12-2008

Automated Data Type Identification and Localization Using Statistical Analysis Data Identification

Sarah Jean Moody

AUTOMATED DATA TYPE IDENTIFICATION AND LOCALIZATION

USING STATISTICAL ANALYSIS DATA IDENTIFICATION

by

Sarah J. Moody

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	Stephen J. Allan Committee Member
_____	_____
Chad Mano Committee Member	Byron R. Burnham Dean of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2008

ABSTRACT

Automated Data Type Identification and Localization

Using Statistical Analysis Data Identification

by

Sarah J. Moody, Master of Science

Utah State University, 2008

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

This research presents a new and unique technique called SÁDI, statistical analysis data identification, for identifying the type of data on a digital device and its storage format based on data type, specifically the values of the bytes representing the data being examined. This research incorporates the automation required for specialized data identification tools to be useful and applicable in real-world applications. The SÁDI technique utilizes the byte values of the data stored on a digital storage device in such a way that the accuracy of the technique does not rely solely on the potentially misleading metadata information but rather on the values of the data itself. SÁDI provides the capability to identify what digitally stored data actually represents. The identification of the relevancy of data is often dependent upon the identification of the type of data being examined. Typical file type identification is based upon file extensions or magic keys. These typical techniques fail in many typical forensic analysis scenarios, such as needing to deal with embedded data, as in the case of Microsoft Word files or file fragments. These typical techniques for file identification can also be easily circumvented, and individuals with nefarious purposes often do so.

The results from the development of this technique will greatly enhance the capabilities of legal forensic units, as well as expand the knowledge base in the fields of computer forensics and digital security. The results presented here are promising and certainly do not represent the complete capability of this new technique. They compare favorably with other techniques from recent research and with the capabilities and performance of the professional tools currently in use in real-world forensics situations.

(76 pages)

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In the field of computer forensics, there have been many techniques developed that attempt to aid in the identification of file types. However, none of the techniques currently available have acceptable accuracy in the detection of file types except when relying upon file header information, file extensions, fixed "magic numbers," and other such information associated with the file. The most common method is to use the file extension to identify file type; this method, however, is extremely unreliable, as in most cases the extension is changeable by any user or application. Other commonly used identification schemes can all be maliciously manipulated, and none handle embedded and obfuscated data well.

The goals of this research were to 1) develop an algorithm for data type identification and 2) analyze the effectiveness and accuracy of the developed algorithm. The algorithm is designed to identify and locate date types within a file system and automate this process. As specified by the name, statistical analysis data identification (SÁDI), this method uses various results from a statistical analysis to identify the data type. The most important statistical measurements include: average, kurtosis, distribution of averages, standard deviation, distribution of standard deviations, and the unique characteristics from the distribution of byte values. These particular statistics are the most unique in differentiating among the various data types. No other currently available methods utilize all of these statistics. SÁDI utilizes the byte frequency distribution information differently than other recent identification techniques by focusing only on the unique portions for each type. The most novel addition the SÁDI technique makes is the addition of the other statistics to the analysis process and the utilization of a unique application of a sliding windows technique to allow the identification of fragments of file data.

These novel methods were chosen primarily to obtain greater accuracy in the identification process and to reach the differing goal of data type identification rather than file

type identification. This differing focus is the key to achieving greater accuracy in the analysis. Focusing on data types allows SÁDI to handle situations involving data fragments, embedded and appended data. Future research on SÁDI can expand to evaluate how it handles situations involving cryptography, steganography, and other such data manipulations.

The SÁDI technique involves taking a memory block and identifying the data types within the block, based upon the block's statistical analysis results. This technique also identifies what type of file the memory block is (if it is only one file) based upon the composition of data types and the format of the available data. For example: "Is the data being examined similar to that which would occur in a PDF file, or is it in more of a raw form as would be seen in a doc?' or 'Is the data stored in a binary format?" SÁDI is also unique in that only the content of the memory block is used in the identification of the data types, and it can identify pieces and fragments of files. The memory blocks are divided up into smaller chunks or fragments usually in size of 256, 512, or 1024 bytes. These chunks are then analyzed separately allowing the SÁDI technique to identify file fragments (at the very least 64 bytes, preferably one of the three sizes listed previously). All the data is analyzed in a consistent manner without any user prejudice as to what the data is supposed to mean. Earlier methods focusing on file type identification contained this drawback. Several file types can easily have other data types embedded within them, for example a spreadsheet table embedded within a Word document. In these cases, the identification of the file type is subsumed by the more important issue of identifying the data type. The SÁDI technique focuses on data type identification, and this change in focus is one of the strong points for the proposed technique. This focus is what allows embedded data and data fragments to be identified. The fact that some file types have other data types embedded within them is a major reason earlier methods not based on a content only analysis struggle to accurately identify file types. The data types found within a file are not solely indicative of the file type but also of the component data types.

Chapter 2 is a conference paper from the 2008 Systematic Analysis of Digital Forensics Evidence (SADFE) Workshop being held in May at the Claremont Resort in Oakland, CA. The paper has been accepted and will be published in the proceedings of that workshop. Chapter 3 is a draft of an article that will eventually be submitted to a professional journal in the area of digital forensics. Both works contain detailed information about the implementation and evaluation of the SÁDI technique. Chapter 4 presents conclusions and areas for future work.

CHAPTER 2

SÁDI – STATISTICAL ANALYSIS FOR DATA TYPE IDENTIFICATION[1]

Abstract

A key task in digital forensic analysis is the location of relevant information within the computer system. Identification of the relevancy of data is often dependent upon the identification of the type of data examined. Typical file type identification is based on file extension or magic keys. These typical techniques fail in many typical forensic analysis scenarios, such as needing to deal with embedded data, as in the case of Microsoft Word files or file fragments.

The SÁDI technique applies statistical analysis of the byte values of the data in such a way that the accuracy of the technique does not rely on the potentially misleading metadata information but rather the values of the data itself. SÁDI has the capability to identify what the digitally stored data actually represents and also selectively extracts portions of the data for additional investigation; i.e., in the case of embedded data. Thus, our research provides a more effective data type identification technique that does not fail on file fragments, embedded data types, or with obfuscated data.

2.1 Introduction and Motivation for the Work

In computer forensics, the goal is to locate criminally relevant information on a computer system. Today's operating systems allow such relevant information to be stored in many places within a computer system. The most common place to locate such information is on the hard drive. Given the size of today's hard drives, locating small snippets of criminally relevant data can be extremely cumbersome, especially when sophisticated data hiding paradigms are used. A

[1] Co-authored by Dr. Robert F. Erbacher

digital forensic analyst must be able to locate the evidence, or lack thereof, that might be found on any number of various types of digital storage devices. Rather than simply having to locate files containing criminal activity hidden within the morass of files, analysts must locate the information hidden within otherwise innocuous files. While many techniques can be used to hide information on the hard drive, we focus on the location and identification of relevant information embedded in or appended to other innocuous appearing files. Further, there are many techniques that can be applied for hiding data [17][20].

