12-2008

Forensic and Anti-Forensic Techniques for Object Linking and Embedding 2 (OLE2)-Formatted Documents

Jason M. Daniels
Utah State University

FORENSIC AND ANTI-FORENSIC TECHNIQUES FOR OBJECT LINKING AND EMBEDDING 2 (OLE2)-FORMATTED DOCUMENTS

by

Jason Daniels

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Science

Approved:

_____	_____
Robert F. Erbacher Major Professor	Scott Cannon Committee Member
_____	_____
Stephen W. Clyde Committee Member	Byron R. Burnham Dean of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2008

ABSTRACT

Forensic and Anti-Forensic Techniques for Object Linking and

Embedding 2 (OLE2)-Formatted Documents

by

Jason Daniels, Master of Science

Utah State University, 2008

Major Professor: Dr. Robert F. Erbacher
Department: Computer Science

Common office documents provide significant opportunity for forensic and anti-forensic work. The Object Linking and Embedding 2 (OLE2) specification used primarily by Microsoft's Office Suite contains unused or dead space regions that can be overwritten to hide covert channels of communication. This thesis describes a technique to detect those covert channels and also describes a different method of encoding that lowers the probability of detection.

The algorithm developed, called OleDetection, is based on the use of kurtosis and byte frequency distribution statistics to accurately identify OLE2 documents with covert channels. OleDetection is able to correctly identify 99.97 percent of documents with covert channel and only a false positive rate OF 0.65 percent.

The improved encoding scheme encodes the covert channel with patterns found in unmodified dead space regions. This anti-forensic technique allows the covert channel to

masquerade as normal data, lowering the probability that any detection tool is able to detect its presence.

(132 pages)

ACKNOWLEDGMENTS

special thanks to my family. My wife, Kristine, has been incredibly supportive in my efforts to complete this work. I want to thank my children for their understanding and prayers during this process. I am also thankful to the rest of my family who have always been there to support and encourage. Lastly, thanks to my parents for their unfailing effort to instill a spirit of independence and confidence in me that has led me to know I can accomplish anything I set out to do.

Thanks to my many teachers and colleagues whose instruction has helped me gain greater knowledge and understanding. Specifically, thanks to Dr. Robert Erbacher, whose insight and knowledge kept me going in the right direction.

Jason Daniels

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Forensic Computing

Stated simply, forensic computing is the science of collecting, discovering, and preserving digital data for use in court or in the protection of businesses, public entities, and private individuals. Put differently, "…computer forensics is considered to be the use of analytical and investigative techniques to identify, collect, examine and preserve evidence/information which is magnetically stored or encoded" [7:1].

Forensic computing is a multi-discipline science in which computer science joins forces with information systems, law enforcement, the legal community, and social science [2]. In addition, businesses, the military, academics, and individuals use forensic computing for their own purposes [2]. The forensic computing community has defined several models with which to handle digital devices involved in an investigation. Generally the approach is as follows [2, 21]:

1. Identification
2. Preparation
3. Approach of strategy
4. Preservation
5. Collection
6. Examination
7. Analysis
8. Presentation
9. Return of evidence

The overarching goal is to provide evidence proving the innocence or guilt of a party. A large part played by the computer science discipline, however, is the development of tools that enable investigators to examine and analyze (steps 6 and 7) the

collected data. Computer forensics specializes in finding digital data to assist in an investigation. However, this is not an easy task. Creating steganographic techniques, performing encryption, and the large amount of disk space that needs to be searched are by no means trivial tasks. In fact, much research has been done around the detection of stenographic algorithms and finding hidden data files in large disk spaces [9, 12, 14, 19, 26].

1.2 Anti-Forensics

Anti-forensics is the science of avoiding the detection of incriminating data either by destroying it entirely or hiding it from an investigator. Anti-forensic techniques can be broken into four different categories: destroying evidence, hiding information, elimination of sources, and counterfeiting evidence [13, 22].

The destruction of evidence can be either a physical or logical destruction. While the preferred method is to use a grinder on the hard drive platters, physical destruction may also include other methods, such as using a magnet to destroy a hard drive. Whatever the method of choice, the result is the same: making retrieval of information impossible. Logical destruction wherein the incriminating bytes are overwritten with random bytes is also a highly effective means of making data retrieval impossible. When the data is destroyed, forensic attempts to collect the evidence is obviously hampered.

Hiding evidence of foul play or other data is applicable for hacked servers or data on a local computer. Techniques used to hide the presence of a hacked machine include modifying logs, applications, runtime-libraries or even the operating system itself, preventing incriminating evidence to be returned, and thus making the system appear as if

it had never been compromised [5]. Additional anti-forensic methods conceal the existence of illegal data on a local machine owned by a criminal. Most of these methods covertly hide data in non-portable ways directly on the hard drive. These include the use of slack space between partitions, using the hidden protected area (HPA) sector, the creation of bad sectors on the hard drive and the storage of data on those sectors, or even the creation of second hidden OS [1].

Portable digital documents can also be used to hide the presence of data via encryption, masquerading, or steganography. Encryption shows the existence of data, but if done properly, encryption can prevent the actual data from ever being recovered. Files masquerading as other documents can be achieved as simply as changing the extension of the file (e.g. jpg to txt), or more sophisticated schemes can be employed to place the data in other documents, append it to the end of an executable [9]. Steganography or the hiding of data in some cover medium creates a stego medium that on cursory inspection will appear to have not been tampered with or to have anything notably wrong about the file – despite the hidden data [23]. The locations and ways data is hidden inside other media is called a covert channels and has a defined maximum hiding capacity [18].

The elimination of sources can be simply the fact that data is never recorded – digitally or otherwise. The counterfeiting of evidence is the modification of the files to incriminate someone besides the defendant. Examples of this may include simply the recording of misleading information or the modification of files (e.g., modifying the last modified date, or changing the metadata in a Word document)

1.3 Forensic Computing with OLE2 Documents

Programmatic features of an application can assist in forensic computing. An example may include an application maintaining meta-data on a document or tracking how a document changes. This meta-data or change tracking can contain specific information about the creator of the document, track when and by whom it is updated, and even track the actual changes that were made. A prime example is the use of the tracking feature. Many companies and individuals have been put in the hot seat when the tracked changes were brought to light [11, 31].

Most of the undesired outcomes from the use of this meta-data have been passive in their creation and usually have resulted from a misunderstanding of the features of the applications by users. However, there are other users that aggressively manipulate documents to hide information and use said documents to covertly communicate sensitive information without being detected by authorities. Recently, a new technique for implementing data hiding was developed that finds slack or unused space in the document and then encodes a covert channel of hidden data without any visual indication the file has been tampered with. This technique was developed for the Microsoft OLE2 format, which is used in Microsoft (MS) Word, MS Excel, and MS PowerPoint documents [6].

This new data hiding technique provides a big problem for investigators and employers. Employees or criminals may prevent an investigator or employer from discovering covert plans, shared industrial secrets, or other illegal activity by simply hiding information in a common document format used on most computers. An

employee could send confidential information out of the company and leave no clue of the illegal activity. Although there are no visual clues of data being hidden, there still exist telltale signs that can be used to potentially detect the modified documents.

This thesis a) presents a statistical-based detection tool to find the presence of covert channels in OLE2-formatted documents and b) introduces an improved method of encoding covert channels that lowers the probability of detecting hidden data.

A guide to the rest of this thesis: Chapter 2 reviews related works, Chapters 3, 4, and 5 review the OLE2 format and statistical properties. Chapter 6 focuses on an anti-forensic technique for improving data hiding. Chapter 7 reviews the forensic computing experiments and results of detecting hidden data. Chapters 8 and 9 present future works and the conclusion.

CHAPTER 2

RELATED WORK

Information hiding occurs in various digital mediums: file systems, networking protocols, and digital documents. There is a large body of research and development that exploits the ability to hide information in images [29], audio files [32], executables [8], and movie files [30]. However, comparatively little work has been done regarding hiding data in common office documents.

Cantrell et al. [4] and Byers [3] explored the ability to hide information in MS Office documents and found ways to hide information in the metadata sections of the documents. This method works, but is easily detected. Research in this area points out that large sections of documents are uniformly 0x00 or 0xFF, however; any time those sections are exploited, the document becomes corrupt and cannot be reopened in the native MS Office application. Contrell et al. [4] specifically point out that nearly all types of documents are vulnerable to inserting data past the end of the EOF marker, in which case the documents can still be reopened.

Tsung-Yuan et al. [17] introduce a steganographic method of hiding data in Microsoft documents by using the tracking mechanism available in Microsoft Word. Using a synonym dictionary with the track changes feature, one is able to make the document appear as though it had simply been through several editorial revisions when in reality, the tracked changes are hidden data.

A paper by Castiglione et al. [6] presents *StegOlè,* a method of hiding data in the unused sectors of the Microsoft compound document file format, otherwise known as the

OLE2 Compound Format. This methodology is based on the fact that the document format uses fixed 64- and 512-byte sector sizes, some of which are left in unused sections of the document. Using the file's index or section allocation table (SAT), unused sectors are identified and overwritten without negatively impacting the visual or functional aspect of the document. After the cover document is modified into a stego-document, the file may continue to be the same size and can be opened by the native application. A combination of compression, AES encryption, and an SHA-1 hash are used in combination to maximize space, secrecy, and message verification on extraction

Existing detection tools concentrate on finding hidden information on file systems [10], images [12], video files [28], executables [15], databases [20], and networks [27]. Currently, there are no known techniques for detecting covert channels in common office documents.

At the time of the original literature search for this thesis, December 2006 and January 2007, only a minimal amount of research had been published on creating covert channels in OLE2 formatted documents. In fact, Cantrell et al. [4] and Byers [3] were the only known articles hiding data in MS Office documents. However, the work in [3 and 4] resulted in corrupt files native applications could not open. Shortly after doing reviewing the literature, I started developing of a data-hiding tool for OLE documents, and eventually I successfully implemented a data-hiding scheme. The developed algorithm is able to encrypt and embed, and then extract and decrypt any message into an OLE2-formatted document. The modified document file size is not modified nor is the

look of the document impacted; furthermore the document's integrity is not compromised, allowing the file to be opened by its native application.

In September 2007, I looked for additional research articles and found [6] by Castiglione. The article introduced *StegOlè*, which implements essentially the same algorithm I had developed. There are some minor feature differences between the two implementations, but the core idea of finding and overwriting unused sectors is identical. Even though both methods were developed independently of each other, because *StegOlè* was developed and published first, this work uses it as the reference implementation. Specifically, *StegOlè* adds covert channels for testing my detection tool and provides a base from which to lower the detection rate of my encoding even further.

The work for this thesis goes beyond the work done by *StegOlè*. In the current work, I develop and implement a detection tool for finding any OLE2 document containing a covert channel. The rest of this thesis is organized as follows. After showing how the covert channels are detected, I propose new encoding techniques that do a better job of lowering the detection rate. Chapters 3 through 5 provide a review of the used statistical methods and the OLE2 format. Chapters 6 and 7 discuss the details and results of the detection tool and encoding schemes. Chapter 8 presents future work in this area. Finally, Chapter 9 gives the conclusion.

CHAPTER 3

COVERT CHANNELS IN OLE2-FORMATTED DOCUMENTS

This thesis focuses on the detecting data hidden in unused portions of the OLE2 format and improving that method to decrease the ability to detect the covert channel. The following sections introduce the *StegOlè* algorithm, the stegomedia detection, and my proposed modification to the *StegOlè* algorithm to decrease its detectability.

Microsoft Office Suite uses the Object Linking and Embedding 2 (OLE2) as its Compound Document format. The power of this format is the ability to contain multiple document types within a single file, allowing a single application to embed the contents of different applications. The format is entirely structured around the idea of sectors and streams. An OLE2 document has multiple streams representing the different objects embedded in the document. Each stream, and the file as a whole, is partitioned into 512-byte sectors. In other words, regardless of the actual data size, the file size is always a factor of 512.

A master table identifies each of the streams and the starting sector. To manage which sector belongs to which stream, a summary information stream maintains a block allocation table (BAT). This BAT is essentially an array of 4-byte integers that represent each sector in the document. Often, a block is unused and is identified with a -1; other valid entries maintain a linked list of sectors. Basically, each entry contains the next sector number in the stream. The terminating sector can be identified by the -2 as the value of the entry in the BAT.

Creating covert channels in the OLE2 format takes advantage of properties of this format: the fix sector size and the unused sectors. Often, a particular stream does not terminate on exactly the 512 byte boundary. In such cases, the space between the end of the stream and the sector can safely be overwritten. When a sector is discovered as unused, it is also considered as dead space in the BAT, and that particular sector can be entirely overwritten with no fear of overwriting valid data.

CHAPTER 4

A STATISTICAL APPROACH TO DETECTING COVERT CHANNELS

4.1 Statistical Methods for Identifying Unknown Files

Research has shown how unknown files can be identified using statistical analysis. Using byte frequency analysis and byte frequency correlation combined with file header/trailer techniques, McDaniel et al. achieved a 95.6 percent detection rate [16]. Kolter et al. provide a method of detecting unknown malicious code in executables using machine learning with boosted decision trees. By using 500 n-grams or data points from the byte code, Kolter et al. were able to gain a 98 percent detection rate and a 0.5 percent false positive detection rate [15]. Karresand et al. showed the use of a byte frequency distribution of a sliding and its derivative to be effective in uniquely identifying different types of documents [14]. Erbacher et al. showed that data types contained within a file can be potentially identified using kurtosis, byte averages, standard deviations, and standard deviation averages [9]. The current work builds on previous work using statistical analysis to identify covert channels inserted into OLE2 formatted documents.

