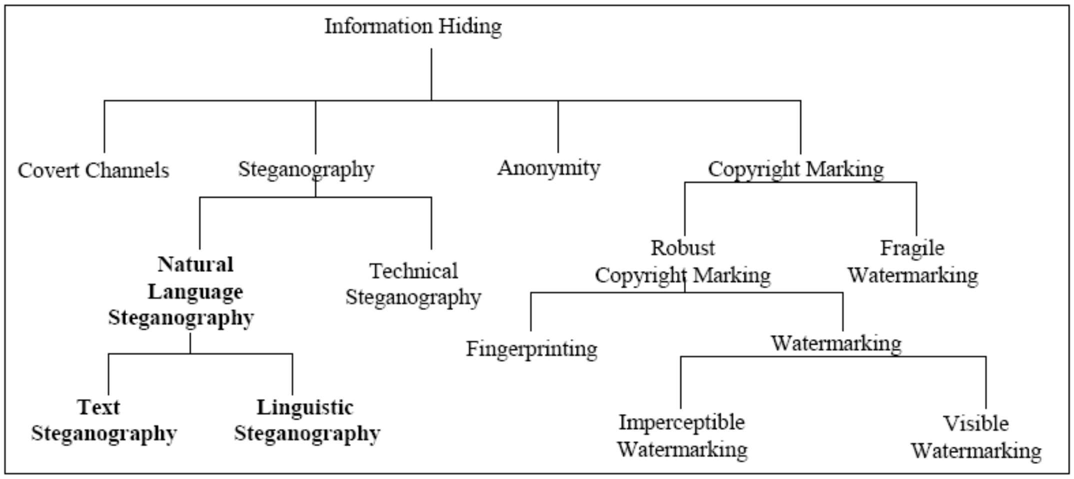

FIGURE 1: Field of Information Hiding and Its Classification (Adopted from [8]).

One of the linguistic steganography strength is the ability to hide a secret message during sending process [9]. The efficiency and the level of complexity to encode the hidden message is one of the linguistic steganography issues. Another important issue is the speed performance of the tools in executing the encoding process [10]. This issue raises a question on how to examine the execution time and tools complexity.

In order to improve the productivity and development of steganographic tools, the evaluation of existing steganographic tools must be carried out and use as a yardstick to improve the quality of software industry [11] especially in natural language steganography. Equally important, is the attacks analysis on steganographic tools which is deemed essential in evaluating the steganographic tools performance in order to improve the steganography algorithm [12]. Therefore, the natural language steganographic tools should be examined both from the software metric perspective as well as their robustness against attack. Thus, our main objective of this study is to analysis the performance of natural language steganographic tools based on these two perspectives.

The rest of the paper is organized as follows: In Section 2 we introduce text steganography tools that are currently being used. Section 3 discusses the parameter measurement of the selected steganographic tools in message hiding. In Section 4 we discuss the measurement of the software metric including Percent Lines with Comments *(PLwC)*, Average Statements per Function *(ASpF)*, and Average Block Depth *(ABD).* Section 4 also discusses execution time of the steganographic tools. Section 5 provides a discussion on the evaluation of the text steganographic. Section 6 is the conclusion of this paper.

2. TEXT STEGANOGRAPHY

Natural language steganographic tools and techniques are becoming more widespread and need to be evaluated [13]. One of the methods in evaluating natural language steganographic tools is by employing software metric [14], which is important in determining the tools efficiency. Among the components of software metrics are cost and effect estimation, productivity measures, quality measures, reliability tools, structural complexity metrics and execution time.

This study used text steganographic tools in order to analyze the field of natural language steganography. Numerous tools have been identified as text steganographic tools [15-22]. They are: Texto, Stego, SNOW, Stegparty, Steganos, Snowdrop, PGPn123, and FFEncode. TABLE 1 shows the description of each steganography tools.

No.	Tools	Platform	Description of encoding process
1.	Texto	DOS:WIN(Unix Linux)	Transform uuencode or PGP-armoured ascii data
2.	Steganosaurus (Stego)	C: DOS	Encode binary to text based on the dictionary from a source document
3.	SNOW	C/C++: DOS WIN	Append tabs and white space to end of text lines
4.	Stegparty	Unix/Linux	Alter the text on spelling and punctuation
5.	Steganos	DOS	Looks at a list of words on text to find and match words
6.	Snowdrop	C	Embed the text in the least significant portions of some binary output
7.	PGPn123	Window front-end to PGP	Using PGP shell tools to hide text
8.	FFEncode	DOS	Using Morse code in null characters to encode message

TABLE 1: The Description of Text Steganography Tools.

Among the identified text steganoography tools, only Texto, Stego, SNOW, and Stegparty are being examined. The selection is based on similarity criterion such as the steganography approach (line-shift coding, word-shift coding and feature coding) and manipulation of dictionary/corpora. In addition, the four identified tools also share common standard source code (C/C++) which are accessible from the open source. The accessibility of the codes aid tremendously in the evaluation process.