This need to locate evidence highlights the need to be able to "identify the type of information stored in a device and the format in which it is stored" [14] so the forensic analyst can retrieve the relevant portions of the data and utilize that information in the investigation. This research presents a new and unique technique for identifying the type of data on a digital device and its storage format based upon the values of the stored data bytes and a statistical analysis of those values. While we focus on the identification of data on hard drives, the discussed technique is applicable across the board to all forms of digitally stored data. SÁDI's ability to identify data types as opposed to file types is of critical importance, a definition of file type versus data type is in order. Distinguishing between the two is particularly critical when considering hybrid data types, such as Microsoft Word, which can incorporate text, images, html, spreadsheets, etc.

File type – The overall type of a file. This is often indicated by the application used to create or access the file.

Data type – Indicative of the type of data embedded in a file. Thus, a single file type will often incorporate multiple data types.

Thus, when attempting to locate relevant files, the goal is the location of relevant data types. For instance, when attempting to locate child pornography on a hard drive, we must consider locating the following:

- Image files as separate whole units

- Fragments of image files, i.e., deleted files
- Images or image fragments appended to files
- Images or image fragments embedded into hybrid files, such as Microsoft Word
- Images camouflaged on the hard drive

These scenarios limit the effectiveness of relying on file header information or file extensions that are the primary focus of most detection techniques.

SÁDI takes a block of memory, i.e., a single file, and performs a statistical analysis on it. The file's blocks are processed using a sliding window paradigm [4], and various statistics are calculated for each window and the memory block as a whole. These statistical results are analyzed to identify their relationship to the unique characteristics representative of individual data types. The technique does not rely on the potentially misleading metadata information but rather the values of the data itself.

2.2 Research Problem

There have been many techniques developed that attempt to identify file types; Section 2.3 discusses these techniques. However, currently available techniques do not have acceptable accuracy in the detection of file types except when relying upon file header information, file extensions, fixed magic numbers, and other such information associated with the file. The most common method is to use the file extension to identify file type; this method, however, is extremely unreliable, as in most cases the extension is changeable by any user or application. Many operating systems do not open a file that has been renamed with an incorrect extension, and some virus scanners will not scan files unless they have an executable extension [13].

UNIX systems utilize the *file* command to identify file types. This command utilizes three different methods to identify the parameters passed to it. The first method uses system information to recognize system files, the second utilizes magic numbers found in the first 16 bits

of a file, and the third method utilizes any ASCII content within the file in order to categorize the ASCII data according to language (such as C or a troff input file) [2][13]. For the magic number test to accurately identify the file, the magic number found in the first 16 bits of the file must be found in the */etc/magic* file and be associated with the correct file type as described in [2]. When dealing with file fragments or obfuscated files and data, the reliance on header information prevents magic numbers and file extensions from being useful. Other work on file header analysis, such as that by Li et al. [12], apply enhanced string kernels (ESK) and extended suffix arrays (ESA) to identify header fragments based upon the header content.

Another tool for identifying files is a freeware product called TrID [19]. TrID utilizes the binary structure of files to identify them based upon recurring patterns found within files of the same type. This tool, however, was not designed as a forensic tool and, therefore, does not take into account situations involving covert channels or other such manipulated data purposefully hidden from file type identifying tools. It also has no available documentation on the accuracy or false positive/negative rate. Therefore, although TrID can be useful for many computer users, it cannot be considered a forensic tool, nor does it appear to provide more capabilities concerning file identification than already provided in current forensic tools previously listed.

File header information and other embedded magic numbers can be manipulated to prevent the file from being identifiable by techniques that use this type of information [8]. In addition, the magic number file itself (*/etc/magic*) can be modified by a user to misrepresent files. This manipulation prevents all current techniques from accurately being able to identify file type. Such manipulations are typical of viruses attempting to cloak themselves. Hence, all of the above mentioned methods of identification are easily circumvented.

Another problem with focusing on identifying file type rather than data type is the issue of embedded data. It is very easy for a criminal to hide a table of child porn sales, for example, within a very large word document so that it is not discovered. Current forensics tools lack the

ability to find and locate embedded data, thus causing resources to be spent trying to locate information that may or may not be located on a hard drive [1] [3] [5] [6] [16].

2.3 Background and Related Work

This research extends the work of Erbacher and Mulholland [4]. In this previous work, the technique and an analysis of the potential of the technique were presented. The current research examines the actual implementation of the technique as well as measures its effectiveness and accuracy.

Other previous work in the area of type identification is found in [13]. In this work, three different techniques are used to identify file types. Note, however, that although these detection algorithms utilize file content to perform the identification, the overarching goal of these techniques is to identify the file type regardless of what data or data types might be contained or embedded therein. Consequently, the techniques by McDaniel and Heydari [13] are not able to identify embedded data accurately. McDaniel et al.'s first technique uses a file fingerprint created from the byte frequency distributions of files of the same type. Files are then identified depending upon their match with a type's fingerprint. This algorithm has an average accuracy rate of 27.5%. The second algorithm is the byte frequency cross-correlation algorithm (BCA). This algorithm is similar to the proposed method except it extends the process to look at correlations between byte values. The authors report that it is a much slower method than the BFA method and has an accuracy of just 45.83%. The third algorithm proposed by McDaniel et al. is the file header/trailer algorithm (FHT). This algorithm looks for patterns and correlations in a certain number of bytes at the beginning and ending of a file. Although this method achieves an accuracy of 95.83%, it reduces the problem to a noncontent-based approach that only relies upon headers and trailers; i.e., it fails in many situations of interest to forensic analysts.

In [10], Karresand and Shahmehri present an algorithm that utilizes the measure of the rate of change of the byte contents of a file and extends the byte frequency distribution based on the authors' Oscar method [11]. A centroid, created from byte frequency distributions, byte averages, and the standard deviation for byte values, is used to identify file fragments. To match a type, the file fragment has to be closest to the type's centroid. In [10], this Oscar method is then extended to incorporate the ordering of the byte values in a fragment using the rate of change between consecutive byte values. This Oscar method achieved 92.1% detection rate with a 20.6% false positive rate for JPEG files. Zip files achieved a detection rate of 46% to 80% with a false positive rate of 11% to 37%, while exe files only achieved a detection rate of 12.6% and a false positive rate of about 1.9%. For both zip files and exe files, as the detection rate increased so did the false positive rate. The fact that the technique was effective at identifying jpeg files is misleading. The authors incorporated analysis steps into their algorithm designed to specifically detect JPEG files; i.e., they look for byte patterns required to appear in JPEG files. This is the reason for the high false positive rate with JPEG files, and it brings into question the overall usefulness of the technique. These Oscar-based methods focus on identifying file type and, hence, have the same drawbacks as found in the methods developed by McDaniel et al.

Hall and Davis [7] present an entropy-based method for performing broad file type classification. The technique uses an entropy measurement and a compressibility measurement through the application of a sliding window technique. The technique fails to identify file types accurately, but it aids in differentiating the type of data contained within the file, such as compressed versus uncompressed data.

Stolfo et al. [18] show the results of inserting common malware into doc or pdf files. The authors inserted malware at the beginning, at the tail or in the middle. A common virus scanner was then used to see if the malware could be detected. Except those cases involving the CodeRed worm, the malware was not detectable. But, by the same token, the malware also affected the

ability to open the file without an error occurring. This work did not attempt to develop any new techniques for identifying malware. SÁDI, on the other hand, has the capability to identify embedded data. Future work will incorporate these container types and allow for the identification of executable data embedded within other data types.