4.2 Detecting Covert Channels

The manner in which *StegOlè* hides data and the format of OLE2 documents allows for a unique approach for detection. The covert channel is not actually encoded as part of the existing data; rather, it is hidden in unused portions of the document. This method of hiding data coupled with the fact that OLE2-formatted documents are guaranteed to be split into 512 sections byte allows the use of sliding window statistical

analysis to locate any stegomedia. Based on previous research, at the start of this project, my expectation was that a statistical analysis would be able to correctly identify files modified to hide data. I used two statistical analysis techniques to analyze each window, viz., byte frequency distribution (BFD) [14] and kurtosis [9], both of which have been used successfully in file type identification. I identified and chose these particular measurements based on the work and analysis done by Kerrasand [14] and Erbacher [9]. As is demonstrated in this thesis, choosing these methods was an apt decision.

4.3 Kurtosis

Often used to identify unknown data, kurtosis is a statistical algorithm used to show the peakedness of a set of data. Higher kurtosis values indicate large swings in the data's values, as opposed to more consistent data which have lower kurtosis values. Used by Erbacher in 2007 [9] to efficiently detect file types, it is calculated by multiplying the number of elements in the data set by the sum of the difference between the average value taken to the 4th power. This total is divided by the square of the sum of the difference between the average value and each element take to the 2nd power.

$$K_j = N * \frac{\sum_{i=1}^{N} \left(X_i - \tilde{X}_j \right)^4}{\left(\sum_{i=1}^{N} \left(X_i - \tilde{X}_j \right)^2 \right)^2} \qquad \textbf{(1)}$$

4.4 Byte Frequency Distribution

The second statistic used to detect the existence of convert channels is the byte frequency distribution (BFD). First presented by Karresand et al. in 2006, the BFD is a method of creating a unique numerical representation of a distribution of a set of bytes [14]. This statistic is exactly what is says it is, a distribution of the number of times each byte value is encountered in a set of data. The result is a distribution with 2^8 or 256 entries, the maximum value for a byte. Averaging multiple distributions of the same type of data creates a mean distribution and corresponding standard deviation distribution. The combination of the distribution of mean values and standard deviations define what is called a centroid, or a fingerprint of sorts that is theoretically unique for that particular type of data.

Given a centroid and any particular BFD, just how closely the two are related can be calculated by determining the distance between the two with a quadratic formula. The quadratic formula in essence is the sum of distances between each byte value count, weighted by the standard deviation. Hence, the more consistent the data (lower standard deviation) the more weight that particular value will weigh into the measurement. In more detail, the difference is taken between the window byte count (s_i) and the mean of the Centroid (c_i). The difference is squared and divided by the corresponding standard deviation ($sigma_i$). A smoothing factor (*alpha*) is added to the standard deviation to avoid division by 0 errors. The resulting value is then summed up with the distances for each value in the distribution.

$$\sum_{i=0}^{n-1} \left(s_i - c_i\right)^2 / \left(\sigma_i + \alpha\right) \qquad \textbf{(2)}$$

4.5 Summary

I use these statistical algorithms together on each sliding window to create a unique numerical representation of the data. Called WindowPrints, these representations are similar to fingerprints in that they have multiple points of reference for determining uniqueness. Comparing the distance between the pre-calculated WindowPrints for *StegOlè* data with those from other documents being analyzed determines if the said documents contain covert channels.

CHAPTER 5

STATISTICAL PROPERTIES OF OLE2-FORMATTED DOCUMENTS

5.1 The Data Set

As stated earlier, OLE2 specifies a compound document format that allows a single file to contain many other files, such as images, data, sound, etc. The common office documents created by the Microsoft Office, e.g., Word, Excel, PowerPoint, are implementations of this specification. Because the principals and techniques identified here are applicable across the board to all documents that use the specification, the data used for experiments and analysis here come primarily from MS Word documents. In addition, I collected representative datasets for Excel and PowerPoint documents for a smaller set of experiments.

Each set of documents gathered for these experiments was randomly collected from various websites and varies in size and content. I collected a total of 293 MS Word documents, ranging in size from 20.5 kilobytes to 4,858 kilobytes. The contents of the documents range from primarily text to forms with several tables, while others contain images of various sizes. To determine the impact caused by specific data, two additional Word documents were created and tested separately from the larger dataset, one containing only images and one with only embedded OLE objects. The PowerPoint dataset contains 99 documents ranging in content and size from 26.5 to 31,148 kilobytes. The Excel data set contains 109 files ranging in size from 10.0 to 8,108 kilobytes.

5.2 The Payload

StegOlè requires the hidden message be entered as text from the command line,

because of this limitation all the experimentation done with text data (see Figure 1). The algorithm can easily be modified to encode other types of data, Table 1 shows that after encryption, these other types of data also share very similar kurtosis and BFD properties. The kurtosis value vary from the text value be less the 0.63 percent and the BFD distance only differs by less the 1.45 percent. These results show that as long is encryption is being used, the experiments and results explained are agnostic to the type of data being hidden.

5.3 Experiments with the Kurtosis Statistic

To begin, let us do an overview of the kurtosis values found in an OLE2 document and how they change when the documented are modified by *StegOlè*. Based on the OLE2 specification to use 512-byte blocks, a 512-byte sliding window is applied to both an

```
2   Byte Payload:   ':)'
16  Byte Payload:   '16_This paper is'
32  Byte Payload:   '32_This paper is intended to exp'
64  Byte Payload:   '64_This paper is intended to explain the configuration and usage'
128 Byte Payload:   '128_This paper is intended to explain the configuration and usage
                     of the Log Miner featureThis paper is intended to explain the '
```

Figure 1. Sample of payload data used in experiments

Table 1. Kurtosis and BFD Comparisons with Different Data

	Average Kurtosis	Percent Difference from Text Data	Average BFD Distance	Percent Difference from Text Data
Text Data	1.803	--	468.580	--
Bitmap Data	1.795	**0.45%**	473.912	**1.14%**
JPG Data	1.813	**0.53%**	475.380	**1.45%**
MP3 Data	1.792	**0.63%**	466.849	**0.37%**

unmodified Word document and then the same document after it has been modified by *StegOlè* to contain hidden data. The kurtosis value is calculated for each window and mapped next to each other. When comparing the two side by side, differences in the kurtosis indicate sectors containing hidden data.

Table **2** illustrates this. The highlighted sections wherein the kurtosis differ are those sections that have been modified to create the covert channel, in other words to contain the hidden data.

The highlighted windows, 80, 120, and 129, indicate hidden data. Further analysis shows that if the entire 512 bytes are overwritten, the kurtosis value will be on

Table 2. Kurtosis Values for Original and Modified Document with 512-byte Sliding Window.

Window Count	Original Document	Tampered Document	Difference
77	8.313	8.313	0.000
78	3.138	3.138	0.000
79	8.798	8.798	0.000
80	24.975	1.852	-23.123
81	4.450	4.450	0.000
82	7.885	7.885	0.000
83	0.000	0.000	0.000
….	…	…	…
118	6.281	6.281	0.000
119	26.315	26.315	0.000
120	4.692	2.119	-2.573
121	3.220	3.220	0.000
122	3.722	3.722	0.000
123	4.445	4.445	0.000
124	6.360	6.360	0.000
125	4.496	4.496	0.000
126	6.017	6.017	0.000

127	7.692	7.692	0.000
128	10.197	10.197	0.000
129	11.566	9.527	-2.040
130	2.668	2.668	0.000

average 1.804 with a standard deviation of 0.039. The windows with different kurtosis values do not land within the expected range (e.g. window 129). Instead, they are only partially overwritten with hidden data. This is confirmed when the kurtosis value is calculated with a 64-byte sliding window. The comparison of the unmodified and modified documents illustrate how only 64 bytes of the 512 bytes are overwritten with hidden data. For example, window number 129 in Table 2 has a kurtosis value of 9.527, which is different from the original document but still outside the expected range.

5.4 Experiments with Byte Frequency Properties

The byte frequency distribution is a good mechanism for viewing a profile of the data contained in a document. The real usefulness of the BFD is when centroids [14] are created with specific data. Using the BFD distance calculation, one can determine how closely other sections of data may or may not be related to a given centroid. To showcase what different BFDs look like, I created several centroids from different data, including an entire Word document, *StegOlè*, standard text, and JPG data. Figure 2 gives a profile of an entire OLE2 Word document composed entirely of text and one composed only of images. In the end, the type of data contained in the document does not affect the ability to hide or detect information.

Figure 2 shows the average number of byte values found in the 512-byte sliding windows of a Word document. It also shows a concentration of byte values around the

lower case alphabet ASCII values, with a spike at byte value 30 for the ASCII space character. There is also a smattering of upper case ASCII characters, between 65 and 90.

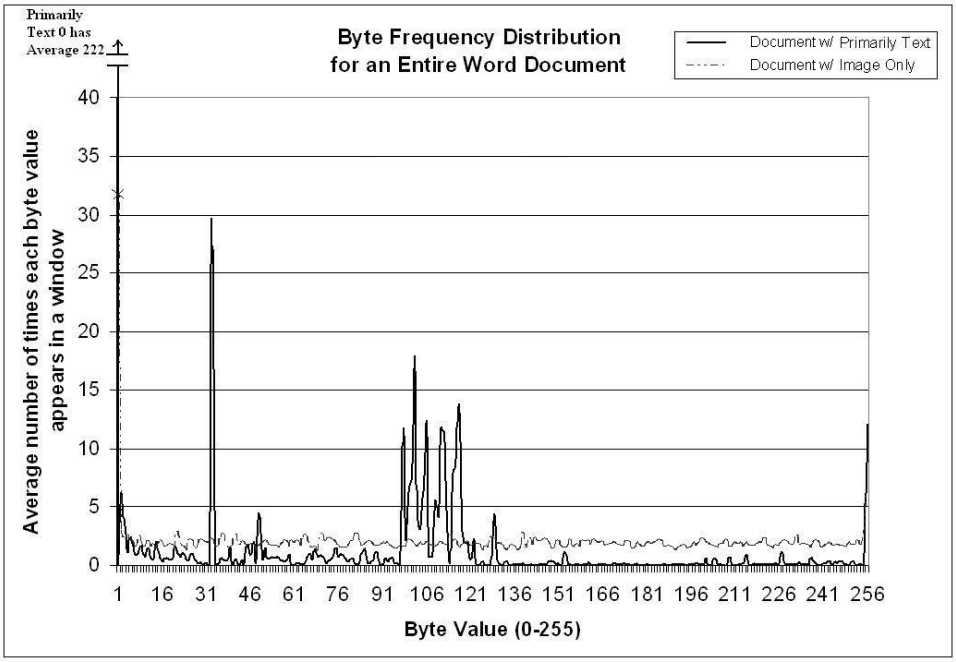

Figure 2. Graph of byte frequency distributions for Word documents.

This is expected for a Word document composed primarily of text. An interesting characteristics of this centroid is the incredibly high count of byte value 0 (average of 222 and a standard deviation of 190) and the generally lower value of all the other byte values. While this is a general view of the byte distribution, it is not consistent through the entire document, showcased by the standard deviation for each byte value, which is 2-4 times higher then the mean, indicating high of variance between the different windows. Figure 3 presents an excerpt of an existing Word file and shows a side by side comparison of the original document and the same document containing a covert channel. Note the high variability of byte values, especially in the unseen used sections of the

document. This excerpt also shows how sometimes the unused section is simply filled with zeros.

When a centroid is calculated for a specific type of data, the BFD values normalize, and lower standard deviations emerge. As shown in Figure 4, the distribution for text appears as expected with a concentration around the alphabet and space ASCII values, while the *StegOlè* data and JPG data are fairly evenly distributed across all values. When mapping the values next to each other, I unexpectedly discovered that the binary distributions for the two are very similar.

To generate the centroid for the covert channels created by *StegOlè* requires several steps. First, because each modified document only has two modified sectors, I modified multiple documents to contain hidden data. Then, I extracted 512-byte window sections from those documents by using the technique to identify the covert channel sections (see Section 5.2). The resulting centroid is subsequently generated by calculating the mean and standard deviation of those harvested windows. See Figure 4 for a visual representation of the *StegOlè* centroid. When the BFD distance between the *StegOlè* Centroid and each sliding window in a Word document is taken, the distances will generally vary from the 400s for similar data to 8.38×10^{10} for dissimilar data. In this thesis, future references to the BFD distance indicate distances between the *StegOlè* centroid and a given data set.

Experiments using 64-byte sliding windows did not yield any interesting data. I found that when windows smaller than 512 bytes are used, there just is not enough data to generate a distribution with any interesting characteristics. This supports the conclusion

made by Kerresand et al. [14] to not use windows smaller then 512 bytes.

5.5 Dead Space Properties of OLE2 Documents

My original research created an algorithm to find within dead space regions of the OLE2 format vulnerable sections that can be overwritten without affecting the document's integrity. The development of this algorithm is based on POI File System (POIFS) [24], an open source, Java-based implementation of the OLE2 format. By leveraging this tool, the Section Allocation Table (SAT) is read and the dead space sectors can be identified. This algorithm developed independently of *StegOlè*, identified nearly all the same dead space sectors that *StegOlè* uses to hide data. These sectors, referred to as dead space, exist in every OLE2 Document that has been examined. Analysis of 293 randomly collected word documents (see Figure 5) show there is an average of 1699.8 bytes (7680 bytes maximum and 368.64 bytes minimum) of dead space per file that can be overwritten without increasing the file size. A significant observation is the lack of correlation between the size of the file and the amount of dead space in a given file.