2.1 Texto

Texto is a rudimentary text steganography program to facilitate the exchange of binary data. It is using a simple substitution cipher which transforms *uuencoded* or *pgp ascii-armoured ascii* data, especially encrypted data into English sentences so that the text will look apparently reasonable during data transmission. FIGURE 2 shows the flow of Texto program.

Each symbol is replaced by nouns, verbs, adjectives, and adverbs in the preset sentence structures without punctuation or "connecting" words through English sentences. However, not all of the words in the resulting English are significant to the Texto program. Usually, the output of Texto is close enough to normal English text that it will slip by any kind of automated scanning.

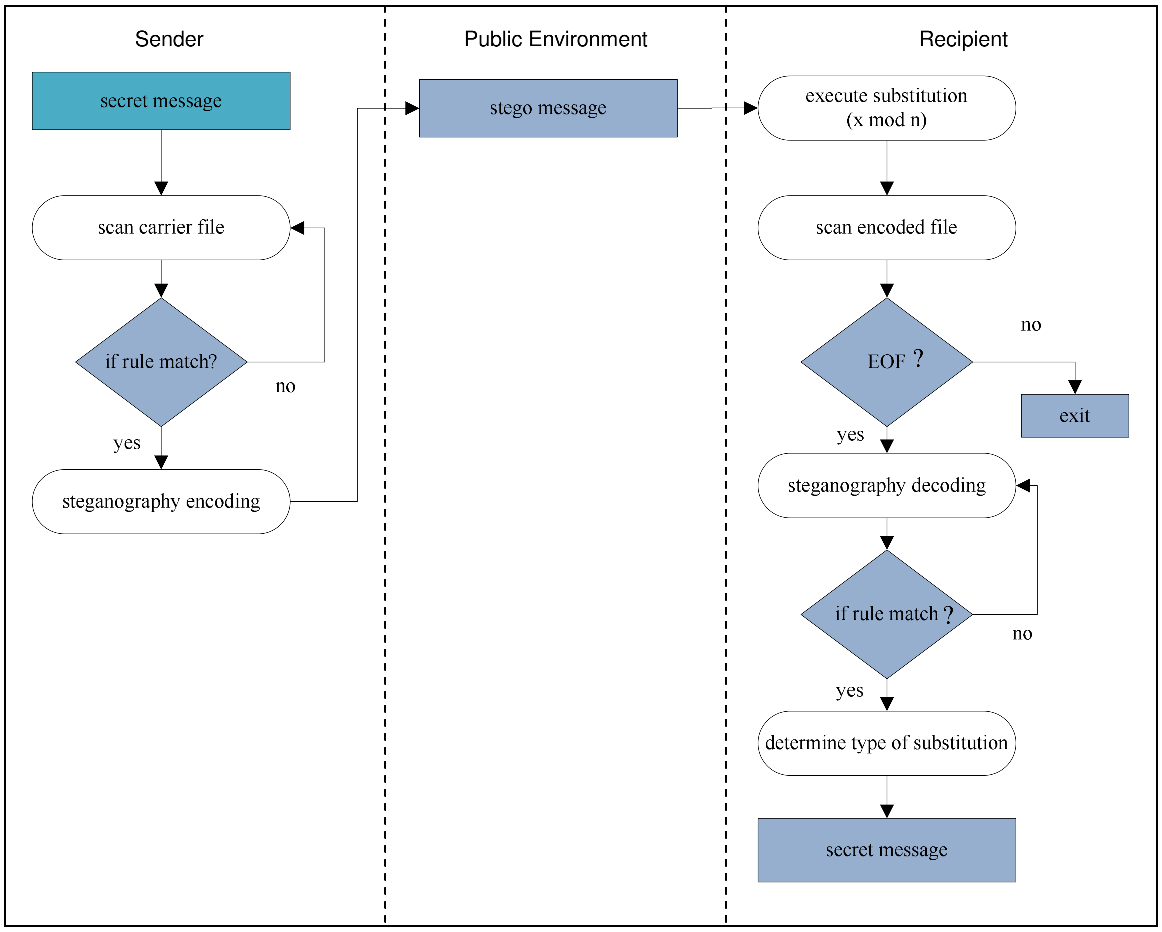

FIGURE 2: The Process Flow of Texto Steganography Tool.

2.2 Steganosaurus (Stego)

Steganosaurus also known as Stego uses a line-shift coding method which is a plain text steganography utility which encodes a binary file into a gibberish text based on either a spelling dictionary or words taken from a text document. The output of Stego converts any binary file into nonsense text based on a dictionary from a source document. The output of stego is nonsense but statistically resembles text in the language of the dictionary supplied. A human reader will instantly recognize it as gibberish while to eavesdroppers; the encrypted messages may consider it to be unremarkable, especially if a relatively small amount of such text appears within a large document. Stego makes no attempt, on its own, to prevent the message from being read. It is the equivalents of a code book with unique words as large as the dictionary.