2.4 Research Project Goals, Approach, and Uniqueness

Given the weaknesses of the existing tools for the identification of file types, we set out to develop a new methodology for file type identification that built on and improved previous work. Our goal with the development of the new methodology was to:

- More accurately identify of file and data types
- Identify obfuscated data and covert channels
- Locate and extract hidden data

While a full range of statistical techniques were examined for SÁDI, the following were identified as the most relevant for the data type differentiation: average, kurtosis, distribution of averages, standard deviation, distribution of standard deviations, and byte distribution.

The graph of averages shows how the range of values in each window changes across the file. The kurtosis is used to show peakedness in a dataset and, hence, identifies the flatness or consistency of the data directly. The kurtosis is essentially another measure of the consistency of the data. The standard deviation essentially identifies how chaotic values within a window are and how tightly fit the elements are to the median; i.e., are there many outliers in the window or are the values mostly consistent? The distribution of averages and the distribution of standard deviations are both alternative ways of viewing the average and the standard deviation. The byte distribution allows us to differentiate between very similar data types, such as in the case of html, txt, and csv data. All of these data types are strictly textual data, but each has unique

characteristics in their distributions that allows for differentiation among them through the distribution of byte values.

The most novel addition the SÁDI technique makes is the addition of the other statistics to the analysis process and the utilization of a sliding windows technique to allow the identification of fragments of data. These novel methods were chosen primarily to obtain greater accuracy in the identification process and to reach the goal of data type identification rather than file type identification.

2.5 Methodology

2.5.1 Applied Statistical Techniques

As demonstrated by Erbacher and Mulholland [4], the most useful statistics for data type identification (regarding currently studied data types) include average, distribution of averages, standard deviation, distribution of the standard deviations, and kurtosis; to which we added the distribution of byte values. More specifically:

Average – the average is taken by averaging the byte values for each window *i* and averaging the set of window averages. N denotes the number of bytes in the window. The graph of averages shows how the range of values in each window changes across the file.

$$\tilde{X}_j = \frac{1}{N}\sum_{i=1}^{N} X_i$$

Distribution of Averages – the probability that an average chosen from all the averages of a memory block is of value B in the range of 0-255. The goal of mapping the distribution of the statistics, i.e., measuring the probability of a statistical value occurring, is to provide a summary of the type of data in a file, providing an overview of the components of a file.

$$D_{\tilde{X}_B} = \Pr((B+1) > \tilde{X}_j \geq B)$$

Standard Deviation –the standard deviation of the byte values of a window from the average for the window. This essentially identifies how chaotic elements' values within a window are and how tightly knit the elements are to the median; i.e., are there many outliers in the window or are the values mostly consistent?

$$S_j = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(X_i - \tilde{X}_j\right)^2}$$

Distribution of Standard Deviations – the probability that a standard deviation chosen from all the standard deviations of a file is the value B.

$$D_{S_B} = \Pr((B+1) > S_j \geq B)$$

Kurtosis – the peakedness or consistency of the data calculated from two different modifications of the standard deviation. The numerator is the standard deviation squared with a fourth power instead of a square power, and the denominator is the standard deviation squared.

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

Distribution of Byte Values – the probability that a byte chosen from all the bytes in a window is the value of B. Only unique values are used in the analysis.

$$D_{X_i} = \Pr((B+1) > X_i \geq B)$$

These statistical characteristics are then utilized in the algorithmic analysis of the digital data to uniquely identify data of each data type. In various cases, the other statistics mentioned in [4] can be used to increase accuracy and differentiate between very similar data types.

2.5.2 Identifying Unique Data Type Characteristics

As a starting point, only *base* data types were studied and applied to whole files in order to verify the accuracy and effectiveness of the statistical techniques and identified range characteristics associated with each data type. Base data types are those for which the entire file can be considered to be of the same data type (ignoring header information); some examples include jpg, exe, dll, and txt. The range characteristics amount to identifying the expected window values for each statistical technique. Unique range values for a sufficient number of statistics were used for each data type to ensure differential diagnosis. Creation of these range values was done through two phases. First, the statistical graphs for a large number of files with a similar type were compared. The goal here was to examine the graphs to identify unique and consistent patterns that may aid differential diagnosis. Second, the actual numerical values of the statistical techniques were examined to identify the exact values that gave rise to the unique characteristics identified in the graphs. These statistical differentiators were then stored in a configuration file in association with each of the individual data types. The identified range characteristics are shown in Table 1. These ranges apply to a 256-byte window.

Due to the shifts in the statistical results for the different window sizes, this textual input file used initially in this research project is only applicable for the window size used to generate the characteristic data found therein. Hence, for varying window sizes multiple characteristic input files are required. The advantage of allowing for different window sizes comes from the fact that each window size produces slightly varied statistical results for the data types, thus may highlight differing unique characteristics that may not be as clearly visible at alternate window sizes. We primarily use a window size of 256 bytes. This size was chosen because it is small enough to not obfuscate the data, a common problem with too large a window, yet it is large enough to provide a good amount of data in each window to produce unique characteristic statistical information.

Table 1. Configuration Values for the Differentiation of Typical Data Types.

Type	Average		Kurtosis		Std. Dev.	
	Min.	Max.	Min.	Max.	Min.	Max.
NULL	0	0	0	0	0	0
Txt	58.156	97.640	1.013	5.065	26.482	38.07
Html	40.40	97.70	1.40	4.10	22.00	38.40
Csv	44.0	100.0	1.40	23.40	4.50	38.07
Jpg	103.0	148.40	1.48	3.30	56.70	88.80
Dll	0.0	178.0	0.0	20.80	20.0	106.70
Xls	2.0	87.690	1.265	51.993	5.025	93.735
Exe	1.510	165.86	0.0	58.620	0.0	118.58
Bmp	0.0	255.0	0.0	109.0	0.0	120.0

Several data types cannot be differentiated due to their extremely similar binary structure. Data types currently being analyzed that cannot be differentiated include:

- Exe and dll files – These file formats are both binary with similar statistical values and compiled code content.
- Csv, html, txt – Since these are all text oriented file formats, they have similar narrow ranges of values. The goal for differentiation is the identification of unique probability characteristics, namely, a high appearance of "," in csv and {"<",">"} in html.

Some data types also have unique identifying patterns for some statistics. This is seen, for instance, with Microsoft Excel spreadsheet files in conjunction with byte averages. This particular data type and statistic has a stair-step pattern unique to average statistical results for spreadsheet data [4]; see Figure 1.

Also of note are the peaks at the beginning and end of the data streams. Finally, an embedded graph is visible at the 38% mark for one of the data streams. This unusual pattern in

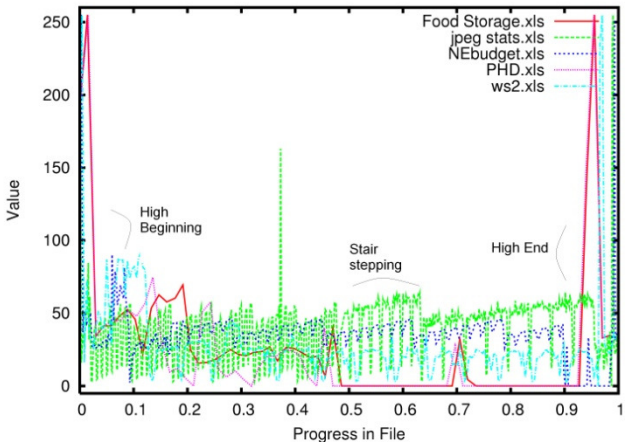

Figure 1. Average values for xls data.

the xls average statistical data engendered a need for a second pass within the analysis code. The first pass focused solely on applying statistical analysis techniques and attempts to identify data blocks that match the statistical structure of known data types. The second pass performed an analysis that identified unusual patterns in these computed statistics, as seen with Microsoft Excel spreadsheet files; see Figure 2.