5.6 Similarity between PowerPoint and Excel Documents

Further analysis of other document types using the OLE2 format, shows similar patterns to those in Word documents. Figure 6 shows a comparison of the average dead space and number of dead space regions in a set of Word, Excel, and PowerPoint documents. The Word data set is as described in Section 5.1, the Excel and PowerPoint datasets are

composed of 103 and 99 randomly collected documents, respectively. Figure 6 also includes sample Word documents containing specific data, one only with images and the other containing only embedded OLE objects. Other than PowerPoint documents which

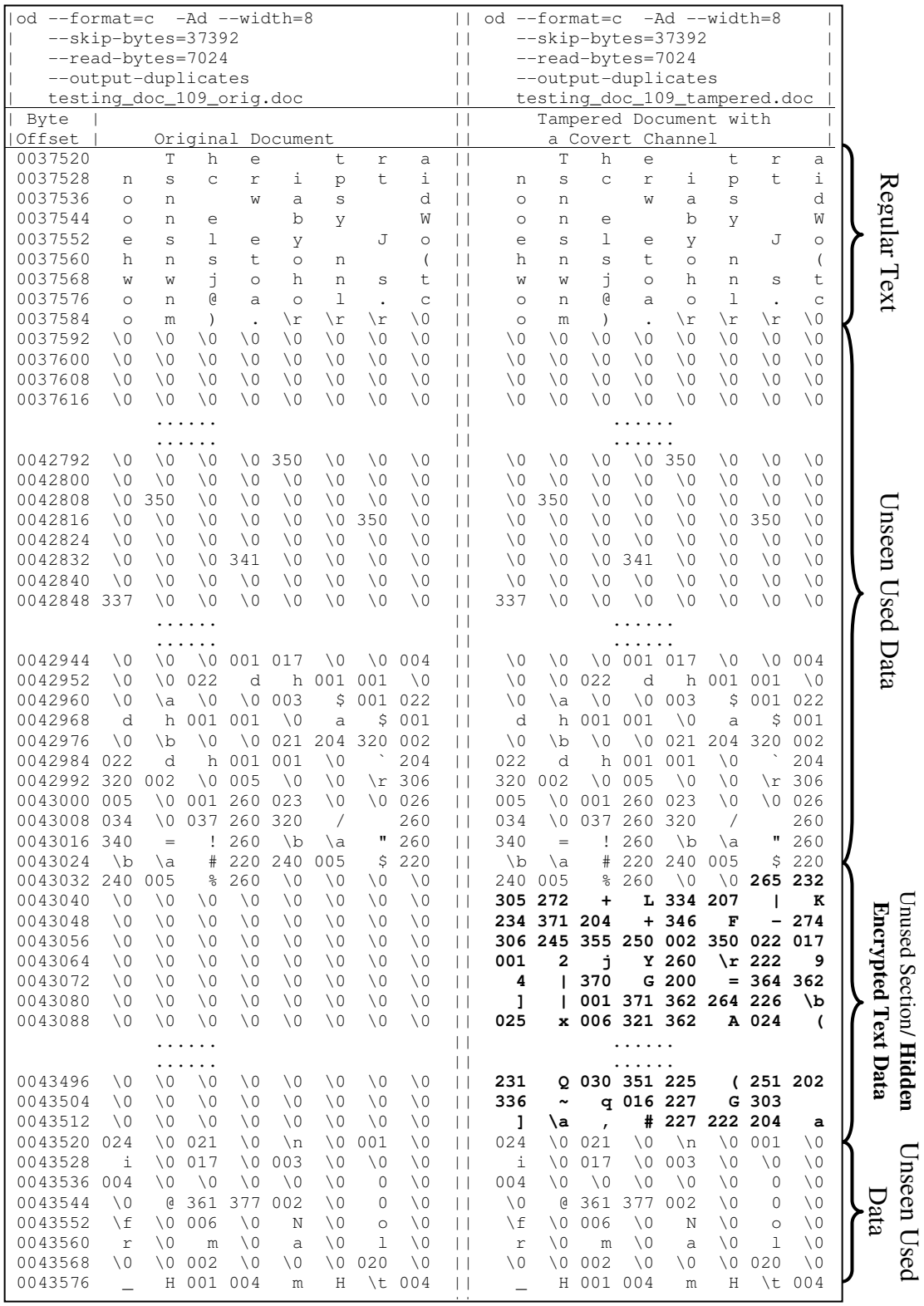

Figure 3. Character data dump from original Word document (left) and altered Word document (right).

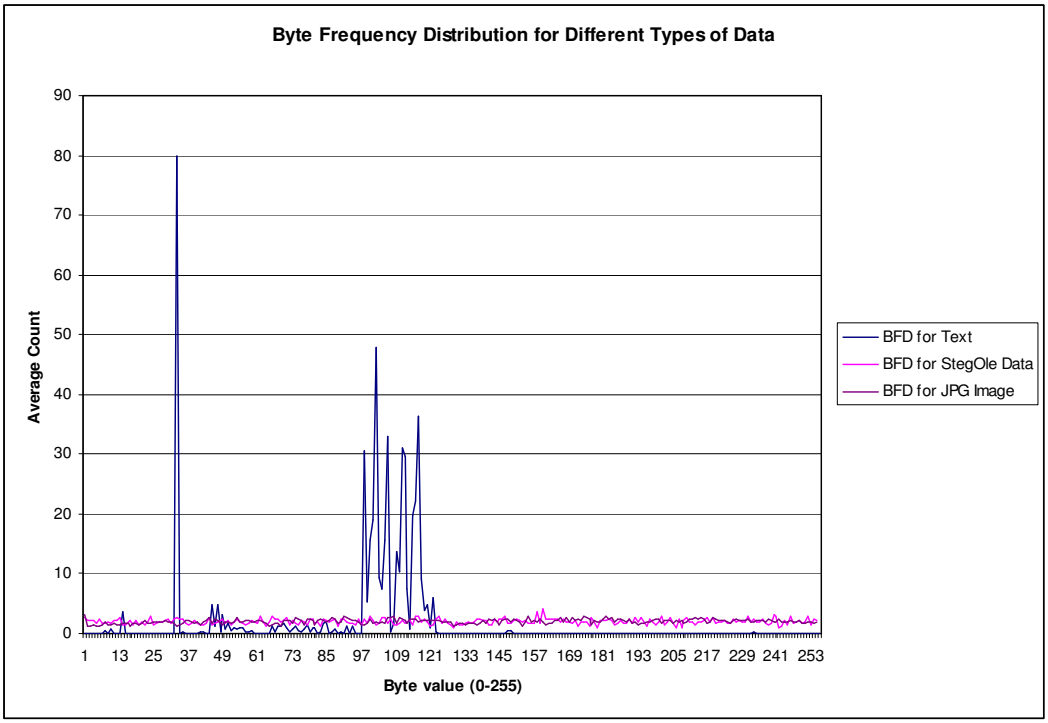

Figure 4. Byte frequency distribution for different text, StegOle covert channel and JPG.

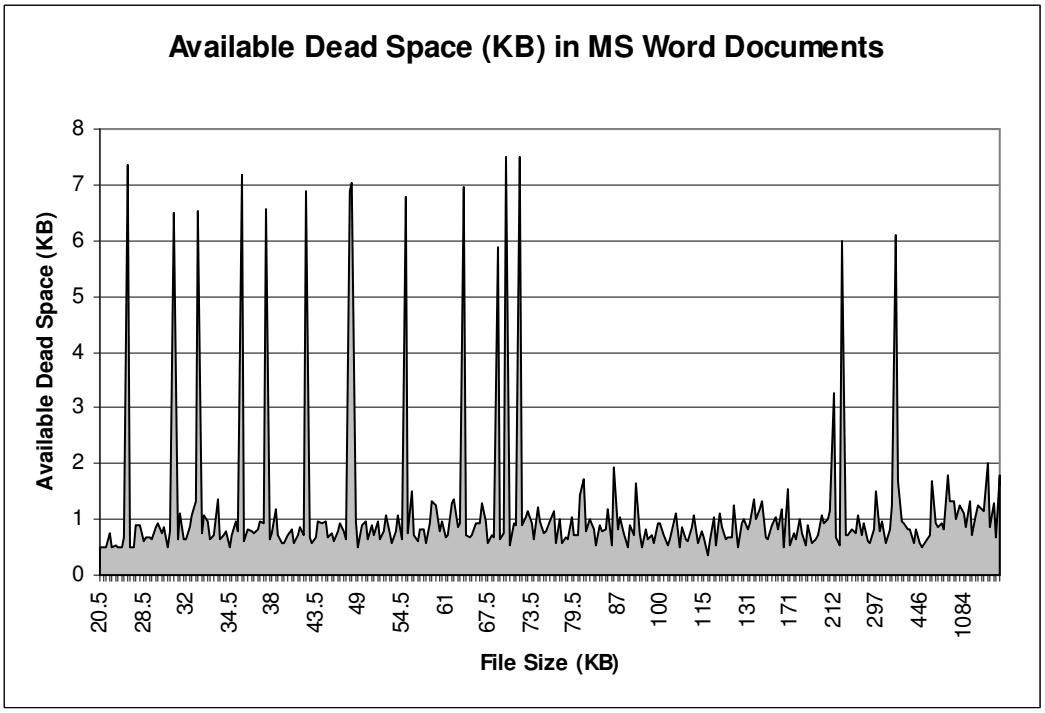

Figure 5. Available dead space (KB) in Word documents.

which generally contain more dead space and more dead space regions than other types of documents, the patterns of available dead space generally match those found in the larger dataset. This additional space found in PowerPoint documents is a bonus, as it increases the potential payload. Unlike a PowerPoint document, even though a Word document with only embedded OLE objects has more dead space and a few more dead space regions than typical Word documents, it still falls within the norm of Word documents. Thus, collecting and enumerating this data demonstrates that each document type is uniquely vulnerable to containing covert channels. While a particular document type or data type may result in a higher capacity for the covert channel, all will be able to contain hidden data that will not affect the behavior or functionality of the document itself.

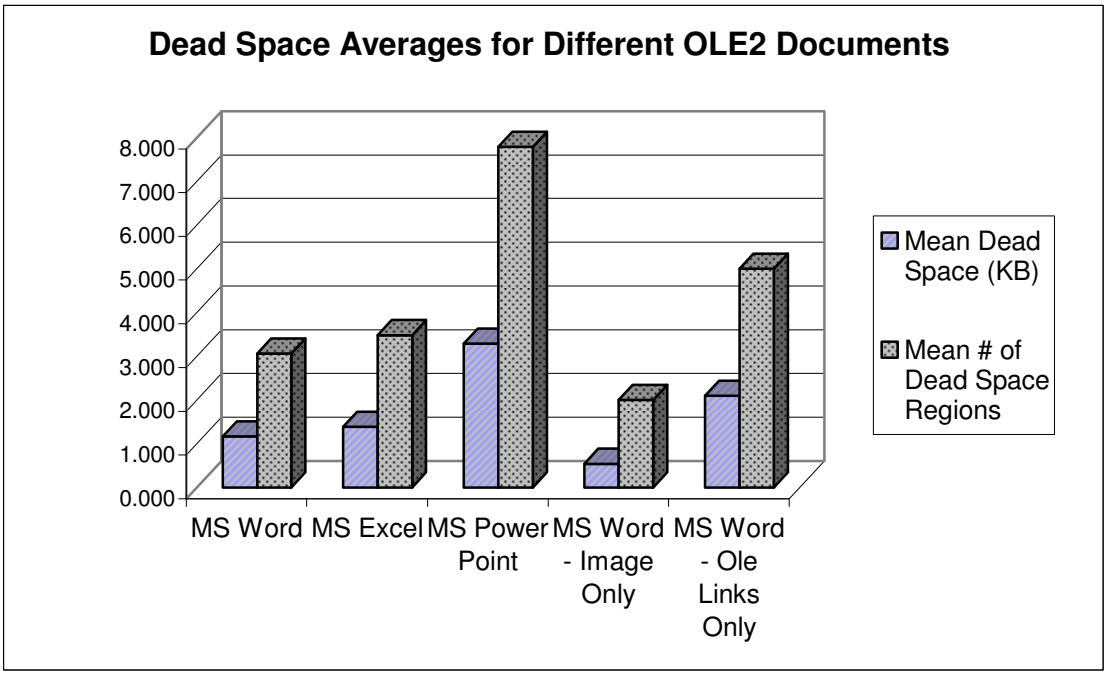

Figure 6. Dead space averages for different OLE2 documents.

CHAPTER 6

A BETTER WAY TO HIDE DATA

6.1 Decreasing the Probability of Finding True Positives

Changing our focus to anti-forensics, in this chapter, I describe a method of decreasing the ability to detect the presence of any covert channels in an OLE2 document. A proficient method to achieve lower true positive detection and increase the false positives is to encode the covert channel to be statistically similar to the existing document. This not only includes picking a single target statistical value and encoding the covert channel to match, but includes variable statistical values similar to those found in the document itself. By matching the statistical values this way, distinguishing between documents that actually contain covert channels and those that do not is extremely difficult. Any threshold defined to detect the differences is inevitably plagued with large numbers of false positives, rendering the detection algorithm useless.

6.2 Exploring OLE2 Vulnerable Regions

Every OLE2 document has dead space regions (see Figure 5) that can be exploited to contain a covert channel. These regions have unique properties different from the rest of the OLE2 document, because the data has no value to the document itself. From the dataset of 293 unique documents described earlier, I extracted and reviewed for common statistical properties the dead space regions. As shown in Table 3, the single most common data points are regions of only zeros, accounting for 39.56

Table 3. General Kurtosis Properties of Dead Space Regions.