Based on FIGURE 3, text created by stego uses only characters in the source dictionary or document. It means that during encoding process, the message will be converted into an output text file using the specified (or default) dictionary. The specified file called '*dictfile'* is used as the dictionary to encode the file. The dictionary is assumed to be a text file with one word per line, containing no extraneous white space, duplicate words, or punctuation within the words. All duplicate words and words containing punctuation characters are deleted, and each word appears by itself on a separate line. The Stego text will look less obviously gibberish if the output is based upon template sentence structures filled in by dictionaries. The efficiency of encoding a file as words depends upon the size of the dictionary used and the average length of the words in the dictionary. Preprocessing a text file into dictionary format allows it to be loaded much faster in subsequent runs of stego. Another file namely '*textdict*', is created to build the dictionary for input file used during encoding or decoding process. The '*textdict'* is scanned and words, consisting of alphanumeric characters, are extracted. Duplicate words are automatically discarded to prevent errors in encoding and decoding processes.

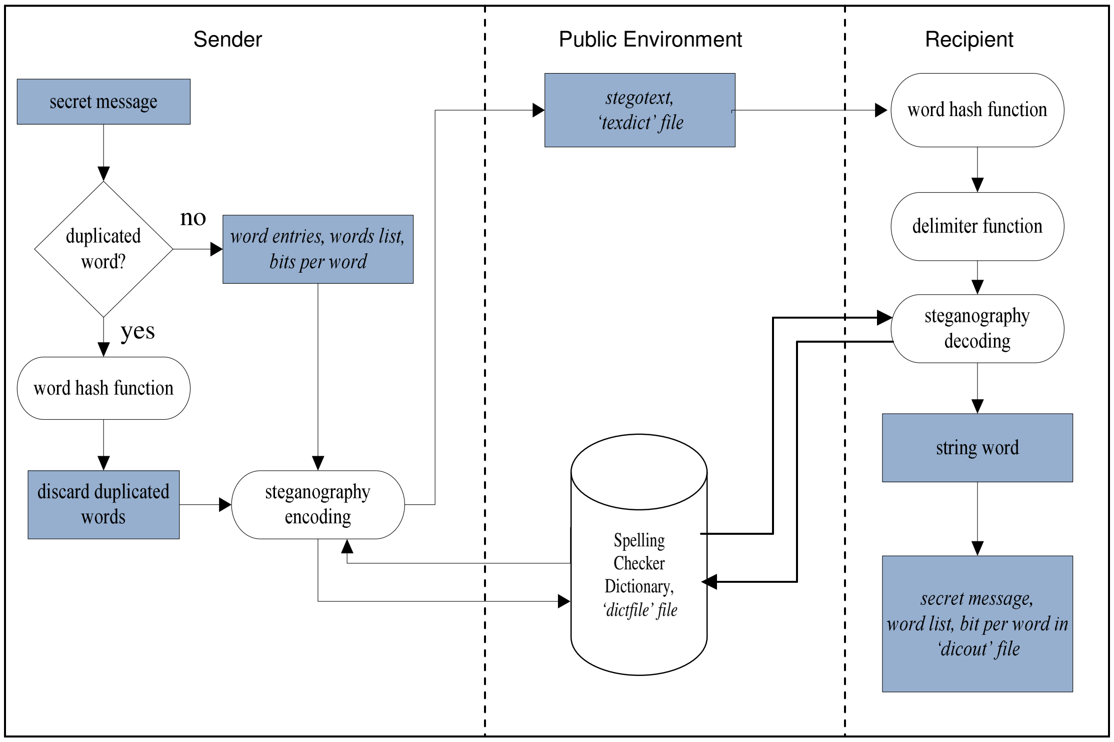

FIGURE 3: The Process Flow of Stego Steganography Tool.

Stego can then be applied to the encrypted output, transforming it into seemingly innocuous text for transmission, so it can be sent by sender through media, such as electronic mail, which cannot transmit binary information. If the medium used to transmit the output of stego, *'textdict'* cannot correctly deliver such data; the recipient will be unable to reconstruct the original message. To avoid this problem, the sender can either encode the data before transmission or use a dictionary which contains only characters which can be transmitted without loss. The decoding process by receiver to recover the original message, *'dictout'*, must be carried out using the same dictionary as encoding process because the ability to recognize gibberish in text is highly language dependent. Usually, the default dictionary is the system spelling checker dictionary. However, this dictionary is not standard across all systems.

2.3 SNOW

Steganographic **N**ature **O**f **W**hitespace or SNOW, is a program for conceling messages and extracting messages in ASCII text file. This feature coding method conceals messages by appending tabs and spaces (known as whitespace) at the end of lines. Tabs and spaces are invisible to most text viewers, hence the steganographic nature of this encoding scheme.

This allows messages to be hidden in the ASCII text without affecting the text visual presentation. Since trailing spaces and tabs occasionally occur naturally, their existence should not be deemed sufficient to immediately alert an observer who stumbles across them.