The figure also shows the whole structure of the xls file; the beginning windows match an xls file header followed by windows that match spreadsheet data. These are then followed by windows matching empty spreadsheets and finally the last few windows match an xls trailer, clearly showing the data components found within an xls file.

Output is provided to the user designating the most likely data types for each file. Any embedded or appended data should be able to be identified on a per window basis, similar to that shown for the component data types in the xls file shown in Figure 2. The characteristic pattern typical of xls spreadsheets can be clearly seen. This pattern remains identifiable even with larger numbers of conflicting matches in the first pass.

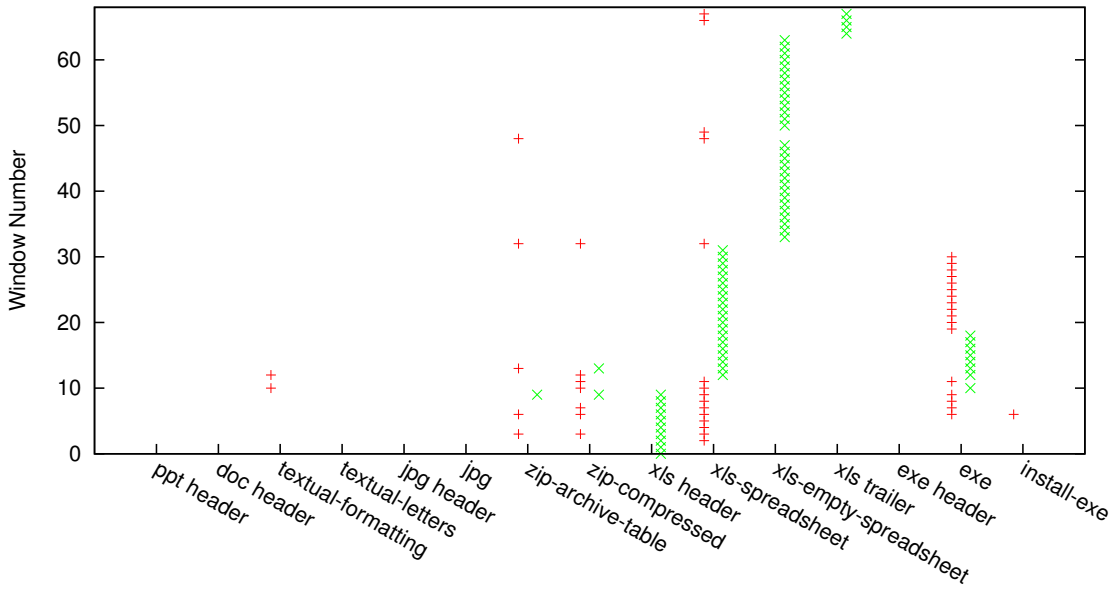

Figure 2. Secondary analysis of Food_Storage.xls.

2.6 SÁDI Implementation

SÁDI's task flow [15], as illustrated in Figure 3, is based on a continuous feedback loop paradigm to allow for the continuous improvement and refinement of the data differentiation parameters. This task flow diagram exemplifies the process of computing the initial statistical matching parameters and applying those parameters to test data for data type identification. Such cyclic improvement available through this process is critical, given the continuous deployment of new or modified file formats; i.e., the Microsoft Word format changes with every new release of Microsoft Word. This ever-changing nature of file formats adds to the difficulty of differentiating file formats, as files that are the same essential type can have slight variations.

2.6.1 Data Preprocessing

Through the data preprocessor, we generate the statistical result files. These statistical result files are plain text and contain the statistics for each window from the original file. At the beginning of each of these statistical result files are three lines of text containing the name of the

original file (including any file extension), the size of window used, and other formatting information. Following these three lines, the remainder of the file is organized as a very large table of values. Each column contains the values for a single statistic and in the cases of the distribution statistics, an array of values. Each row corresponds with the values for a single window from the original file. The statistical result files generated by the preprocessor are then used as input into the analysis code. These are also the results used to initially construct the list of data type characteristics used in the identification process.

2.6.2 Analysis Code

The analysis code is organized such that each file's window data is looped-over multiple times (once for each statistic for each data type); hence, the innermost loop is over the window data. The Big O complexity is O(f*d*s*m) where f is the number of files to be analyzed, d is the number of potential data types involved in the analysis, s is the number of statistics for each data type, and m is the length of the file.

Accuracy enhancing features of the environment include:

- Flexibility in setting weights for the values of the data type statistics
- Inclusion of a percent match total for the block as a whole (facilitates file identification) rather than only on a per window basis
- Addition of timing code for performance evaluations
- Extension to use of a two pass analysis, with the second pass performing pattern matching.

The benefit of a flexible weighting scheme for the data types' statistics is in the ability to weight more unique statistics higher; therefore, they will have a greater effect on the resulting identification process than less unique statistics.

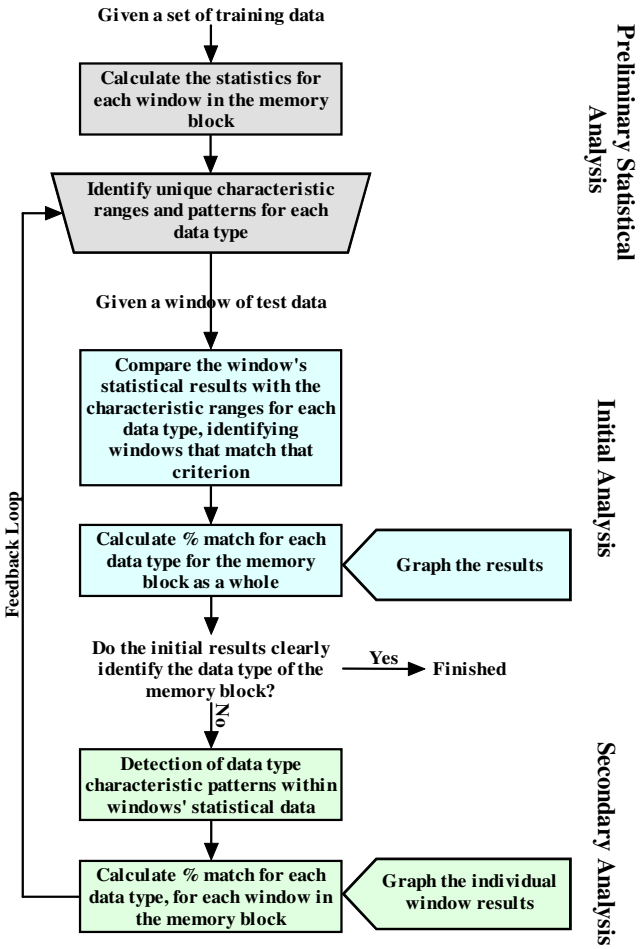

Figure 3. Analysis process.