General Kurtosis Properties of Dead Space Regions		
	Total Population	Of Unique Values*
Dead Space Regions	365	98
Minimum	1.04	1.11
Maximum	508.94	126.01
Averge	27.22	18.18
Std. Dev.	47.94	16.26
ZERO Count	220	0
Non ZERO Values	144	98
Median	21.69	20.63

* Excludes Outliers

percent of the all the dead space regions. Unfortunately, no data can be hidden in these regions, so the focus turns to those containing any non-zero data. Of the 220 regions, there were 98 that generated unique kurtosis values (excluding outliers). Figure 6 shows the distribution of the 98 different values, the majority of values being between 1 and 10 and between 20 and 30. Ironically, even though there are fewer dead space regions with kurtosis values between 20 to 30 than between 1 to 10, when all the data is considered, the kurtosis values between 20 to 30 account for 16.5 percent of the data points while the 1 to 10 range accounts for a smaller 12.9 percent. Given this information, I deduce that maintaining kurtosis values in the 20 to 30 range will result in a covert channel with the lowest detectability. By focusing on the kurtosis value it is expected and has shown through the experimentation that the BFD distance changes accordingly and blends into the statistical similarities of the document as well.

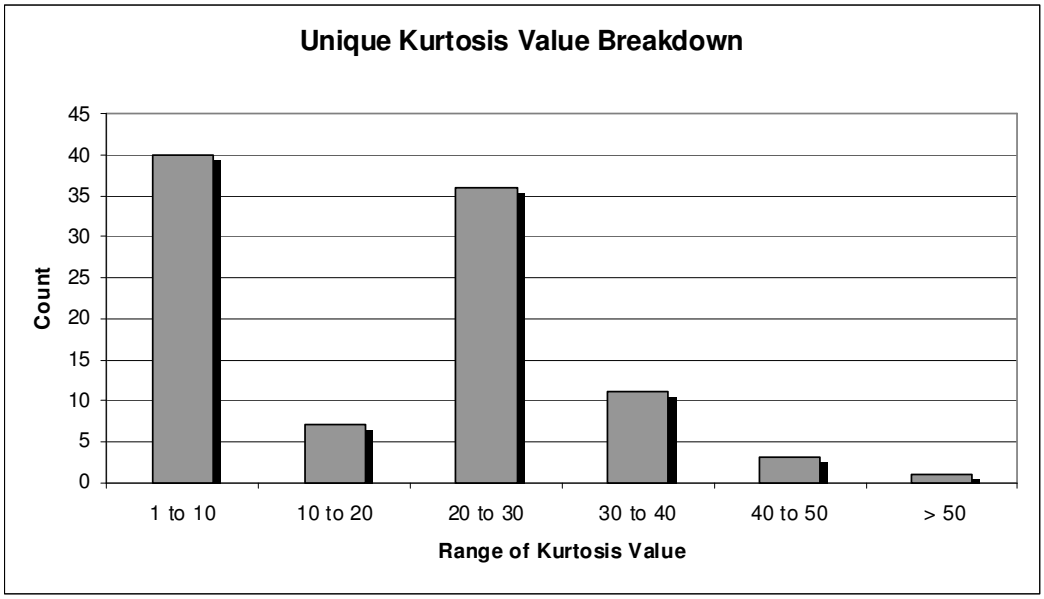

Figure 7. Graph of unique kurtosis value.

6.3 Experiments with Different Encodings of the Payload

The purpose of the experiments was to determine the best combination of payload and filler values needed to blend with the existing documents. I took several approaches. The first was simply encoding the payload with a base 64 encoding, the second attempt was to insert data only every third byte. In additional I tried several more experiments modifying the payload density until the payload was statistically similar to the surrounding data. In these experiments, the measurement used to indicate the payload density was in bytes per 512 bytes region. For example, if the payload is 52, there are 52 bytes of actual data encoded in a 512 byte region, while the rest is filler data, either 0 or 255.

Each experiment explored how the kurtosis and BFD distance thresholds need to change to account for the new encoding and how detection results change.

Table 4 shows the expected thresholds needed to detect each of the different encodings. Each successive encoding attempted to increase the kurtosis and BFD distance, this results in a general trend of the threshold increasing from a kurtosis threshold of 2.2 and BFD distance of 1400 to 38 and 165000, respectively. A different set of documents was then encoded with each type of encoding, and a detection algorithm (see Chapter 7) with each of the different thresholds was run against those documents. The results for each encoding and the detection results are discussed in detail in the following sections. In general, however, the experiments prove that in order to detect the different encodings the threshold must be raised. The drawback is that the higher the thresholds the more false positives are encountered.

Table 4. Detection Thresholds Needed to Detect Each Encoding.

32 Byte Load w/ Zeros	24.23	22.02	25.76	157621	157614	157627	26	160000
22 Byte Load w/ Zeros	34.66	30.51	37.92	164278	163672	164362	38	165000

Experiment Name	Average Kurtosis	Low Kurtosis	High Kurtosis	Average BFD Distance	Low BFD Distance	High BFD Distance	Threshold Kurtosis	BFD Threshold
StegOle (Reference)	1.80	1.75	1.87	406	296	547	**2.2**	**1400**
Base64	4.30	2.99	7.80	2477	716	3601	**8**	**4000**
Every 3rd Byte	4.02	3.56	4.42	68776	2559	80559	**5**	**80000**
52 Byte Load w/ 255	13.31	10.65	15.99	305609	305354	306723	**17**	**350000**

6.3.1 Base 64 Encoding Experiment

The simplest and easiest encoding scheme to test was a base 64 encoding. Using the sun.misc.Base64Encoder, I encoded and hid the payload in an OLE2 document. The resulting analysis showed an increase of the average kurtosis to 4.3 and the average BFD distance to 2477. Immediately these values pushed the covert channel into a range undetectable by the current detection algorithm. Table 5 shows how the detection tool reacted to the increased variability in the new encoding. The reference thresholds of 2.2 and 1400 resulted in almost none of the encoded documents being detected. However, when the threshold was raised to a kurtosis value of 8 and BFD distance of 4000 all the documents were detected. As expected, the number of false positives increased each time the threshold was raised.

Table 5. Detection Results for Base 64 Encoding.

Used Threshold		False Positive Percent	False Negative Percent
Kurtosis Threshold	BFD Threshold		
2.2	1400	0.71%	97.37%
8	4000	0.71%	0.00%
5	80000	19.86%	0.00%
17	350000	33.33%	0.00%
18	145000	32.62%	0.00%
22	155000	46.81%	0.00%
26	160000	64.54%	0.00%
38	165000	89.36%	0.00%

6.3.2 Every Third Byte Experiment

Encoding every third byte with a payload byte was done based on multiple observations of such a pattern. The scheme resulted in a 33 percent density or roughly 168 bytes per region. When the filler bytes were set to zero, the resulting kurtosis average value of 4 was similar to the Base64 Encoding; however, the BFD distance was significantly higher at 80000 (see

Table **6**). The predicted threshold found 95.66 percent of the encoded documents (4.44 percent false negative rate). The false positive percent also increased to 17 percent. Each subsequent threshold resulted in zero false negatives, but the false positive percent increased over the previous experiment.

6.3.3 52-Byte Payload with Filler Bytes Set to 255 Experiment

Another pattern regularly seen is the first part of a dead space region contains varying values, followed by a bunch of zeros. In my next experiment, I used filler values

Table 6. Detection Results for Every 3rd Byte Encoding.

Used Threshold		False Positive Percent	False Negative Percent
Kurtosis Threshold	BFD Threshold		
2.2	1400	0.63%	97.04%
8	4000	1.27%	74.81%
5	80000	17.09%	4.44%
17	350000	31.65%	0.00%
18	145000	31.65%	0.00%
22	155000	43.67%	0.00%
26	160000	61.39%	0.00%
38	165000	90.51%	0.00%

of 255 instead of zeros. This experiment showed an average kurtosis value of 13.31 and an average BFD distance of 305609. This BFD distance was significantly higher than in any other experiment, but the kurtosis value still was not as high as was hoped for. The detection results in

Table 7 show that all the modified documents were discovered with the predicted thresholds. An important observation is how the lower thresholds had high false negative rates (45 to 86 percent).

6.3.4 52-Byte Payload Experiment

Encoding the first 52 bytes of a region with data followed by 450 bytes of 0 closely matched data patterns seen in the data. This payload density of 10 percent had kurtosis values slightly higher than when using a filler of 255 with a value of 14.62, but the BFD distance dropped significantly, down to 143378. With the new threshold values of 18 and 145000,

Table **8** shows all the modified documents were detected. It also makes sense that the previous threshold with a kurtosis of 17 and BFD distance of

Table 7. Detection Results for 52 Byte with 255 Filler Values Encoding.

Used Threshold		False Positive Percent	False Negative Percent
Kurtosis Threshold	BFD Threshold		
2.2	1400	0.68%	86.90%
8	4000	0.68%	81.38%
5	80000	13.51%	45.52%
17	350000	27.03%	0.00%

18	145000	27.03%	0.00%
22	155000	41.89%	0.00%
26	160000	58.78%	0.00%
38	165000	86.49%	0.00%

Table 8. Detection Results for 52 Byte Encoding.

Used Threshold		False Positive Percent	False Negative Percent
Kurtosis Threshold	BFD Threshold		
2.2	1400	0.00%	95.73%
8	4000	0.00%	82.32%
5	80000	18.60%	73.17%
17	350000	31.01%	0.00%
18	145000	30.23%	0.00%
22	155000	41.86%	0.00%
26	160000	54.26%	0.00%
38	165000	86.05%	0.00%

350000 also found all the modified documents. They also have similar false positive percents, because the kurtosis values are similar and BFD distances are so different, it indicates that the kurtosis threshold is the bounding value.

6.3.5 42-Byte Payload Experiment

This experiment was the same as the previous experiment, but I only encoded the first 42 bytes of the region with actual data, with filler values of zero. This resulted in a kurtosis value of 18.88 and a BFD distance of 151058. The predicted threshold in OleDetection with a kurtosis of 22 and BFD distance of 15500, not only correctly detected all the modified documents but also resulted in a jump to 40 percent of the documents falsely identified as containing a covert channel, as shown in

Table **9**.

6.3.6 32-Byte Payload Experiment

A payload of 32 bytes per 512 byte region showed the best combination of kurtosis values and matched the general document statistics. The resulting kurtosis values were in the 20-30 range – the kurtosis value most commonly encountered in the dataset. In combination with the average BFD distance of 157621, the new thresholds in OleDetection were set high enough to detect all the modified documents (see

Table **10**) and incorrectly identify 54.43 percent of the remaining documents as containing a covert

Table 9. Detection Results for 42 Byte Encoding.

Used Threshold		False Positive Percent	False Negative Percent
Threshold Kurtosis	BFD Threshold		
2.2	1400	1.50%	92.50%
8	4000	1.50%	79.38%
5	80000	12.03%	77.50%
17	350000	25.56%	0.63%
18	145000	25.56%	0.00%
22	155000	39.85%	0.00%
26	160000	54.14%	0.00%
38	165000	88.72%	0.00%

Table 10. Detection Results for 32 Byte Encoding.

Used Threshold		False Positive Percent	False Negative Percent
Kurtosis Threshold	BFD Threshold		
2.2	1400	0.63%	90.37%
8	4000	1.27%	74.81%
5	80000	15.19%	74.81%
17	350000	27.22%	4.44%
18	145000	27.22%	0.00%
22	155000	41.77%	0.00%
26	160000	54.43%	0.00%
38	165000	87.34%	0.00%

channel. This combination had a high enough false positive rate and a close enough blend to the existing document's properties to be the most promising approach.

6.3.7 22-Byte Payload Experiment

Dropping the payload to 22 bytes or ~5 percent gave the highest average kurtosis value of 34.66 and an average BFD distance of 163672. Table 11 shows the detection results with the corresponding thresholds of a kurtosis of 28 and BFD distance of 165000 not only found all the modified documents it also incorrectly identified a whopping 89-92 percent (see Tables 5-11) of the documents as containing a covert channel. However, a drawback to this value is the uniqueness found in the kurtosis values above 30. Only 3 percent of the overall dead space regions had kurtosis value from 30 to 40. It should be noted that if this encoding is used, thresholds can be set in such a way as to detect the covert channels and only have a 3 percent false positive error rate.

6.4 Experiment Conclusion

Diluting the payload density is a careful balance between detection and actually being able to provide enough bandwidth as to still be usable. Limiting the payload to 32 bytes per 512 region is probably sufficient to avoid most detection attempts, taken a step further to alter regions between an "every third byte" encoding and a "32-byte payload" encoding would more closely blend in with the overall existing data patterns that already exist in the dead space regions.

Table 11. Detection Results for 22 Byte Encoding.

Used Threshold		False Positive Percent	False Negative Percent
Kurtosis Threshold	BFD Threshold		
2.2	1400	0.68%	91.72%
8	4000	0.68%	80.00%
5	80000	14.86%	80.00%
17	350000	25.00%	12.41%
18	145000	23.65%	1.38%
22	155000	39.86%	0.69%
26	160000	56.76%	0.00%
38	165000	91.89%	0.00%

CHAPTER 7

EXPERIMENTAL RESULTS FOR DETECTING COVERT CHANNELS

7.1 Experiments

The experiments described in this chapter demonstrate the effectiveness of the detection algorithm. The first publication and tool for creating covert channels in the dead space of OLE2 documents was in May 2007, this means the possibility of encountering a modified OLE2 document on the internet is very low. However, given how common OLE2 documents are and the potential for a quick, widespread adoption for malicious use, there a significant need to create a mechanism to detect its.