The data is concealed in the text file by appending sequences of up to 7 spaces, interspersed with tabs. This usually allows 3 bits to be stored in every 8 columns. The SNOW program runs in two modes which are message concealment and message extraction as shown in FIGURE 4.

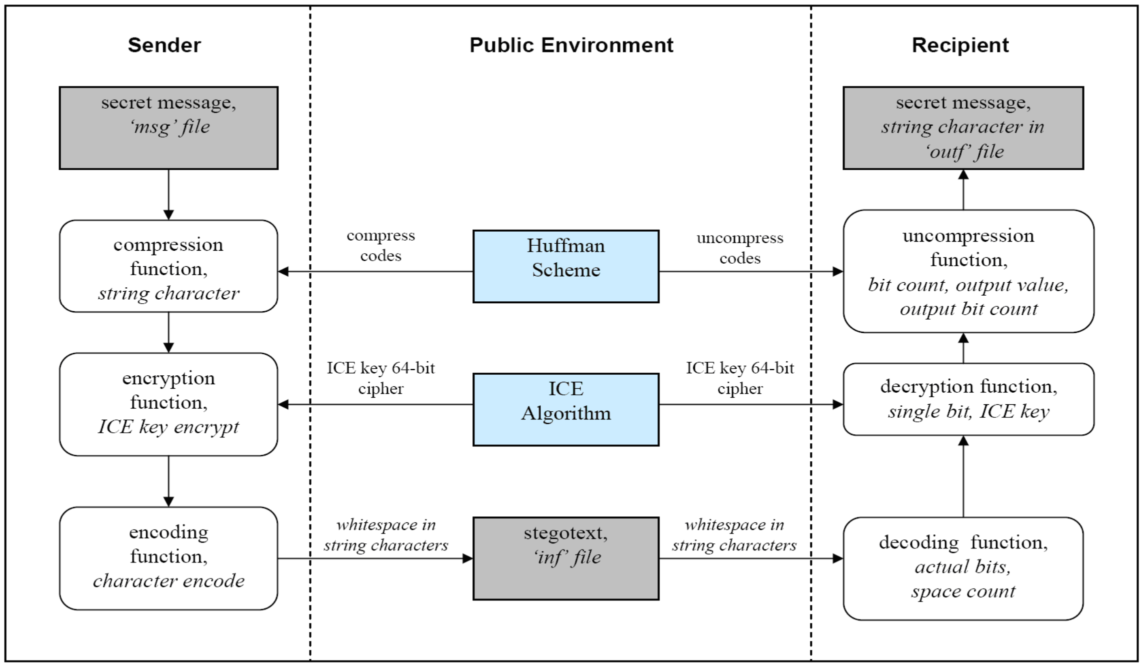
Sender
Public Environment
Recipient
secret message, 'msg' file
compression function, string character
encryption function, ICE key encrypt
encoding function, character encode
compress codes
Huffman Scheme
uncompress codes
ICE key 64-bit cipher
ICE Algorithm
ICE key 64-bit cipher
whitespace in string characters
stegotext, 'inf' file
whitespace in string characters
secret message, string character in 'outf' file
uncompression function, bit count, output value, output bit count
decryption function, single bit, ICE key
decoding function, actual bits, space count

FIGURE 4: The Process Flow of SNOW Steganography Tool.

There are three important steps involve in the concealing process which are;

i. *Compression* - used a rudimentary Huffman encoding scheme where the tables are optimized for English text. This was chosen because the *whitespace* encoding scheme provides very limited storage space in some situations, and a compression algorithm with low overhead was needed.

ii. *Encryption* - used an *ICE* encryption algorithm [17] with 64-bit block cipher. It runs on a 1-bit *cipher-feedback* (CFB) mode, which is quite inefficient (requiring a full 64-bit encryption for each bit of output).

iii. *Encoding scheme* – at the beginning of a message, a tab is added immediately after the text on the first line where it will fit. Tabs are used to separate the blocks of spaces. A tab is not appended to the end of a line unless the last 3 bits coded to zero spaces, in which case it is needed to show some bits are actually there.

While in extracting process, there are also three steps involved which are decoding, decryption and decompression. All of these steps are running on sequential during extraction process. After extraction process is completed, an extracted message is transferred to output text called *outf*.

2.4 Stegparty

Stegparty is a hiding information system that hides data inside a text file by using a set of rules defining various flexible components within the English language. Stegparty can hide small alterations to the message by matching the text and replacing it with small typos, grammatical errors, or equivalent expressions such as spelling and punctuation changes as shown in FIGURE 5. It is a unique data hiding method by creating misspellings inside original text files.