The second pass analysis, i.e., pattern matching, was only performed on those memory blocks that could not be identified with the faster initial analysis pass. This improved overall performance without sacrificing the accuracy of the entire analysis process. Currently, this pattern matching is primarily focused on detecting xls files.

2.7 Evaluation

To evaluate the accuracy of SÁDI, we gathered 25 files of each data type being analyzed: bmp, csv, dll, exe, html, jpg, txt, and xls, for a total of 8 different data types and 200 files. The first five files of each data type were used to identify the unique characteristics of each data type

initially and specify the statistical parameters used for the testing portion of the evaluation. Next, the files were run through the preprocessor to generate the statistical result files that were then used in the automated analysis to identify the data type(s) contained in each.

The results from the initial analysis pass without pattern matching are shown in Table 2, all requiring a minimum percentage match of 92% before a type could be considered a match. These results have combined csv, html, and txt data into one type as well as dll and exe data into one type. By adjusting this minimum percentage, we can vary the resulting percentage of correct matches, which in turn, alters the false positive and false negative rates. The ideal minimum percentage varies depending upon type. Worth noting is the fact that although xls is included here, a secondary pass had not yet been done and, therefore, much of this data were incorrectly identified.

Table 2. Initial Analysis Results.

	# of files	Bmp	Dll and exe	Textual	jpg	xls	% Correct
Bmp	25	16	9	0	0	0	64%
Csv	25	0	0	25	0	0	100%
Dll	25	6	19	0	0	0	76%
Exe	25	6	19	0	0	0	76%
Html	25	0	0	25	0	0	100%
Jpg	25	1	7	0	17	0	68%
NULL	0	0	0	0	0	0	100%
Text	25	3	2	20	0	0	80%
Xls	25	22	1	1	0	1	4%
Totals	200	54	37	71	17	1	74.2%

Table 3 shows the results for xls data when a secondary pass has been taken into account. The incorporation of the pattern-based analysis greatly improves the results over the single pass only analysis. The most influential reason for the terrible results for xls data when only considering the initial pass is that many Excel files by default include two sheets of blank data that most users simply leave blank and do not delete. These blank spreadsheets are stored primarily as NULL data, thus causing most xls files to have a relatively high match for null data. Even without the secondary pass, the results for xls data improve when the NULL data type is combined with the xls data type, producing a 64% accuracy compared with the 4% accuracy achieved when considering the two data types separately.

The results of the analysis of the 25-bmp files are shown in

Table 4. The identification of matches with textual data was the key to improving the accuracy of bmp matches. When considering the dissimilarity of bmp and textual data, it was very odd to see so many files identified as textual data.

Upon an inspection of the original files being matched to the textual type, it was discovered that the files were of the form shown in Figure 4. Hence, the identification of the files as not actually being bmp files but text files, and the analysis correctly identified them as such. These appear to be source code files with embedded bitmap data, for instance, to act as icons. Those files were then replaced with actual bmp files for use in the results shown in Table 1.

Table 3. Secondary Analysis Results for Xls Data.

	# of files	bmp	Dll and exe	Textual	Xls	% Correct	False + %
Xls	25	2	3	1	19	76%	0

Table 4. Secondary Analysis Results for Bmp Data.

	# of files	bmp	Dll and exe	Textual	Xls	% Correct	False + %
Bmp	25	11	7	7	0	44%	0

Also of note is the combination of dll and exe types. Because both are compiled code and contain binary data, these types cannot usually be differentiated and commonly have the exact same characteristic ranges. When considering them as separate types, both tended to match dll because that is the slightly more unique type, and most exe data do fall within the bounds of the dll data type. Future research will consider if the byte distribution has any effect upon the differentiation between these two similar types.

When considering jpg data, although most jpg files were correctly matched, there were several that matched the wider range found within the dll and exe data types. This percentage is expected to fall as more files are tested. However, given the binary and compressed nature of jpg data, there will always be a subset of files falling just outside of the jpg data characteristic range, thus failing to match jpg data as accurately as other types of binary data, such as dll and exe data.

```
#define info_width 8
#define info_height 21
static unsigned char info_bits[] = {
 0x3c, 0x2a, 0x16, 0x2a, 0x14, 0x00,
0x00, 0x3f, 0x15, 0x2e, 0x14, 0x2c,
 0x14, 0x2c, 0x14, 0x2c, 0x14, 0x2c,
0xd7, 0xab, 0x55};
```

Figure 4. Content of info.bmp. This is bitmap data in a textual form.

Table 5 gives the results with a varying minimum percentage; by varying the minimum percentage required for a match, we obtained the most promising accuracy for each data type. This table identifies the changes (improvements) in accuracy achieved by letting the minimum percentages float to optimal values independently for each data type. The minimum percentage is the minimum percentage required to have the corresponding type be considered a match. The data being analyzed were identified as being of the most unique type that still meets this minimum percentage requirement. For example, some files containing data match the bmp type 100% but also match the text data 94.5% and dll/exe 97%. If the minimum required percentage were 95%, the data would be categorized as dll data since that is the most unique type that still meets the minimum percentage match requirement. If the minimum percentage required for a match were instead 92%, the text type is more unique and meets this requirement, so the data would be identified as text. The xls data type has been left out as the most accurate results for that data type comes from the secondary analysis.

To differentiate between the three kinds of textual data, the distribution of byte values is used. Although not yet fully incorporated into the automated analysis, we obtained some preliminary results to verify this differentiation is possible for most files. Each data type has a

Table 5. Varied Minimum Percentages.

	# files	Min. %	Bmp	Dll and Exe	Textual	Jpg	Null	Xls	% correct
Bmp	25	95	16	9	0	0	0	0	64%
Text	75	92	3	2	70	0	0	0	93%
Dll and Exe	50	92	12	38	0	0	0	0	76%
Jpg	25	88	1	6	0	18	0	0	72%

byte distribution of average values. A script compares each byte distribution value from a window with the expected average byte characteristic value(s) for the given types. The number of matching values is calculated for the window's distribution, producing a window match percentage. The file's match percentage is calculated by summing all of the windows' match percentages and then dividing by the total number of windows. If this match percentage is greater than 80%, the file is counted as a match for the corresponding data type. The number of matched files is then divided by the total number of files analyzed, producing a percentage representing the number of correctly matched files. These percentages are presented in Table 6. Here, we applied distribution analysis in order to differentiate csv, html, and txt files. This distribution analysis relies on the presence of unique frequently occurring bytes in each of the types, such as comma in csv files. Each row represents the files' data of the corresponding type while the values in each column represent the percentage of files matched for that column's type. For example, csv files have 96.0% of the files matching csv data while 0.0% of the files matched html data and 4.0% of windows matched plain text data. Therefore, more csv files match the csv data type than any other data type.

Although the accuracy of the html data is reasonably high (84.0%), it could be higher for many files. Out of the 25 html files analyzed, four matched textual data better than html. When examining the original files, one reason for this is the presence of JavaScript code within the html proposed

Table 6. Percentage of Matched Files from Textual Distribution Analysis.