The experiments described here were run against the dataset of 293 collected documents (see Section 5.1), of which ~50 percent were randomly modified to hide a message using *StegOlè*. Thus, my first task was to collect a set of OLE2 documents and modify a subset to contain a hidden message. As stated previously, success is achieved when a high percentage of the modified documents are correctly identified with a low false positive rate. The developed detection tool uses the kurtosis statistic in combination with the BFD to precisely identify documents that have been modified to contain a covert channel. The following sections describe the experiments that led to the development of this tool.

7.2 Failed First Attempt

The initial intent was to create a general purpose detection tool that considered the document as a whole without taking into account any information the OLE2 format might provide. The approach used a sliding window of 512 bytes and then used the BFD and

kurtosis statistics to identify modified sections of the document. I quickly realized this approach was not feasible because the data patterns generated by StegOlè match those of any other binary data type, such as JPG and BMP images.

Statistics gathered on several different binary formats, including JPG, BMP, MP3, and covert channel data from *StegOlè*, show how the kurtosis and BFD data patterns overlap between the different types of data. Table 12 and 6 followed by Figure 8 and 9 contain detailed data illustrating this pattern overlap. The data was generated from 16 different samples of 512 byte windows randomly taken from files of the given format, the kurtosis and BFD distance are then calculated and summarized.

The JPG and MP3 data shown in Table 12 are a close match to the *StegOlè* hidden data in Table 13. Specifically, the difference between the average kurtosis values of JPG/MP3 data and the *StegOlè* Hidden Data average is less then 0.2. Even though the minimum values are lower then the *StegOlè* min value, the maximum values are very close (within 0.05). See Figure 9. The *StegOlè* BFD distance data has a narrower range, between 296.927 and 547.679, than the JPG or MP3 data, but the min values of the data ranges, 478.076 and 445.507 respectively, do not overlap. See Figure 9.

The bitmap data (BMP) does not overlap as significantly with the *StegOlè* data as with the JPG or MP3, yet there is enough overlap for it to be misidentified. The BMP kurtosis data points on average differ by 0.5. and the overall data ranges do not overlap. The sampling of BFD distances for the BMP here do not show an overlapping either, but they are still close, with roughly only a 7.1% difference between the high *StegOlè* BFD distance and the low value of the BMP distance. Even though this sampling does not

Table 12. Kurtosis Value and BFD Distance Patterns for JPG and MP3 Data.

	JPG (Image) Data		MP3 (Sound) Data	
ID	Kurtosis	BFD Distance	Kurtosis	BFD Distance
0	1.864	557.619	1.701	724.324
1	1.880	751.896	1.819	445.507
2	1.662	859.855	1.868	515.760
3	1.836	592.729	1.838	505.495
4	1.731	608.064	1.733	551.383
5	1.880	616.000	1.782	694.210
6	1.683	874.547	1.702	637.129
7	1.788	569.273	1.849	697.657
8	1.737	524.309	1.817	809.938
9	1.749	583.262	1.835	816.192
10	1.779	524.439	1.765	578.427
11	1.729	478.076	1.816	685.461
12	1.748	604.885	1.771	760.590
13	1.818	487.540	1.800	774.252
14	1.890	508.664	1.874	532.204
15	1.707	632.147	1.812	601.904
	JPG Statistics		**MP3 Statistics**	
Average	**1.780**	**610.832**	**1.799**	**645.652**
Min	**1.662**	**478.076**	**1.701**	**445.507**
Max	**1.890**	**874.547**	**1.874**	**816.192**
Std Dev	**0.071**	**116.147**	**0.052**	**112.776**

Table 13. Kurtosis and BFD Distance Patterns for BMP and *StegOlè*.

	Bitmap (bmp) Image Data		StegOle Hidden Data	
ID	Kurtosis	BFD Distance	Kurtosis	BFD Distance
0	2.331	739.414	1.864	456.889
1	2.545	657.912	1.756	389.996
2	2.331	589.474	1.782	439.945
3	2.225	799.953	1.749	547.679
4	2.339	765.989	1.810	508.936
5	2.362	668.586	1.787	490.766
6	2.171	963.891	1.759	516.013
7	2.388	649.128	1.841	396.611
8	2.392	1111.846	1.872	301.004
9	2.519	629.496	1.809	296.927
10	2.480	721.579	1.814	303.970
11	2.400	656.525	1.794	394.606
12	2.332	789.442	1.872	314.663
13	2.294	649.216	1.757	300.787
14	2.445	727.427	1.794	379.075
15	2.392	672.385	1.798	465.210
	Bitmap Statistics		**StegOle Statistics**	
Average	**2.372**	**737.016**	**1.804**	**406.442**
Min	**2.171**	**589.474**	**1.749**	**296.927**
Max	**2.545**	**1111.846**	**1.872**	**547.679**
Std Dev	**0.095**	**129.955**	**0.039**	**83.346**

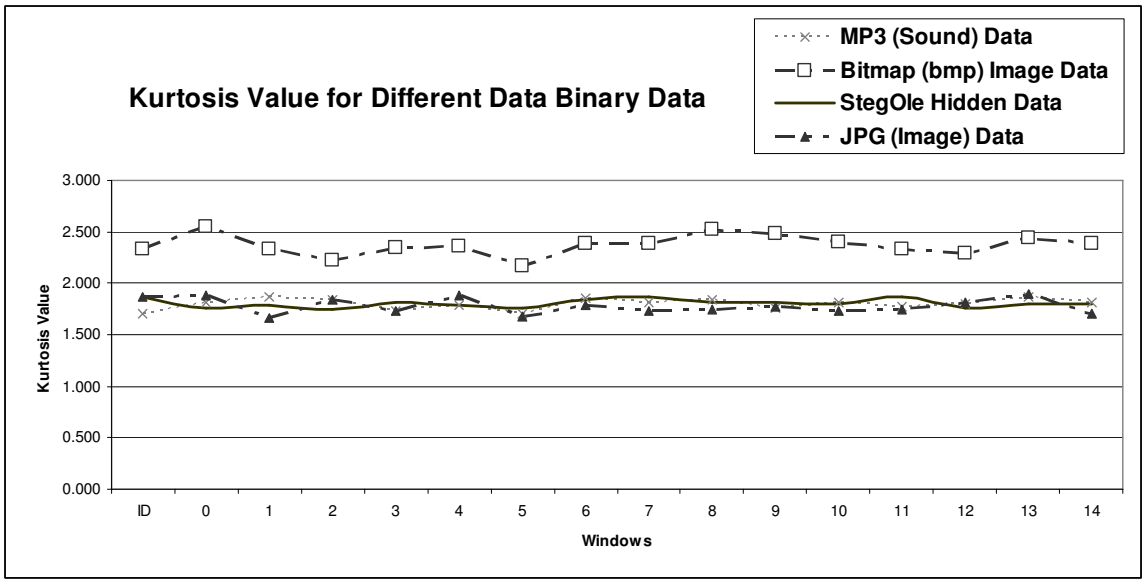

Figure 8. Graph of kurtosis data patterns for MP3, BMP, StegOle, and JPG data.

show an overlap in the data values, they are close enough that when they are put into practice, detecting the difference between the BMP and StegOle data is difficult.

Another important detail is to have the final threshold used for the detection tool incorporate all of these binary types, creating many additional false positives. Establishing a threshold able to distinguish between general binary data and *StegOlè* covert channels requires more than can be provided with the kurtosis and BFD distance individually or in combination. The additional element that enables this detection to work is to use the OLE2 specification to identify the sections of a document that need to be tested.

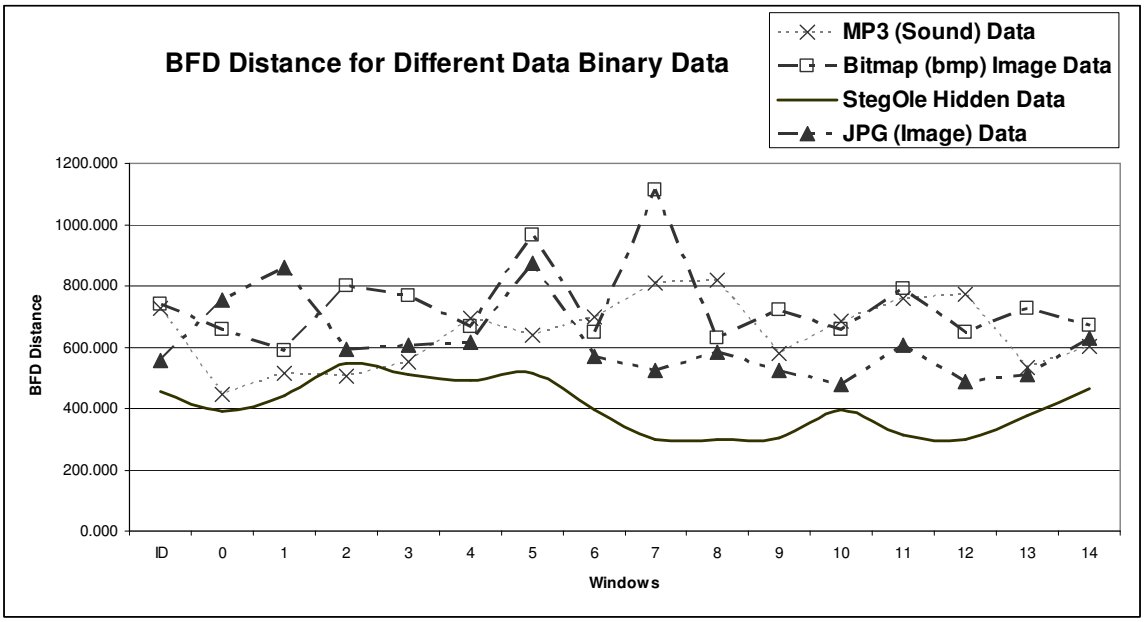

Figure 9. Graph of BFD distance patterns for MP3, BMP, StegOle, and JPG data.

7.3 Detection Tool Experimental Results

The ability to concentrate on specific vulnerable dead space regions allows the use of the kurtosis and BFD statistics to precisely identify documents that have been tampered with to hide data. I developed detection algorithm that given an OLE2 document, the dead space sectors are identified and statistics created. The statistics are then compared against a threshold for the kurtosis and the BFD. If the threshold is met, the document is considered to contain hidden data.

I performed several iterations of experiments to determine the best threshold to use. I tested each threshold against 11 different data sets of 293 Word files or a total of 3223 files, Excel and PowerPoint datasets were also validated against of the final thresholds. I randomly selected roughly 50% of the documents from each data set and modified them to contain a covert channel. Each data set hid a different amount of data in

its covert channel. For example, data set one had 148 randomly picked documents modified by *StegOlè* to hide two bytes of data. I modified each subsequent data set to hide increasingly more data (see Table 14).

7.3.1 Initial Exploration of Threshold Values

Based on the results of the experimentation described in Section 7.2, I targeted the dead space regions and calculated the statistics for each region. The remaining question was the thresholds to use to indicate those documents containing a covert channel.

Given the data in Table 13, the first round of tests used thresholds set to a single standard deviation from the average. The kurtosis value set to 1.81 and the BFD distance set to 490. The results showed on average a 69.2 percent detection rate, well short of the desired mark. I then set the thresholds to the max value encountered in Table 13, a kurtosis value of 1.87 and BFD distance of 547. This showed an improved performance with an average discovery of 73.5 percent but still was not high enough. The following four sections describe my experiments in achieving the desired threshold.

Table 14. Details for Each Test Data Set.

	1	2	3	4	5	6	7	8	9	10	11
Bytes Hidden (Bytes)	2	16	32	64	128	256	512	1024	2048	4096	Assorted
Tampered Files	148	155	148	144	131	151	133	143	137	140	147

7.3.2 Experiment 1

Experiment 1 required that a WindowPrint have a kurtosis value of less than 1.99 and a BFD distance of less than 900 in at least 50 percent of the windows. I selected this threshold based on visual observations made of the modified data. The results were promising, with a majority of true-positive detection rates around 97 percent and no false positives. The surprising exception to this was dataset 8, containing 1024 bytes of hidden data wherein only 88 percent of the documents were detected. This anomaly can be explained by the fact that hiding 1024 bytes of data requires hiding data in two and sometimes three sectors. However, not every sector is completely overwritten; in fact, analysis showed that only one or two of the sectors were modified enough to meet the threshold. This resulted in less then 50 percent of the window prints passing the threshold, thus failing to be identified as containing hidden data.

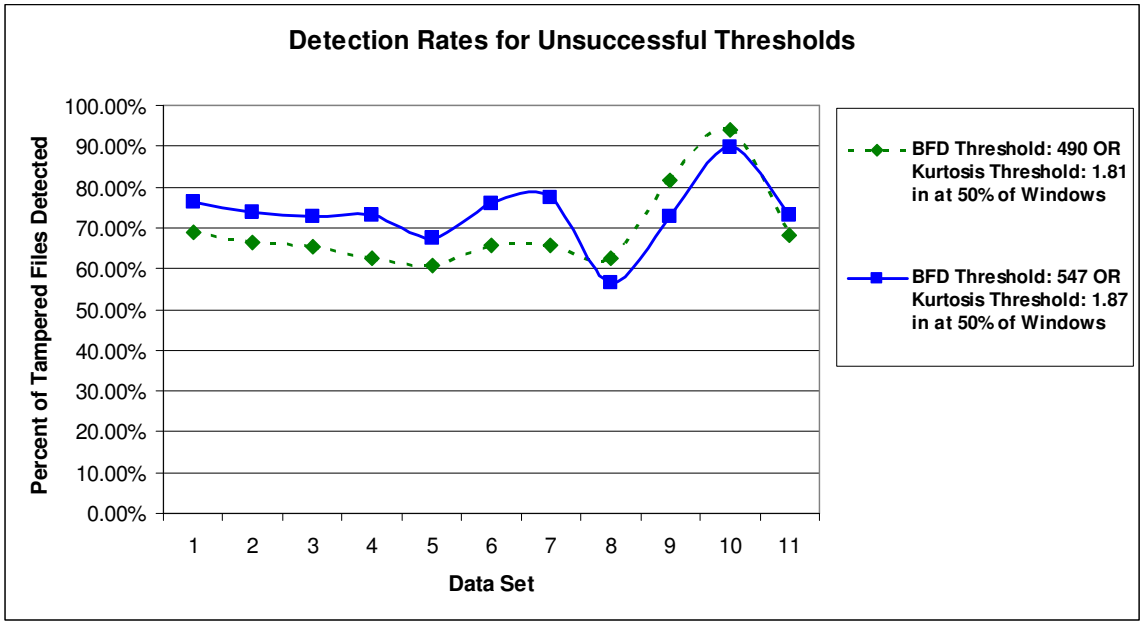

Figure 10. Graph of detection rates for unsuccessful thresholds.