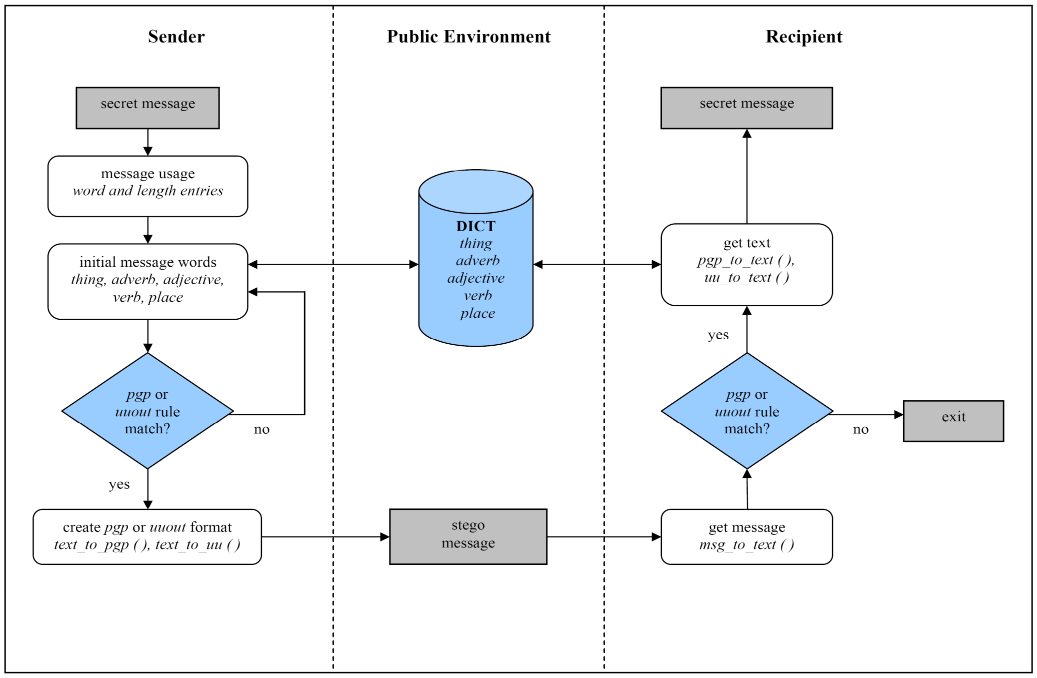

FIGURE 5: The Process Flow of Stegparty Steganography Tool.

3. EXPERIMENTAL WORK

This section discusses the parameter measurement of the selected steganographic tools during the hiding of a message. This study use software metric, such as dataset selection, execution time, and algorithm segment in order to analyse the performance of steganographic tools.

3.1 Software Metrics

In this analysis, three attributes of the software metric parameter have been used which are *Percent Lines with Comments (PLwC)*, *Average Statements per Function (ASpF)*, and *Average Block Depth (ABD)* [23-25].

a) *Percent Lines with Comments*

Percent Lines with Comments (*PLwC*) is used to determine the documentation level of selected tools. This analysis is very important for developers to understand the inner workings of each tool.

$$PLwC = \left(\frac{\text{Total Number of Lines}}{\text{Total Number of Comments}}\right) \times 100\%$$

$$\text{Percent Lines (\%)} = \frac{\sum_{i=1}^{1<m<2000} a_i}{\sum_{i=1}^{1<n<100} b_i} \times 100\% \qquad (1)$$

where

a = *Total Number of Lines*
b = *Total Number of Comments*

b) *Average Statements per Function (ASpF)*

ASpF calculates the average number of statements per function. Thus, *ASpF* can be used to determine the complexity of the each selected tool.

$$Average\ Statements\ per\ Function = \left(\frac{Total\ Number\ of\ Statements}{Number\ of\ Functions}\right)$$

$$\bar{x} = \frac{\sum_{i=1}^{1<n<100} c}{d} \qquad (2)$$

where
c = Total Number of Statement
d = Number of functions

c) *Average Block Depth (ABD)*

This analysis is used to determine the average depth of the available block in each of the selected tools. Higher value of *ABD* may lead to higher usage of memory space of the tools.

$$AverageBlockDepth = \left(\frac{Total\ Number\ of\ Nested\ Block\ Depth}{Block\ Depth}\right)$$

$$\bar{x} = \frac{\sum_{i=1}^{1<n<100} e}{f} \qquad (3)$$

where
$\bar{x}$ *= Average Block Depth*
e = Total Number of Nested Block
f = Block Depth

3.2 Data Set Selection
Text chosen dataset is one of the important components in benchmarking steganographic techniques [26]. Our study used a dataset of text which includes a variety of textures and sizes. In order to evaluate the text steganographic techniques, various file sizes have been categorized in four phases during evaluation process from phase I to phase IV, respectively. Phase I started with 10 bytes, followed by phase II with 100 bytes, phase III with 1000 bytes and end up with 20 kilobytes of plaintext files size in phase IV as shown in TABLE 2. For every size category, hundreds of files are created to ascertain the relation of time taken based on the different type of file size. Then, the result of execution time of each data files in the selected text steganographic tools during evaluation process is recorded.

Testing Phase	Size (bytes)
Phase I	10
Phase II	100
Phase III	1000
Phase IV	20 000

TABLE 2: Distribution of File Size Category.