	Csv	**Html**	**Txt**
Csv	96.0%	0.0%	4.0%
Html	0.0%	84.0%	28.0%
Txt	4.0%	0.0%	80.0%

file. The other html files contained much less of this embedded code or none at all and were accurately identified as html data. This is yet another example of the superiority of the technique's ability to identify embedded data.

2.8 Conclusion

The results presented here are highly promising. The technique has been shown to accurately differentiate data types. This is highlighted in the results from the initial analysis of bmp files of which seven textual files were unknowingly included and were then identified as textual data rather than bmp data. In general, accuracy rates are far better than prior techniques and do not rely on header information, file extensions, or any other form of metadata.

With regard to false positives, SÁDI compares well with previous techniques. False positive percentages range from 13.6% for dll and exe data through 10.67% for bmp data and down to 0% for jpg and xls data. One reason for the higher false negative percentages is that some files match a more unique type or a slightly less unique type. For example, most dll and exe data were incorrectly identified as bmp data, which has wider ranges and hence allowed for data with more varying values. Also, most incorrectly identified bmp data alternatively are identified as dll and exe data because they happen to fall within the smaller ranges of the characteristics of the dll and exe data types.

Even in cases wherein covert channels are used to obfuscate data [9], SÁDI can be an effective analysis tool. One such covert channel method is to utilize file slack space, or extra space, in file headers to hide data [9]; the SÁDI technique would identify this embedded or appended data simply because of its existence and its differing statistical structure from the rest of the file.

Bmp data presents a challenge because technically any binary value is a valid value; therefore, depending upon the colors present within the bmp picture, the file can fall into a much

narrower category. An extreme example is a bmp that is simply a black background, which is stored as all zeros and, hence, is identified as NULL data. This example is extreme because NULL is at the opposite end of the spectrum of data types due to the single value range for all statistics, compared with bmp that has the broadest range for all statistics.

2.9 Future Work

Further study of the effect of varying window sizes, both in identifying an optimal window size and in identifying applications outside of forensics, is warranted. For instance, SÁDI could benefit virus scanners and in identification of copyright or privacy violations. Similarly, simply identifying covert channels or unusual dissemination can aid identification of violations of trade secrets or other malicious uses of covert channels.

Currently, we are looking to extend SÁDI to support more base types as well as the more difficult container type files, such as Microsoft Word, in order to identify the data types embedded therein. Doing so will make SÁDI far more generally applicable to the detection of hidden data and allow SÁDI to locate information even when no file system is available. Doing so, however, will require applying SÁDI to fragments of data rather than to entire files and identify dynamically when the data type is substantially changing.

When dealing with embedded data or data that has been appended, the technique also needs to identify the location of the data. To identify specific starting and ending locations of these kinds of data further analysis would be needed to break down the windows into byte values.

2.10 References

[1] Bryan Carrier, *The Sleuth Kit*. Available: http://www.sleuthkit.org/sleuthkit/desc.php, accessed on 2007-11-30

[2] Darwin, *file(1)*, Online Linux man page for the file command. Available: http://linux.die.net/man/1/file, accessed on 2007-11-30.

[3] *Encase® Enterprise*. Available: http://www.guidancesoftware.com/products/ee_ index.aspx, accessed 2007-11-30.

[4] Robert F. Erbacher and John Mulholland, "Identification and Localization of Data Types within Large-Scale File Systems," In *Proc. of the 2nd International Workshop on Systematic Approaches to Digital Forensic Engineering*, Seattle, WA, April 2007, pp. 55-70. *Best Paper Award.*

[5] Dan Farmer and Wietse Venema, *The Coroners Toolkit* (TCT). Available: http://www.porcupine.org/forensics/tct.html, accessed on 2007-11-30.

[6] *FTK*® AccessData. Available: http://www.accessdata.com/common/pagedetail. aspx?PageCode=ftk2test, accessed 2007-11-30.

[7] Gregory A. Hall and Wilbon P. Davis, "Sliding Window Measurement for File Type Identification." Avaiable: http://www.mantech.com/cfia2/SlidingWindow MeasurementforFileTypeIndentification.pdf, accessed 2007-11-30.

[8] Douglas J. Hickok, Daine R. Lesniak, and Michael C. Rowe, "File Type Detection Technology," in *Proc. from the 38th Midwest Instruction and Computing Symposium*, Apr. 2005, http://www.micsymposium.org/.

[9] Neil F. Johnson and Sushil Jajodia, "Steganalysis: The Investigation of Hidden Information," *IEEE Information Technology Conference*, Syracuse, New York, 1998, pp. 113-116.

[10] M. Karresand and N. Shahmehri, "File Type Identification of Data Fragments by Their Binary Structure," in *Proc. of the IEEE Information Assurance Workshop*, West Point, NY, June 2006, pp. 140-147.

[11] M. Karresand and N. Shahmehri, "Oscar – File tType Identification of Binary Data in Disk Clusters and Ram Pages," *Proceedings of IFIP International Information Security*

Conference: Security and Privacy in Dynamic Environments (SEC2006), LNCS, 2006, pp. 413-424.

[12] Binglong Li, Qingxian Wang, and Junyong Luo, "Forensic Analysis of Document fragment based on SVM," *Proceedings of the 2006 International Conference on Intelligent Information Hiding and Multimedia*, Pasaena, CA, December 2006, pp. 236-239.

[13] Mason McDaniel and M. Hossain Heydari, "Content Based File Type Detection Algorithms," *Proceedings of the IEEE 36th Hawaii International Conference on System Sciences (HICSS '03)*, Washington, DC, 2003, pp. 332.1.

[14] Rodney McKemmish, "What is Forensic Computing?," *Trends and Issues in Crime and Criminal Justice*, June 1999, published by the Australian Institute of Criminology. Available: www.aic.gov.au/, accessed 2007-11-30.

[15] Sarah Moody and Robert F. Erbacher, "Automated Identification of Data Types for Use in Computer Forensics," Poster presented at the *2007 Grace Hopper Conference*, Oct. 17 Poster Session, Orlando, FL.

[16] *ProDiscover® for Windows*. Available: http://www.techpathways.com/prodiscoverWindows.htm, accessed 2007-11-30.

[17] G.J. Simmons, "The Prisoner's Problem and the Subliminal Channel," in *Proc. of CRYPTO* '83, 1984, pp. 51-67.

[18] Salvatore J. Stolfo, Ke Wang, and Wei-Jen Li, "Fileprint Analysis for Malware Detection," *Proceedings of WORMS 2005*, Fairfax, VA, November 2005.

[19] *TrID,* published by Marco Pontello. Available: http://www.brothersoft.com/trid-20596.html, accessed on 2007-10-30.

[20] http://berghel.net/publications/data_hiding/data_hiding.php

CHAPTER 3

AUTOMATED DATA TYPE IDENTIFICATION AND LOCALIZATION USING SÁDI – STATISTICAL ANALYSIS DATA IDENTIFICATION[2]

3.1 Introduction and Motivation for the Work

In his introduction to McKemmish's *What is Forensic Computing?* [13], Graycar states, "More and more, information technology is becoming the instrument of criminal activity." McKemmish then states [13],

> "The application of computer technology to the investigation of computer based crime has given rise to a new field of specialization – forensic computing – which is the process of identifying, preserving, analyzing and presenting digital evidence in a manner that is legally acceptable."