7.3.3 Experiment 2

Experiment 2 was a slight modification of Experiment 1 to account for the observations made. In fact, Experiment 2 varied from Experiment 1 only in the fact that if a single window had kurtosis values of less then 1.99 and BFD distances of less then 900 it would be considered to contain hidden data. This effectively qualified most of the documents that had been missed in the first experiment. The slight change caused the true positive detection rate jumped 99.7 percent with a false positive rate of 0.62 percent.

7.3.4 Experiment 3

Experiment 3 was an attempt to get 100 percent true positive detection while still keeping the false positive rate low. The thresholds were increased to 2.2 for the kurtosis and less than 1400 for the BFD distance. The increased threshold provided the best result with an average of 99.97 percent detection rate, while the false positive rate only increased to 0.65 percent. See Figure 11.

7.3.5 Experiment 4

Experiment 4 kept the same threshold values as Experiment 3,a kurtosis threshold of less then 2.2 and a BFD distance of less then 1400, but it indicated a file as having been modified if any single window print met either of the two thresholds – not necessarily both. This resulted in 100 percent true positive identification but also an unacceptably high false positive rate of 65.18 percent.

This experiment shows how the combination of statistics provides the success of this detection algorithm. Taken individually, they do not provide sufficient data to accurately identify the difference between a modified and nonmodified file. A summary

of the thresholds used for each of the experiments is found in Table 15.

7.3.6 Experimental Validation with Excel and Powerpoint Documents

It is also important to show that the encoding and detection also works for Excel and Powerpoint documents. From each dataset, I randomly selected files and modified them to hide data of assorted sizes. With the threshold from Experiment 3 (kurtosis of 2.2 and BFD of 1400), I executed the detection algorithm against the dataset to identify the modified documents. For the Excel dataset, of the 109 documents, 30 were modified, and OleDetection correctly identified 100 percent of the modified documents while incorrectly identifying five false positives. In the PowerPoint dataset, of the 99 documents, 42 were modified with hidden data. Of those, 38 or 90.5% were correctly identified with a false positive rate of 9.5%.

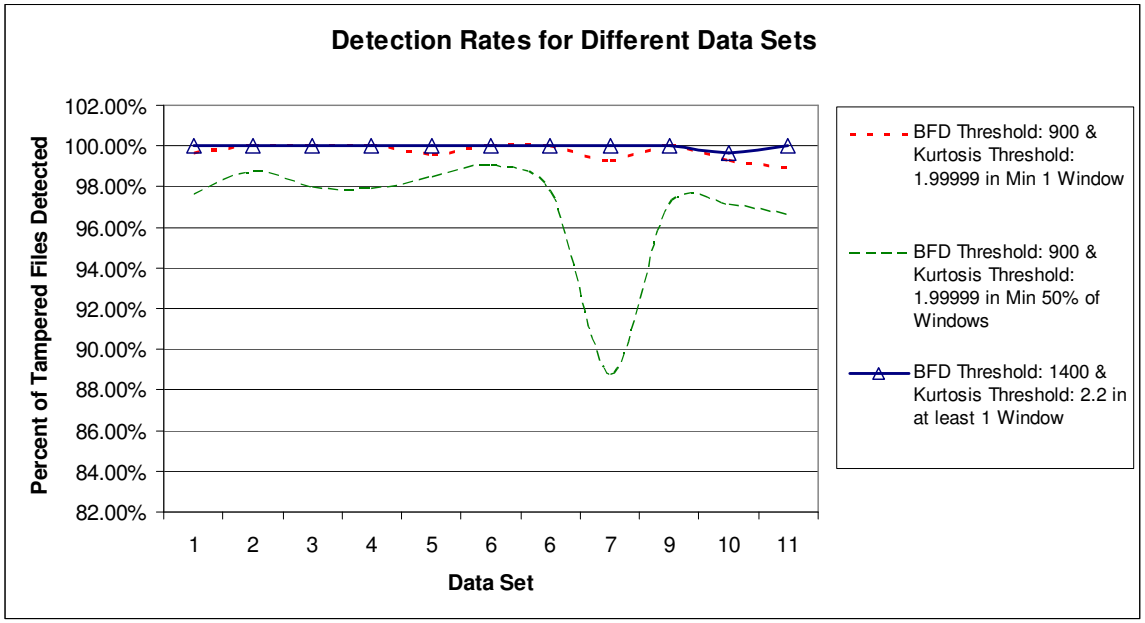

Figure 11. Graph of detection rates for the different datasets.

Table 15. Summary of Experiment Thresholds.

	Kurtosis Threshold	**Predicate**	**BFD Threshold**	**Required Window Count**
Experiment 1	< 1.99	AND	< 900	> 50%
Experiment 3	< 1.99	AND	< 900	At least 1 Window
Experiment 4	< 2.2	AND	< 1400	At least 1 Window
Experiment 5	< 2.2	OR	< 1400	At least 1 Window

The data highlights two points: first, OleDetection work for all OLE2 document types, and, second the threshold could be tweaked for each document type for better results. Yet even so, with the results are not as flawless on these datasets as those for the Word dataset, the detection rate is still respectable and is in the realm of usability. It is not unreasonable to point out that because each document deals with data in a slightly different format, when a section of the document is no longer used, it will carry with it characteristics of that data. The thresholds identified here have been fine tuned for the Word document, the most common of the MS Office documents. With a little tweaking and experimentation, a higher detection rate could be achieved for these types of documents.

7.3.7 Experiment Summary

The value of any detection algorithm is its ability to have a high true positive detection rate and a low false positive count. A point of comparison is the detection algorithm developed by Kolter et al. which was able to achieve detection rates of unknown malicious executables at around 98 percent with a false positive rate of 0.5 percent [15]. Using this as a reference point, Experiment 3 has great results with a 99.97 percent positive detection rate and a 0.65 percent false positive rate. As stated above,

even though the results for the Excel and PowerPoint datasets were a bit lower, a few modifications of the thresholds should result in a higher detection rate.

A different approach is to combine the detection rate and false positive into a single representative number that can indicate the overall value of a particular detection algorithm. This value is a calculated for each dataset by taking the detection rate multiplied by 100 minus 2 times the percent of false positive. For example, if a given threshold resulted in a 95 percent detection rate with a 3 percent false positive, the value for that threshold would be 89. This approach allows for a quantitative approach to evaluating the impact of the false positive detection of the overall algorithm. Figure 12 shows this combined evaluation value for each experimental threshold. Figure 12 highlights an already obvious conclusion, that Experiment 4 is of no real value because of the high false positive detection. This approach helps distinguish between Experiments 2 and 3 wherein the numbers are significantly closer, and the superiority of one is not nearly as obvious. This approach shows that Experiment 3 has the superior threshold with an average value of 98.67. The specific numbers graphed in Figure 12 are shown in Table 16.

7.4 Implementation Details

The OleDetection application is a console-based program able to identify OLE2-formatted documents that have been tampered with to contain hidden data. This implementation allows for the validation of the identified improvements in the algorithm improvements and the accuracy of the numbers. The implementation can examine a directory full of documents or, if needed, a single file. The approach used in

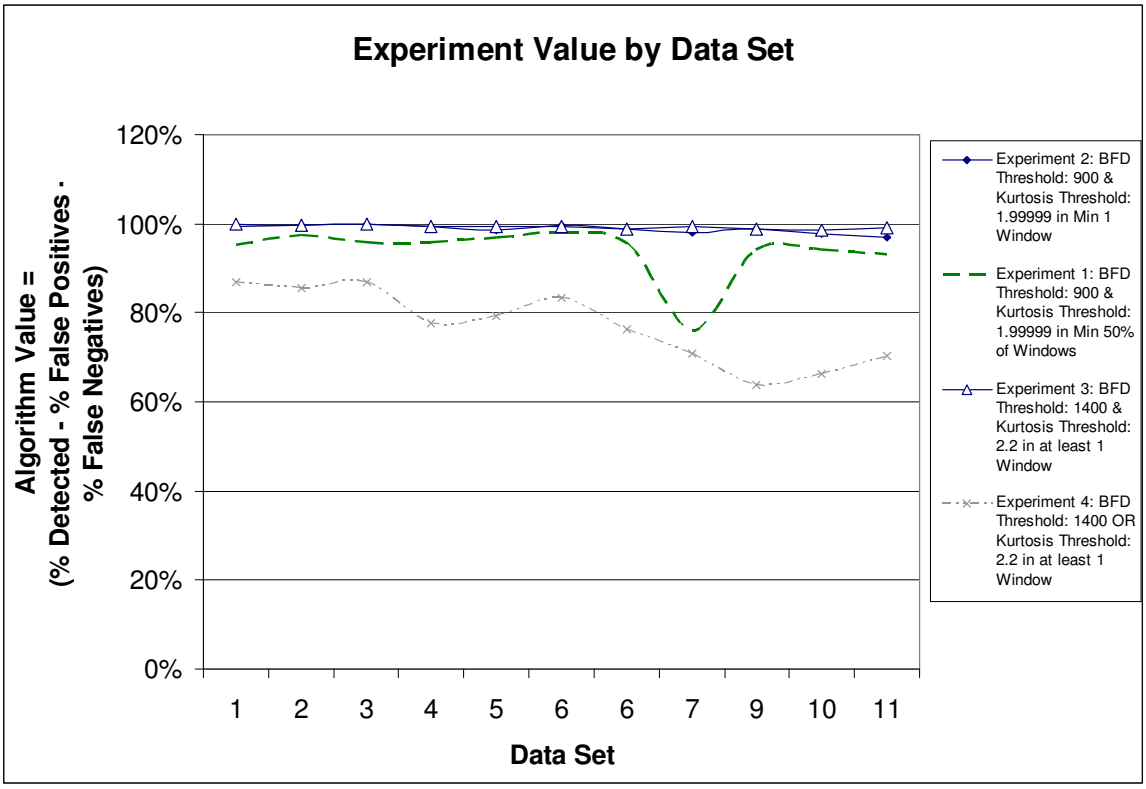

Figure 12. Graph of the relative value for each detection threshold.

Table 16. Average Accuracy of Each Experiment.

	Experiment 1	Experiment 2	**Experiment 3**	Experiment 4
Average True Positive	97.00%	99.71%	**99.97%**	100.00%
Average False Positive	0.00%	0.62%	**0.65%**	22.99%
Average Algorithm Value	93.82	98.81	**99.29**	77.01

OleDetection has a three step process for identifying OLE2 documents with hidden data. Step 1 digests the OLE2 document with OleSteganography and determines the dead space regions. Step 2 extracts the data binary data from those regions and generates the statistics. Finally, Step 3 compares the calculated statistics against the predetermined threshold; if that threshold is met, the document is considered to contain hidden data.

7.4.1 OleSteganography – Finding the Dead Space Regions

Written as part of this research, OleSteganography is a library that is able to find the dead space regions of OLE2-formatted documents. It is built on top of POIFS, an open-source Java implementation of the OLE2 file system specification [25]. To begin, one must remember that OLE2 is a compound file format, or rather a file containing other documents. The development of my method included 1820 lines of code in 17 different classes. The class diagram for this is shown in Figure 14. The ability of a document to contain other data gives the appearance of a file system with containers, leaf data, and indexes to identify the location of data. OleSteganography depends on code that has been inserted into the POIFS code base to identify the dead space regions. When a document is given to OleSteganography, it instantiates a POIFSFileSystem object which, in turn, creates a POIFSDocument. The POIFSDocument is the memory representation of the actual OLE2 document. To determine the contents and the different streams of data contained in the document, a BlockAllocationTableReader which reads the BlockAllocationTable (BAT) starting at offset 0x4C from the start of the document is instantiated. The custom code in the BlockAllocationTableReader detects when a particular block is a DeadSpaceRegion object into a list with the byte offsets to the region not being used (see Figure 13). Once the entire BAT has been read and all the Deadspace regions are identified, OleSteganography is able to retrieve the DeadSpaceRegion objects.