3.3 Execution Time

To obtain the execution time of the whole process, different parameters are selected from all four steganography tools. To get a secret message to be encoded, different tools will transform this message in a four different ways. Stegparty will use code segment *secretfile* to encode message and later the message is stored in *codedfile* code segment. Stego secret message stored text in *secretmessage* code and result will be stored in code segments *cf.* Texto used *msgfile* code segment to store secret message and *engfile* code message for encoded message. In addition, SNOW tools use *–f* code in encoding secret message and *crf* code segment as an encoded message. From the encoding process, the execution time is recorded. *Start time, stop time* and *time taken* to encode message are recorded for every file sizes. The result of the time taken is discussed further in Section 4.2. TABLE 3 shows the various types of codes segment of hidden message and output file on retrieving execution time in each tool.

Tools	Encoded File	Coded File
Stegparty	*secretfile*	*codedfile*
Stego	*secretmessage*	*cf*
Texto	*msgfile*	*engfile*
SNOW	*-f*	*crf*

TABLE 3: Various Types of Codes Segment in Selected Text Steganographic Tools.

In determining the efficiency of execution time, a small segments of code is created in every steganographic tools. This code segment is activated when a *run* command is executed.

```
#include <time.h>
clock_t start, end;
double elapsed;
start = clock();
... /* Do the work. */
end = clock();
elapsed = ((double) (end - start)) / CLOCKS_PER_SEC;
```

4. EXPERIMENTAL RESULT

This section discusses the software metric measurement taken for Percent Lines with Comments *(PLwC)*, Average Statements per Function *(ASpF)*, and Average Block Depth *(ABD)*. This section also discusses execution time of the steganographic tools.

4.1 Software Metric Measurement

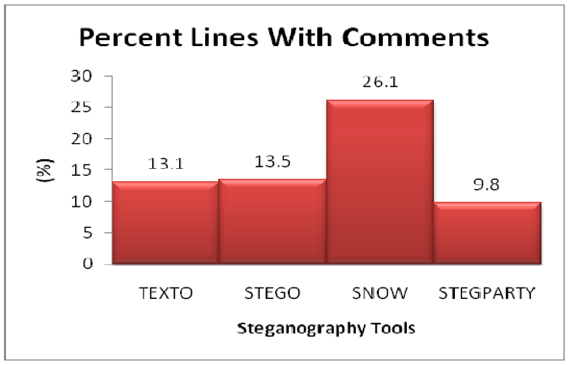

FIGURE 6: Percent Lines with Comments of The Chosen Text Steganographic Tools.

Based on FIGURE 6, the *PLwC* of the selected tools have been identified. A *PLwC* value of Texto is 13.1%, quite close to Stego *PLWC* value which is 13.5%. Another selected tool called Stegparty has recorded a percentage of 9.8% while the SNOW recorded a 26.1% which is the highest value among all tools. This analysis result indicates that SNOW has more documentation in its source code, compared to Texto, Stego and Stegparty.

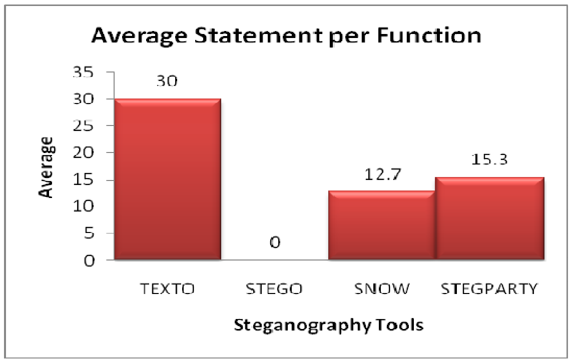

FIGURE 7: Average Statement per Function of The Chosen Text Steganographic Tools.

In analyzing the *ASpF* (see FIGURE 7), this study found that Texto has the highest *ASpF* value with 30 statements compared to Stego with its *ASpF* value of almost 0. While only 12.7 *ASpF* value for SNOW, Stegparty has recorded 15.3 for its *ASpF* value. Thus, SNOW and Stegparty have a comparable value for *ASpF.* The amount of code per function seems to be about the same for both tools.

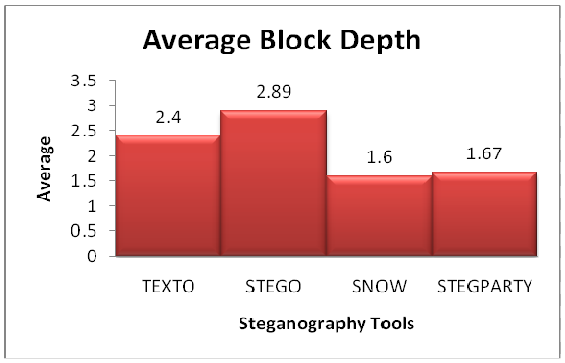

FIGURE 8: Average Block Depth (*ABD*) of The Chosen Text Steganographic Tools.