McKemmish continues by presenting a list of what he considers to be the four key elements in forensic computing. The first of these key components is the identification of digital evidence. Whenever a computer based crime is committed, the perpetrator leaves behind digital evidence of his or her actions. This evidence may be found on any type of digital storage device, including mobile/cellular phones and electronic PDAs as well as the more obvious devices such as a personal or laptop computer. A digital forensic analyst must be able to identify the evidence, or lack thereof, that might be found on any number of these digital storage devices. This highlights the need to be able to the format and type of digitally stored information so that the forensic analyst can retrieve the relevant portions of the data and utilize that information in the investigation [13]. This research presents a new and unique technique for doing just that, identifying the type of data on a digital device and its storage format based upon the values of the

[2] Co-authored by Dr. Robert F. Erbacher

stored data bytes and a statistical analysis of those values. This new technique is termed statistical analysis data identification, or SÁDI.

In today's world, there is a vast amount of digital data, which presents a challenge to forensics analysts. McKemmish points out the time and effort required to identify and retrieve necessary information is much greater than with other similar activities [13]. SÁDI incorporates the automation required for such specialized data identification tools to be useful and applicable in real world applications. The SÁDI technique utilizes the byte values of the data stored on a digital storage device in such a way that the accuracy of the technique does not rely on the potentially misleading metadata information but rather the values of the data itself. SÁDI identifies what digitally stored data actually represent and selectively extracts portions of the data for additional investigation. This can, in effect, be considered the same end goal as file type identification; SÁDI is just working from a different perspective to achieve the end result of data identification.

3.2 Research Problem

In the field of computer forensics, there have been many techniques developed that aid in the identification of file types. However, none of the techniques currently available have acceptable accuracy in the detection of file types except when relying upon file header information, file extensions, fixed magic numbers and other such information associated with the file. The most common method is to use the file extension to identify file type; this method, however, is extremely unreliable, as in most cases the extension is changeable by any user or application. Many operating systems will not open a file that has been renamed with an incorrect extension, and some virus scanners will also not scan files unless they have the executable extension [12]. The Windows operating system utilizes this method of relying upon file extensions to identify files. File types are associated with specific file extensions; these are preset

in the MS operating system, and if a user wishes to change the associations, s/he must do it manually [19].

File header information and fixed magic numbers depend upon predetermined patterns and/or values in the header portion of a file to determine the type of the file. UNIX systems generally utilize a command entitled *file* to identify file types. This command utilizes three different methods to attempt to identify the parameters passed to it. The first method uses system information to recognize system files, the second utilizes magic numbers found in the first 16 bits of a file, and the third method utilizes any ASCII content within the file to categorize the ASCII data according to language (such as C or a troff input file) ([12] and [2]). For the magic number test to accurately identify the file, the magic number found in the first 16 bits of the file must be found in the */etc/magic* file and be associated with the correct file type as described in [2]. Magic numbers must, therefore, also be predefined before the generation of any associated files so that each file can have the magic number built into the header information. This also prevents magic numbers from being useful in situations dealing with file fragments or obfuscated files and data.

File header information and other embedded magic numbers can also be manipulated to prevent the file from being identifiable by techniques that use this information. In addition, the magic number file itself (*/etc/magic*) can also be modified by a user to misrepresent files. Some methods of manipulation include altering byte values located in the header or encrypting information by using any sort of cipher or encryption technique. This manipulation generally prevents all current techniques from accurately being able to identify file type.

Therefore, all of the above methods of identification are easily circumvented. More information about the drawbacks, as well as the advantages, of using either magic numbers or file extensions can be found in [7]. A discussion of the drawbacks and advantages of using a byte frequency distribution for identification purposes in found in [7], this will be discussed more in Section 3.3.

Another problem with focusing on identifying file type rather than data type is the issue of embedded data. It is very easy for a criminal to hide a table of child porn sales, for example, within a very large word document such that it is not discovered. Current methods do not allow for the identification of this embedded table, and so to find this item of evidence, an analyst must open up every file and look through the entire contents of each to see if any embedded data were located in it and if so, determine if that data is applicable to the situation. When considering the number of files on a typical hard drive, this process is not a viable option for the analyst. Current forensics tools lack the ability to find and locate embedded data, thus causing resources to be spent trying to locate information which might or might not be somewhere on a hard drive. The alternative to this large expense of time and effort is to just assume no applicable embedded data exists; this can obviously be a disastrous assumption to make. The most popular forensic tools, including Encase, FTK (Forensics ToolKit), and ProDiscover, as well as many other forensic tools, lack this capability of identifying embedded data [1] [3] [5] [6] [15].

Another tool for identifying files is a Freeware product called TrID [18]. TrID utilizes the binary structure of files to identify them based upon recurring patterns found within files of the same type. An XML database is used to store the definitions for the various file types. According to [18], TrID can be trained to recognize more file types by using another module, TrIDScan, to scan sample files of the same type to generate a new definition for that file type. This tool was not designed as a forensic tool and, therefore, does not take into account situations involving steganography or other such purposefully manipulated and disguised data. It also has no available documentation on the accuracy or false positive/negative rate. Therefore, although TrID can be useful for many computer users, it cannot be considered a forensic tool, nor does it provide more capabilities in regard to file identification than already provided in current forensic tools previously listed.

The proposed technique can identify and, with additional processing, specifically locate embedded data, as well as data hidden by using other techniques including steganography. This research helps fulfill this pressing need of aiding the data analysis process. Even in cases wherein steganography is used to obfuscate data, such as those discussed in [8], SÁDI is an effective steganalysis tool. One such steganographic method mentioned in [8] is to utilize file slack space, or extra space, in file headers to hide data; the SÁDI technique can identify this embedded or appended data simply because of its existence and its differing structure from the rest of the file. In cases where in the embedded data is of the same type as the data in which it is being embedded, if there is any sort of fluctuation in the byte values (such as might be caused by header information on the embedded data), SÁDI identifies such fluctuations and flags them for further analysis.

SÁDI involves taking a block of memory (a simplistic example is a single file) and performing a statistical analysis on it. The blocks of various file and data types are first preprocessed, using a sliding windows technique, and the various statistics are calculated for each window and the memory block as a whole. For further discussion of this preprocessing technique, see [4]. These blocks are then analyzed to identify the unique characteristics of each of the data types. These characteristics are then used in the analysis of memory blocks with unknown types to identify the data types within those blocks on a per window basis. We improved the analysis through feedback generated from correct and incorrect matches throughout the course of this research. This feedback improved the data type characteristics which, in turn, increased the accuracy of the results.

3.3 Background and Related Work

This research extends the work of Erbacher and Mulholland in [4]. In this previous work, the technique and an analysis of the potential of the technique were presented. The current

research extends the proof-of-concept technique found in [4] and implements it. An analysis is given evaluating the accuracy, efficiency, and performance of the implemented algorithm.