While this code has similarities to what *StegOlè* does, when it identifies convert channels to insert data into, it is not tied to *StegOlè's* implementation. The algorithm

```
//Looping through each sector of the document as identified by the BAT
while(children.hasNext()){
    Object nextChild = children.next();
    if (!(nextChild instanceof DocumentProperty)){
      LOGGER.info("Error...unable to process a child of the Docuement: " + nextChild);
      continue;
    }
    DocumentProperty property = (DocumentProperty) nextChild;
    String name = property.getName();
    try {
      int startBlock = property.getStartBlock();
      potentialDeadSpaceByDocumentName.
                      put(name,batReader.getPotentialDeadSpaceList(startBlock));
      int documentSize = property.getSize();
      int lastByteOfDeadSpace;
      int startByteOfDeadSpace;

      //The BAT indicates if this block is used or not, if it isn't it is
      //susceptible to covert channels
      if (!batReader.isValidBlock(startBlock)){
        startByteOfDeadSpace = startBlock * BLOCK_SIZE + documentSize;
        lastByteOfDeadSpace = (startBlock +1)*BLOCK_SIZE-POIFSConstants.ZERO_OFFSET;
      } else {
        //This grabs any dead space between the end of the data and
        //the end of the block
        int lastBlock = batReader.getLastBlockInChain(startBlock);
        int numberOfBlocks = batReader.getBlocksInChain(startBlock);
        if (numberOfBlocks *BLOCK_SIZE < documentSize){
          System.out.println("Unable to calculate dead space for "
                          + name + ", the report size "
                          + documentSize
                          + " is greater than the # of allocated blocks "
                          + numberOfBlocks);
          continue;
        }
        int lastByteOfDataOffset = (documentSize -
                                              (BLOCK_SIZE * (numberOfBlocks - 1)));
        if (documentSize < (BLOCK_SIZE * (numberOfBlocks - 1))) {
          lastByteOfDataOffset = 0;
        }
        startByteOfDeadSpace = (HEADER_BLOCK_SIZE * BLOCK_SIZE) +
                               (lastBlock * BLOCK_SIZE) +
                                lastByteOfDataOffset;
        lastByteOfDeadSpace = (HEADER_BLOCK_SIZE * BLOCK_SIZE +
                              ((lastBlock + 1) * BLOCK_SIZE)) -
                              POIFSConstants.ZERO_OFFSET;
      }
      //Add the new deadSpaceRange to a list, for later user
      deadSpaceRanges.add(
                     new DeadSpaceRange(startByteOfDeadSpace, lastByteOfDeadSpace));
    } catch(IllegalStateException e){
      System.out.println("Something bad happened for property " + name + " : " + e);
    }
  }
```

Figure 13. Code inserted into POIFS to gather dead space regions.

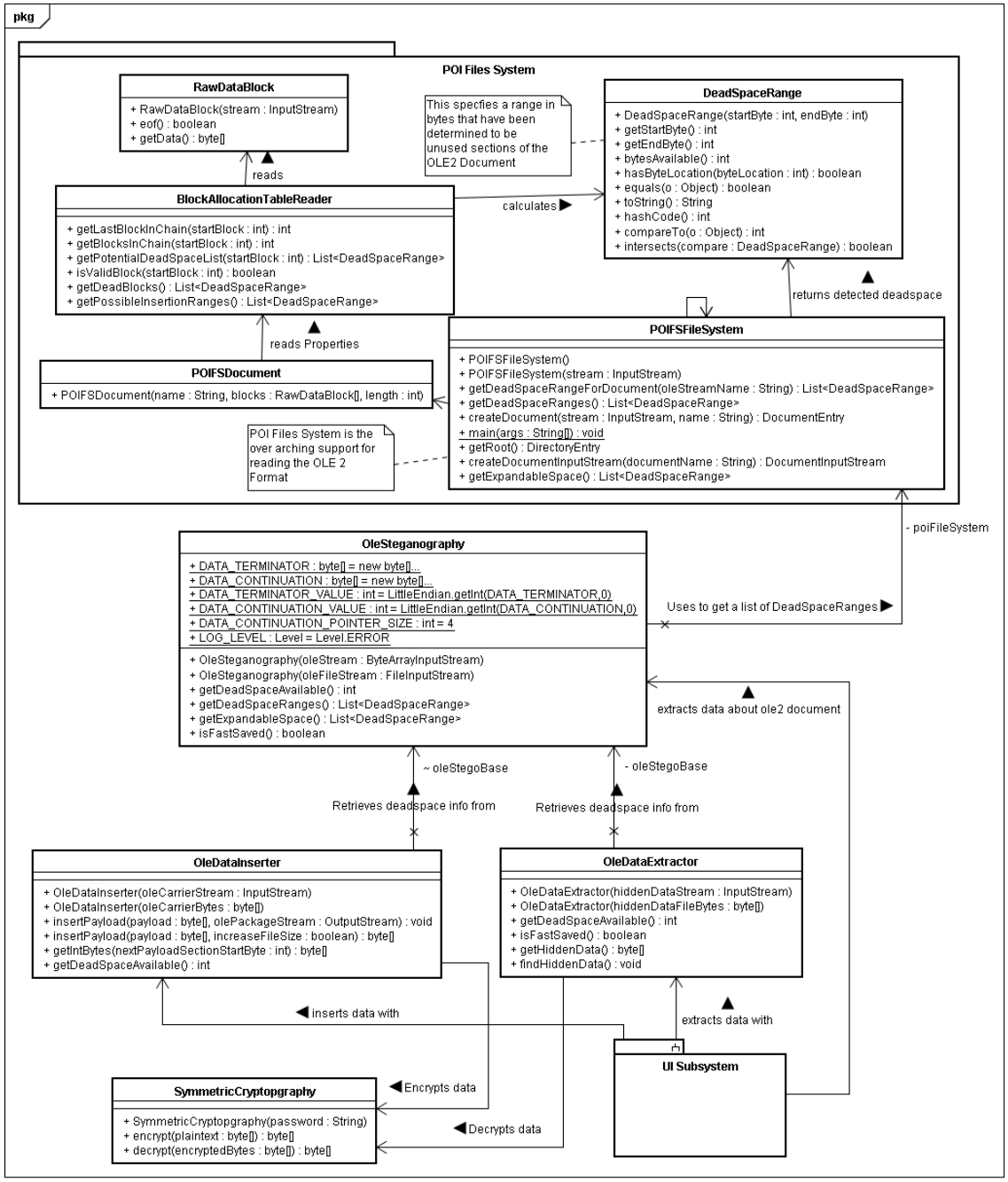

Figure 14. OleSteganography class diagram.

described here is a standard approach to reading OLE2 documents and identifying this type of covert channel. Experiments have shown that the regions found by OleSteganography do not identify every region that has been modified by StegOlè; however, this does not hamper its ability to identify those documents that contain covert channels. The point here is that even if StegOlè changes the location of the covert channel, this process is still be able to identify enough of the regions to detect the hidden data.

7.4.2 OleDectection – Extracting Data and Calculating Statistics

Once the dead space regions of the document are identified, the data needs to be extracted and the statistics calculated. Refer to the class diagram of OleDetection in Figure 15 for a general overview of the classes used and

Figure 16 for the critical piece of code doing the work. The code developed for this portion has about 1500 lines of code in 23 different classes.

To extract the correct regions from the document, OleDetection uses the concept of a WindowIterator. The WindowIterator follows the iterator design pattern, providing an array of integers representing the byte value in the indicated data window for each item in the iteration. The SlidingWindowIterator provides consecutive windows of data from a document; this is used mostly during analysis and testing. The RegionWindowIterator retrieves nonconsecutive windows from a document based on a list of DeadSpaceRegions retrieved from OleSteganography.

A WindowPrint encapsulates all of the data and statistics for a DeadSpaceRegion. The WindowPrint is created with the WindowPrint.Builder and built from the original

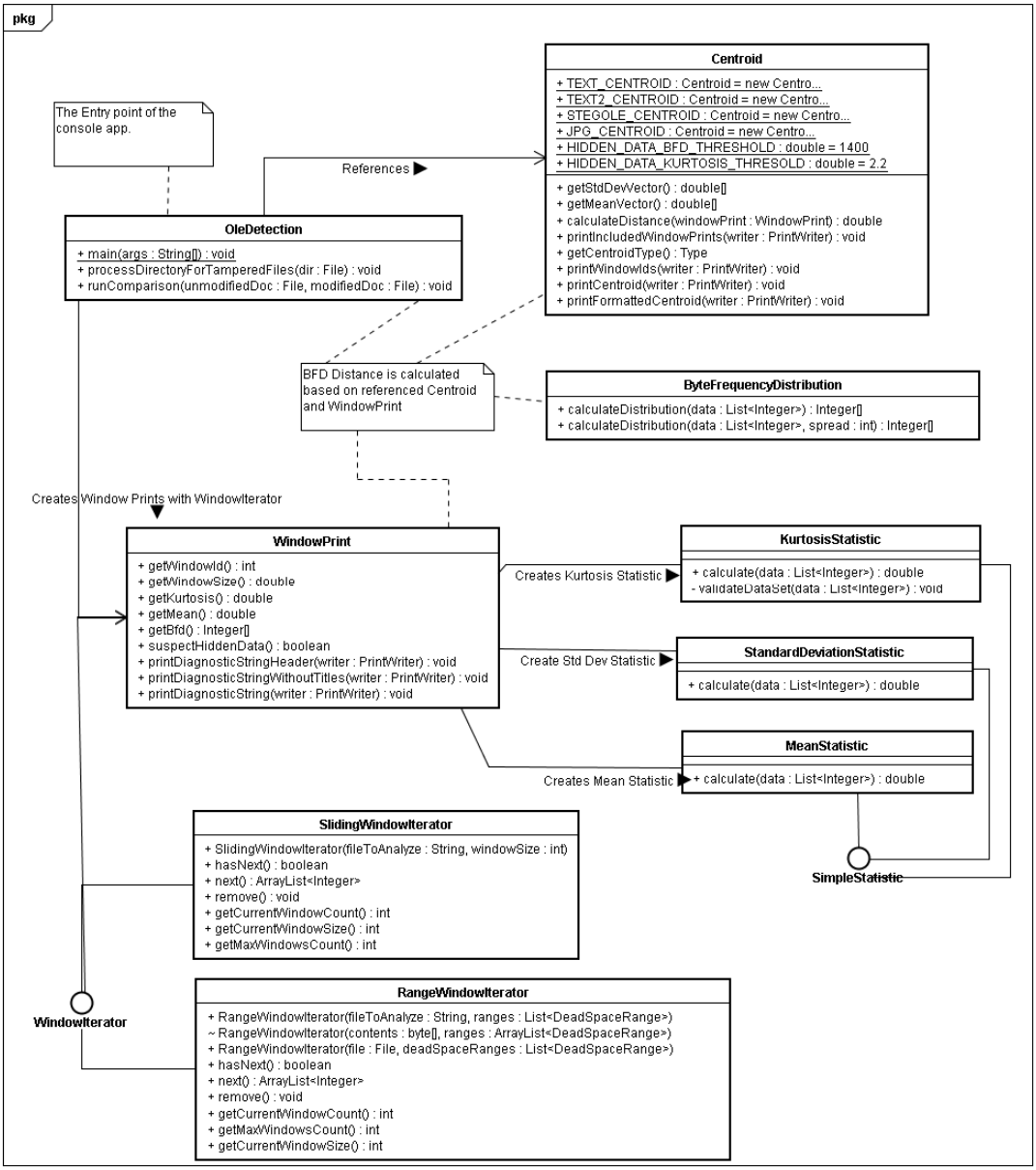

Figure 15. OleDetection class diagram.

```
private boolean presenceOfHiddenDataDetected(File suspectedFile) throws IOException {
  LOGGER.debug("detecting tampered file on : " + suspectedFile);
  if (suspectedFile == null || !suspectedFile.exists()) {
    throw new NullPointerException(
            "Invalid file used to detect presence of hidden data.  \n\tunmodifiedDoc=" +
                    suspectedFile);
  }
  boolean result = false;
  //Find the suceptible regions of the document to analyze
  OleSteganography oleSteganography = new OleSteganography(new FileInputStream(suspectedFile));
  List<DeadSpaceRange> deadSpaceRanges = oleSteganography.getDeadSpaceRanges();

  //Provide easy access to the indicated dead regions using the WindowIterator
  WindowIterator windowIterator = new RangeWindowIterator(suspectedFile, deadSpaceRanges);
  int maxWindows = windowIterator.getMaxWindowsCount();
  ArrayList<WindowPrint> windowPrints = new ArrayList<WindowPrint>(maxWindows);

  //Statistics that will be generated
  ByteFrequencyDistribution byteFrequencyDistribution = new ByteFrequencyDistribution();
  final KurtosisStatistic kurtosisStatistic = new KurtosisStatistic();

  //Loop through each DeadSpace range and generate the Kurtosis and BFD
  while (windowIterator.hasNext()) {
    ArrayList<Integer> currentWindow = windowIterator.next();
    WindowPrint.Builder builder = new WindowPrint.Builder();
    builder.setWindowId(windowIterator.getCurrentWindowCount());
    builder.setWindowSize(windowIterator.getCurrentWindowSize());
    builder.setKurtosis(kurtosisStatistic.calculate(currentWindow));
    builder.setBfd(byteFrequencyDistribution.calculateDistribution(currentWindow,
                                                                   maxDistributionValue));
    WindowPrint curWindowPrint = builder.create();
    windowPrints.add(curWindowPrint);
    LOGGER.debug("Kurtosis Stat for window " + windowIterator.getCurrentWindowCount()
            + " is " + curWindowPrint.getKurtosis());
  }

  //Actually calculate if the particular regions meet the required thresholds
  int suspectedWindowCounts = 0;
  for (WindowPrint each : windowPrints) {
    double bfdDistance = Centroid.STEGOLE_CENTROID.calculateDistance(each);
    if (bfdDistance < Centroid.HIDDEN_DATA_BFD_THRESHOLD &&
            each.getKurtosis() < Centroid.HIDDEN_DATA_KURTOSIS_THRESOLD) {
      suspectedWindowCounts++;
    }
  }
  double percentWindowsSuspected = (double) suspectedWindowCounts / (double) windowPrints.size();
  if (suspectedWindowCounts >= 1) {
    writer.println("Hidden Data Detected:\t " + suspectedFile);
    result = true;
  }
  else {
    writer.println("untampered file     :\t " + suspectedFile);
    result = false;

  }
}
```

Figure 16. Code for detecting presence of covert channels.

integer array, statistics, and meta-data. The centroid class maintains a profile for each type of data, as described in Section 4.4, and calculates the BFD distance between a known centroid and a given WindowPrint. Each statistic is encapsulated in its own object and implements either the SimpleStatistic or DistributionStatistic interface. The KurtosisStatistic is a SimpleStatistic that given an array of integers, calculates the kurtosis value base on Formula 1 (see Section 4.3). The BFD is generated using the ByteFrequencyDistribution base in Formula 2 (see Section 4.4).