The value of *ABD* can be obtained by dividing the number of nested block depth with depth of block for every function. Based on FIGURE 8, the *ABD* for Texto and Stego are 2.4 and 2.89, respectively while *ABD* for SNOW is 1.6 and Stegparty is 1.67, consecutively. In term of block depth complexity, Stego and Texto are somewhat in the same league while another league comprises SNOW and Stegparty. Thus, it can be said that Texto and Stego are having about the same complexity while SNOW and Stegparty share almost the same complexity.

4.2 Execution Time Measurement

TABLE 4 has shown the analysis of execution time for the selected tools based on different file size, which are: 10 bytes, 100 bytes, 1000 bytes and 20 000 bytes. This study required LINUX platform and running on the Solaris 7 operating system under SunOS 5.7 version with 64-bit UltraSPARC microprocessor.

TOOLS / BYTES	Texto	Stego	SNOW	Stegparty
10	0.86	0.85	13.35	0.85
100	0.12	0.12	0.15	0.12
1000	0.01	0.01	0.07	0.01
20000	0.04	0.04	0.11	0.03

TABLE 4: Execution Time of The Chosen Text Steganographic Tools

It is found that Steparty requires 0.85 to 0.86 seconds to encode file size of 10 bytes to 1000 bytes. The running time increases significantly to 13.35 seconds when encoding file size of 20000 bytes. Stego requires much less time for encoding files where it requires 0.12 seconds to encode file size between 10 bytes to 100 bytes. Encoding file size of 20000 bytes does not change the running time very much as Stego takes only 0.15 seconds to encode the data. While Texto requires 0.01 seconds to encode file size of 10 bytes to 1000 bytes and takes an extra 0.06 seconds to encode file size of 20000 bytes. Likewise, SNOW is also in the same category as Stego and Texto. SNOW encodes file size 10 to 1000 bytes at between 0.03 and 0.04 seconds and taking only 0.11 seconds to encode file size of 20000 bytes.

5. DISCUSSION

This study provides the evaluation of text steganographic tools based on the criteria that has been selected. The result of the software metric and execution time have shown the level of complexity and speed performance, respectively.

In analyzing the software metric, this study has considered the Percent Lines with Comments (*PLwC*), Average Statement per Function (*ASpF*), and Average Block Depth (*ABD*) in order to measure productivity of software development. TABLE 5 shows a summarization of software metric for all four text steganographic tools.

METRIC / TOOLS	Texto	Stego	SNOW	Stegparty
Percent Lines With Comments (*PLwC*)	13.1	13.5	26.1	9.8
Average Statement per Functions (*ASpF*)	30	15.3	12.7	0
Average Block Depth (*ABD*)	2.4	2.89	1.6	1.7

TABLE 5: Software Metric of Text Steganographic Tools.

Result on Percent Lines with Comments (*PLwC*) indicates that SNOW has more documentation in its source code, a characteristic which is good in order to understand the flow of the tool. Compared to SNOW, Stegparty does not have much comment lines in its code. Thus, a Stegparty's developer may face some difficulties in modifying the source code due to lack of description on what each of the code does. Closer examination of Average Statement per Function (*ASpF*) for all source code reveals that Stego has small amount of statements with many functions. As such, its ratio between number of statements and number of functions seems to be quite imbalance with small number of statements and large amount of functions. On the other hand, Texto does not have many functions. Most of its statement is included in the main function. Having a non-function source code makes it difficult for future alteration if need arises. In term of modular programming [27], this study found that the coding style in Texto is weak. However, it contributes to the shortened development time because the modules can be implemented separately, thus increasing the flexibility and comprehensibility of the program [19]. For Average Block Depth (*ABD*), Stego is more complex than the other tools since its blocks traverse deeper. There is also a probability that it uses more stack memory since traversing deeper inside the block will make the parent variables to be pushed onto stack.

In analyzing the execution time, Texto is the fastest in term of encoding secret message inside a file. It does not depend on any carrier file like Stegparty. As such, encoding file size of 10 bytes will not be too much different from encoding file size of 20000 bytes. Stego and SNOW exhibit the same behavior as Texto. These two tools are also not dependent on carrier file size. As such the time required to encode files does not differ very much. It is also discovered that Stegparty took the longest time to encode files.

6. CONCLUSION

In conclusion, this study found that Texto has the highest complexity level and speed performance among the text steganographic tools followed by Stegparty, SNOW and Stegano. The *PLwC* and *AspF* metrics have shown to be an important system parameter to measure productivity of software development. The envisaged future work will involve performance evaluation on the efficiency of pre-encrypt and post-encrypt process. In addition, the software productivity need to be measured more precisely by taking into account not only the four attributes but also will involve more parameters. Finally this study is only running on a single Linux platform. Taking into account more platform and different machine used to compare software performance is considered as a future work.