7.4.3 Threshold Comparison

The final step in the process is to determine just how close the set of WindowPrints came to meeting the threshold. Each WindowPrint's statistics are compared against the individual statistic threshold. If the set of statistics are determined to meet the threshold, a running counter is increased. The result is a count of WindowPrints representing a covert channel in the document. The final determination as to whether the document as a whole has been modified is to compare this running total against the required number of WindowPrints.

7.4.4 Using OleDetection

OleDetection is a command line application not only able to detect the presence of covert channels, but to analyze and provide data on the documents. Figure 17 shows how to interact with the application for detecting the covert channel for a single file, and Figure 18 is an example of how to find all tampered files in an entire directory.

```
C:\Tools\jdk\bin\java  oledetection.OleDetection -f testDoc.doc
HIDDEN DATA DETECTED:    testDoc.doc
```

Figure 17. Using OleDetection to test an individual file for a covert channel.

```
C:\Tools\jdk\bin\java  oledetection.OleDetection -d ".\Document Repository\\"

Detecting tampered files in: .\Document Repository\2 Bytes
untampered file     :   .\Document Repository\2 Bytes\testing_doc_0.doc
HIDDEN DATA DETECTED:   .\Document Repository\2 Bytes\testing_doc_1.doc
untampered file     :   .\Document Repository\2 Bytes\testing_doc_10.doc
HIDDEN DATA DETECTED:   .\Document Repository\2 Bytes\testing_doc_100.doc
untampered file     :   .\Document Repository\2 Bytes\testing_doc_101.doc
HIDDEN DATA DETECTED:   .\Document Repository\2 Bytes\testing_doc_102.doc
HIDDEN DATA DETECTED:   .\Document Repository\2 Bytes\testing_doc_103.doc
HIDDEN DATA DETECTED:   .\Document Repository\2 Bytes\testing_doc_104.doc
untampered file     :   .\Document Repository\2 Bytes\testing_doc_105.doc
untampered file     :   .\Document Repository\2 Bytes\testing_doc_106.doc
untampered file     :   .\Document Repository\2 Bytes\testing_doc_107.doc
HIDDEN DATA DETECTED:   .\Document Repository\2 Bytes\testing_doc_108.doc
HIDDEN DATA DETECTED:   .\Document Repository\2 Bytes\testing_doc_109.doc
HIDDEN DATA DETECTED:   .\Document Repository\2 Bytes\testing_doc_11.doc
HIDDEN DATA DETECTED:   .\Document Repository\2 Bytes\testing_doc_110.doc
HIDDEN DATA DETECTED:   .\Document Repository\2 Bytes\testing_doc_111.doc
............
............
untampered file     :   .\Document Repository\2 Bytes\testing_doc_93.doc
untampered file     :   .\Document Repository\2 Bytes\testing_doc_94.doc
HIDDEN DATA DETECTED:   .\Document Repository\2 Bytes\testing_doc_95.doc
HIDDEN DATA DETECTED:   .\Document Repository\2 Bytes\testing_doc_96.doc
untampered file     :   .\Document Repository\2 Bytes\testing_doc_97.doc
untampered file     :   .\Document Repository\2 Bytes\testing_doc_98.doc
untampered file     :   .\Document Repository\2 Bytes\testing_doc_99.doc


The list of tampered files are:
testing_doc_1.doc
testing_doc_100.doc
testing_doc_102.doc
testing_doc_103.doc
testing_doc_104.doc
testing_doc_108.doc
testing_doc_109.doc
testing_doc_11.doc
testing_doc_110.doc
testing_doc_111.doc
.....
testing_doc_95.doc
testing_doc_96.doc

. . . . . .
***** Summary *****
Files Examined                :     293
Error Count is                :     0
Detected Tampered file count  :     148
.....................................................................
The undetected files              :
Number of Undetected Files        : 0
.....................................................................
The incorrectly identified as being tampered with files:
Number of false positives         : 0
.....................................................................
Number of Known Tampered Files    : 148
Number of Correctly Dectected Files: 148
Number of Undetected Tampered Files: 0
Number of false positives         : 0
Correct Detection Accuracy        : 1.0
```

Figure 18. Detecting covert channels with OleDetection for an entire directory.

CHAPTER 8

FUTURE WORK

This work shows how covert channels can be detected and hidden in OLE2 documents. This work can be expanded by finding other file types with similar vulnerabilities and applying the principles shown here to those files.

This thesis shows that Kurtosis and BFD have valuable properties that can be used to detect the presence of hidden data, but more work can be done to find other statistical metrics that can be used individually or in combination to detect covert channels or malicious ware. These two statistical methods have shown here work well together. Further investigation can be done to see if other type of viruses and malicious programs can be detected. The thresholds for these statistics can also be modified slightly to more accurately identify covert channels hidden in Excel and PowerPoint documents. Another direction for future work would be adding the detection method to a runtime environment, such as a mail server, to detect and handle documents containing covert channels.

The improved method of encoding the covert channels provides an additional challenge to the forensic computing community. Further research can investigate methods and techniques to accurately identify documents with the improved hiding technique. One possibility is to improve the OleDetection technique to detect this new method of encoding covert channels. Additional methods of encoding the data in ways that increase entropy should also be explored.

CHAPTER 9

CONCLUSION

The anti-forensic computing community is constantly in search of more and better places to hide information. The only way to stay ahead of those looking to thwart legitimate means of storing and transporting data is to provide better methods for detection and understand the extent to which data can be hidden. This work has succeeded on both points. The presented detection methodology is able to detect covert channels inserted into the sample OLE2 documents using *StegOle*. In addition, a new encoding approach has proven to be effective in overturning currently known detection algorithms. It does so by changing its statistical fingerprint to match the surrounding data.

OleDetection is an application able to analyze and detect covert channels embedded into the dead or slack space of OLE2 documents. For readers interested in a detailed approach to this project, the code is included in Appendix A and Appendix B. Using kurtosis and byte frequency distributions (BFD) statistics, with a 99.97 percent average true positive accuracy and with only a .65 percent false positive rate, this application identified those sampled Word documents modified with covert channels. These solid results show how statistics can be used to detect those efforts by the anti-forensic community to covertly hide data.

The thresholds for OleDetection were tweaked toward Word documents. As such, the results for detecting the covert channels in Excel and PowerPoint documents were not quite as good, with an average of 95 percent detection rate between the two and a false positive percentage of ~12 percent. The belief is that these results will improve with a

larger data set and slight modifications to the thresholds to better account for the type of data contained in these additional OLE2 documents.

The new encoding mechanism takes a novel approach to blend covert channels with the surrounding document. Conversely, lowering the density of the payload to match the data being replaced results in the kurtosis, BFD, and potentially other statistical tools to lose some of their potency in discovering covert channels. Specifically, the approach causes so many false positives to occur, that the detection algorithm is rendered useless. Several experiments show that a payload density of 32 bytes per 512 bytes or ~6 percent caused a false positive detection to rise to 27 percent, much too high to be usable in any sort of general application.

The work presented here improves the field of forensics and anti-forensics by showing how statistical tools can be successfully used to detect covert channels and by showing how much more difficult it can be to detect the covert channels if they are encoded correctly.

REFERENCES

[1] Berghel, H. Hiding data, forensics, and anti-forensics. *Commun. ACM 50*, 4 (April 2007), 15-20.

[2] Broucek, V. and Turner, P. Winning the battles, losing the war? Rethinking methodology for forensic computing research*, Springer-Verlag France*, 30 June 2006, http://www.springerlink.com/content/b7588214m68u8386/.

[3] Byers, S. Information leakage caused by hidden data in published documents. *IEEE Security and Privacy 2*, 2 (March/April 2004), 23-27.

[4] Cantrell, G. and Dampier, D. Experiments in hiding data inside the file structure of common office documents: a steganography application. In *Proceedings of the 2004 International Symposium on Information and Communication Technologies*, 2004, 146-151.

[5] Carrier, B. Risks of live digital forensic analysis, *Commun. ACM 40*, 2 (February 2006), 56-61.

[6] Castiglione, A., De Santis, A., and Soriente, C. Taking advantages of a disadvantage: Digital forensics and steganography using document metadata. *J. Systems Software 80*, 5 (2007), 750-764.

[7] Computer Forensics World. Computer forensics basics: Frequently asked questions, April 2008, www.computerforensicsworld.com.

[8] El-Khalil, R. and Keromytis, A. Hydan: Hiding information in program binaries. In *Proceedings of the 6th International Conference on Information and Communications Security,* 2004, 187-199.

[9] Erbacher, R. and Mulholland, J. Identification and localization of data types within large-scale file systems. In *Proceedings of the 2nd International Workshop on Systematic Approaches to Digital Forensic Engineering*, 2007, 55-70.

[10] Fairbanks , K., Lee, C., Xia, Y., and Owen, H. TimeKeeper: A metadata archiving method for honeypot forensics. In *IEEE SMC Information Assurance and Security Workshop*, June 2007, 114-118.

[11] Favro, P. A new frontier in electronic discovery: Preserving and obtaining metadata. *Boston University J. Science and Technol. Law 13*, 1(Winter 2007), 4-5.

[12] Johnson, M. and Farid, H. Exposing digital forgeries through chromatic aberration. In *Proceeding of the 8th Workshop on Multimedia and Security*, 2006, 48-55.

[13] Harris, R. Arriving at an anti-forensics consensus: Examining how to define and control the anti-forensics problem. In *Proceedings of the 6th Annual Digital Forensic Research Workshop*, 2006, 44-49.

[14] Karresand, M. and Shahmehri, N. File type identification of data fragments by their binary structure. In *Proceedings of the IEEE Information Assurance Workshop*, 2006, 140-147.

[15] Kolter, J. and Maloof, M. Learning to detect and classify malicious executables in the wild. *J. Machine Learning Res. 7*, (December 2006), 2721-2744.

[16] McDaniel, M. and Heydari, M. Content-based file type detection algorithms. In *Proceedings of the IEEE 36th Hawaii International Conference on System Sciences,* 2003, 332-331.

[17] Liu , T. and Tsai, W. A New Steganographic method for data hiding in Microsoft Word documents by a change tracking technique. *IEEE Trans. on Information Forensics and Security 7*, 1(March 2007), 24-30.

[18] Moulin, P. and O'Sullivan, J. Information-theoretic analysis of information hiding. *IEEE Trans. Information Theory 49*, 3 (March 2003), 563-593.

[19] Monroe, K., Bair, A., and Smith, J. Digital forensics file carving advances. In *Digital Forensics Research Workshop File Carving Challenge*, October 2006, http://www.korelogic.com/Resources/Projects/dfrws_challenge_2006/DFRWS_2006_File_Carving_Challenge.pdf.

[20] Pavlou, K. and Snodgrass, R. Forensic analysis of database tampering. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 109-120.

[21] Robbins, J. An Explanation of Computer Forensics. http://www.computerforensics.net/forensics.htm

[22] Peron, C. and Legary, M. Digital anti-forensics: Emerging trends in data transformation techniques. In *Proceedings of 2005 E-Crime and Computer Evidence Conference*, 2005, http://www.seccuris.com/documents/papers/Seccuris-Antiforensics.pdf.

[23] Provos, N and Honeyman, P. Hide and seek: An introduction to steganography. *IEEE Security & Privacy 1*, 3 (May 2003), 32-44.

[24] POI - Java API To Access Microsoft Format Files. 2002. http://poi.apache.org/. 2008.

[25] POI - POIFS - Java implementation of the OLE 2 Compound Document format. 2002. http://poi.apache.org/poifs/index.html. 2008.

[26] Voloshynovskiy, S., Herrigel, A., Rytsar, Y., and Pun, T. Stegowall: Blind statistical detection of hidden data. In *Proceedings of the International Society for Optical Engineering*, 2002, 57-68.

[27] Wang, X., Li, Z., Xu, J., Reiter, M., Kil, C., and Choi, J. Packet vaccine: Black-box exploit detection and signature generation. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, 2006, 37-46.

[28] Wang , W. and Farid, H. Exposing digital forgeries in video by detecting double MPEG compression. In *Proceeding of the 8th Workshop on Multimedia and Security*, 2006, 37-47.

[29] Wu, M. and Liu, B. Data hiding in binary image for authentication and annotation. *IEEE Trans. Multimedia 6*, 4 (August 2004), 528-538.

[30] Yang, M. and Bourbakis, N. A high bitrate information hiding algorithm for digital video content under H.264/AVC compression. In *IEEE 48th Midwest Symposium on Circuits and Systems*, 2005, 935-938.

[31] Zall, B. Metadata: Hidden information in Microsoft Word documents and its ethical implications. *The Colorado Lawyer 33*, 10 (October 2004), 53-59.

[32] Zhou, L., Liu, C., Ping, X., Zhang, T., and Wang, Y. Information hiding method in digital audio signal based on perfect codes. In *Proceedings of the International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, 2006, 609-612.

APPENDICES