7. REFERENCES

1. M. A. A. Murad, and T. Martin. *“Similarity-based estimation for document summarization using Fuzzy sets”*. International Journal of Computer Science and Security (IJCSS), Computer Science Journal Press, Kuala Lumpur, 1(4): 1 - 12, Nov/Dec 2007

2. B. V. Dhandra, and M. Hangarge. *“Morphological reconstruction for word level script identification”*. International Journal of Computer Science and Security (IJCSS), Computer Science Journal Press, 1(1): 41 - 51, May/June 2007

3. J. J. Eggers, R. Bäuml, and B. Girod. *“A communications approach to image steganography”*. In Proceedings of SPIE: Electronic Imaging, Security and Watermarking of Multimedia Contents IV, 4675:26-37, San Jose, CA, USA, January 2002

4. J. Zollner, H. Federrath, H. Klimant, A. Pritzmann, R. Piotraschke, A. Westfeld, G. Wicke, and G. Wolf. *“Modeling the security of steganographic systems”*. In 2nd International Workshop Information Hiding. Springer, Berlin/Heidelberg, German, vol. 1525: 345 - 255, 1998

5. C. Kant, R. Nath, and S. Chaudhary. “*Biometrics security using steganography”*. International Journal of Security (IJS), Computer Science Journal Press, 2(1): 1 - 5, Jan/Feb 2008

6. N. F. Johnson, S. Jajodia. *“Exploring steganography: seeing the unseen”*. IEEE Computer Magazine, 31(2):26 – 34, 1998

7. M. Chapman, G. I. Davida, and M. Rennhard. *“A practical and effective approach to large-scale automated linguistic steganography”*. In Proceedings of the Information Security Conference (ISC '01), Malaga, Spain, 156 -165, October 2001

8. F. A. P. Petitcolas, R. J. Anderson and M. G. Kuhn. *“Information hiding: A survey”*. In Proceedings of the IEEE on Protection of Multimedia Content, 87(7):1062 - 1078, July 1999

9. Z. Oplatkova, J. Holoska, I. Zelinka, and R. Senkerik. *"Steganography Detection by Means of Neural Networks".* 19th International Conference on Database and Expert Systems Application (DEXA '08), 2008

10. K. Ochiawai, H. Iwasaki, J. Naganuma, M. Endo, and T. Ogura. "*High Speed Software-based Platform for Embedded Softwar of A Single-Chip MPEG-2 Video Encoder LSI with HDTV Scalibility*". In Proceeding of the Conference on Design, Automation and Test in Europe: 1-6, 1999

11. D. Welzel, H. L. Hausen. *"A five steps method for metric-based software evaluation: effective software metrication with respect to quality standard"*. Journal of ACM, 39(2 –5):273 – 276, 1993

12. A. Westfield, A. Pfitzmann. *"Attacks on steganographic systems"*. In Proceedings of 3rd International Workshop Computer Science (IH '99) Germany, 1999

13. S. R. Baragada, M. S. Rao, S. Purushothaman, and S. Ramakrishna. *"Implementation of radial basis function neural network for image steganalysis".* International Journal of Computer Science and Security (IJCSS), Computer Science Journal Press, Kuala Lumpur, 1(4): 12 - 22, Jan/Feb 2008

14. S. Nystedt, C. Sandros. *"Software Complexity and Project Performance*". Master Thesis and Bachelor Thesis, University of Gothenburg, 1999

15. K. Maher. *"Texto"*. Underware Software Production Ltd. Inc.,1995, http://linkbeat.com/files/

16. J. Walker. *"Steganosaurus"*. 1997, http://www.fourmilab.ch/ or http://www.fourmilab.to/stego/

17. M. Kwan. *"SNOW"*. Darkside Technologies Pty Ltd ACN 082 444 246 Australia, 1998 http://www.darkside.com.au/snow/index.html

18. S. E. Hugg. *"StegParty"*. Hamco Software (COMETBUSTERS-DOM) 1249 Turkey Point Rd Edgewater, MD 21037 US, 1999 http://www.cometbusters.com/hugg/projects/stegparty.html

19. *Steganos* can be accessed at; http://zerblatt.forex.ee/~ftp/crypto/code/STEGANOS.ZIP

20. *Snowdrop* can be accessed at; http://linux.softpedia.com/get/Programming/Version-Control/snowdrop-23917.shtml

21. *PGPn123* can be accessed at; ftp://ftp.dei.uc.pt/pub/pgp/pc/windows/

22. *FFEncode* can be accessed at; http://www.rugeley.demon.co.uk/security/ffencode.zip

23. *METRIC DATA PROGRAM* can be accessed at http://mdp.ivv.nasa.gov/loc_metrics.html#PERCENT_COMMENTS

24. C. Johns. *Applied Software Measurement (3rd Edition). USA, 2008*

25. *Metrics Complexity* can be accessed at http://download.instantiatons.com/CodeProDoc

26. M. Kharrazi, H. T. Sencar, Memon. *"Benchmarking steganographic and steganalysis techniques"* Security, Steganography, and Watermarking of Multimedia Contents, 2005

27. R. P. Cook. *"An introduction to modular programming"*, 1995 from http://www.docdubya.com/belvedere/cpp/modular.html