Other work in the area of data type identification is found in [12]. The authors use three different techniques to identify files. Note, however, that although these detection algorithms utilize file content to perform the identification, the overarching goal of these techniques is to identify only the file type, not the data type contained within the file. Consequently, the techniques are not able to accurately identify embedded data. The first technique presented by McDaniel and Heydari is a byte frequency algorithm (BFA) which finds the average frequency of each byte value (0-255) for the entire file and then performs a normalization of the data. After doing this for many files of a particular type, a file type fingerprint is created that incorporates the byte frequency values of all files of the chosen type and allows an analyst to ascertain whether there are any particular byte values that are more common for the chosen file type. To identify unknown files, the byte frequency distribution is calculated for each unknown file and compared to each type's fingerprint; the closest match is then considered the type of the unknown file. During testing, this algorithm had an average accuracy rate of 27.5%. The second algorithm, the byte frequency cross-correlation algorithm (BCA) extends the process to look at correlations among byte values. It had an accuracy of 45.83% and performed a much slower analysis than the BFA method. The third algorithm, the file header/trailer algorithm (FHT) looks for patterns and correlations in a certain number of bytes at the beginning and ending of a file. Although this method achieved an accuracy of 95.83% during testing, it reduced the problem to a noncontent-based approach which only relies upon headers and trailers.

In [9], Karresand and Shahmehri present an algorithm that utilizes the measure of the rate of change of the byte contents of a file and extends the byte frequency distribution-based Oscar method mentioned from their research [10]. This original Oscar method utilizes a byte frequency distribution and then a calculation of the mean and standard deviation for each byte value. This

forms a model, called the centroid by the authors, for the file type. An unknown file fragment is compared with this centroid value, and if the sample fragment is close enough to the given centroid, the fragment is classified as the file type associated with that specific centroid. In [9], this Oscar method is extended to incorporate the ordering of the byte values in a fragment, using the rate of change between consecutive byte values. This method achieved 92.1% detection rate with a 20.6% false positive rate for JPEG files. Zip files achieved a detection rate of 46% to 80% with a false positive rate of 11% to 37%, while exe files only achieved a detection rate of 12.6% and a false positive rate of about 1.9%. For both zip files and exe files, as the detection rate increased so did the false positive rate. Overall, the techniques discussed in [9] and [10] are only useful for identifying a fragment as a JPEG fragment or not, as opposed to actually identifying the type of the fragment from among all types. These Oscar-based methods focus on identifying file type and, hence, have the same drawbacks as those methods developed by McDaniel et al.

Li et al. [11] present a technique closer to SÁDI than either of the two previous techniques. Li et al. utilize an n-gram analysis, and by utilizing a frequency distribution of byte values, generate a fileprint representative of all files of a given type. The accuracy of this fileprint is generally much better than the previous techniques, although no false positive or false negative rates are given. Also, the fileprint combines doc, ppt, and xls file types into one Microsoft Office type, and the dll type is combined with the exe file type, as they have similar fileprints. Although in testing, a fileprint generated more positive results than the previous methods, the technique still is not very applicable to file fragments and, consequently, to data type identification in general. The fileprint method of Li et al. also has the stated drawback of combining various types together to increase accuracy. An interesting highlight noted by the authors is a modification to the technique to increase efficiency and accuracy. By truncating the fileprint to only analyze a smaller portion of the first part of the file, with the highest degree of accuracy being when only the first 20 to 200 bytes are analyzed and the rest of the file is ignored,

a higher accuracy percentage is obtained as well as a much more efficient program (since most of the file is being ignored). Although this does decrease execution time, it prevents the technique from even having the potential of identifying embedded types or in even recognizing their existence within an analyzed file. So, even if a file is accurately recognized, there is no guarantee that it does not contain obfuscated, embedded, or appended data which could potentially be malicious or of forensic worth in an investigation. Li et al. specifically designed their technique to answer the question, 'Given file F and a set of potential file types M_i, can we correctly identify which type the file is or if it is an invalid type not within the set of potential types?' Therefore, it was not designed to be used in situations in which the data type is unknown but discovering what the data actually represents is the desired outcome.

The methods in [9], [10], and [12] are unable to achieve high accuracy rates while concurrently keeping the false positives rates low except when the method focused on header and trailer information or other special file identification numbers (such as are found in JPEG files). Unfortunately, such information can be manipulated or avoided by many data hiding and manipulation techniques. The method in [11] achieves better accuracy but still does not address the problems of steganography and data obfuscation, nor is it able to accurately differentiate all file types. These methods did, however, pave the way for the concept of the SÁDI technique, initially developed by Erbacher and Mulholland in [4] and more fully developed here to provide the capabilities unavailable in earlier techniques.

3.4 Research Project Goals, Approach and Uniqueness

The goals of this research project were to 1) develop algorithms for data type identification and localization and 2) analyze the effectiveness and accuracy of the developed algorithms. The algorithm was designed to identify and locate date types within a file system and automate this process. As specified by its name statistical analysis data identification (SÁDI),

this method uses various results from a statistical analysis to perform the data type identification. These include average, kurtosis, distribution of averages, standard deviation, distribution of standard deviations, and the unique characteristics from the distribution of byte values. These particular statistics are the most unique in differentiating between the various data types, as will be discussed later in more detail. No other currently available methods utilize these statistics. In [12], McDaniel and Heydari utilize a byte frequency distribution which is identical to the distribution of byte values used in SÁDI; however, SÁDI utilizes the information differently and only concentrates on uniquely identifying distribution values. In [9], the same basic byte frequency distribution is utilized to identify file fragments along with a few other modifications mentioned earlier. SÁDI again utilizes the byte frequency distribution information differently by focusing only on the unique portions for each type. The most novel addition the SÁDI technique makes is the addition of the other statistics to the analysis process and the utilization of a sliding windows technique to allow the identification of fragments of data.

These novel methods were chosen primarily to obtain greater accuracy in the identification process and to reach the goal of data type identification rather than file type identification. This differing focus is the key to achieving greater accuracy in the analysis. Focusing on data types allows SÁDI to handle situations involving data fragments as well as embedded and otherwise obfuscated data. In the future, research on SÁDI can expand to evaluate how it can handle situations involving more sophisticated cryptography, steganography, as well as other data manipulations.

Because the SÁDI technique involves taking a memory block and identifying the data types within the block based upon the block's statistical analysis results, it should also be able to identify what type of file the memory block is (if it is only one file), based upon the composition of data types and the format of the available data. For example: "Is the data examined similar to that which would occur in a PDF file or is it in more of a raw form as would be seen in a doc?"

SÁDI is also unique in that only the content of the memory block is used in the identification of the data types, and it can identify pieces and fragments of files with acceptable accuracy. The memory blocks are divided up into smaller chunks or fragments usually in sizes of 256, 512, or 1024 bytes. These chunks are then analyzed separately, hence allowing the SÁDI technique to identify file fragments given enough data (at very least 64 bytes, preferably one of the three sizes listed previously). All data is analyzed in a consistent manner without any user prejudice as to what the data is supposed to mean. Earlier methods focusing on file type identification contained this drawback by assuming all data within a file are of the same type. Several file types can easily have other data types embedded within them, such as the example of a spreadsheet table embedded within a Word document. In these cases, the identification of the file type is subsumed by the more important issue of identifying data type. The SÁDI technique focuses on data type identification. This change in focus, one of its many strong points, allows embedded data to be identified and localized. The fact that some file types have other data types embedded within them is a major reason why earlier methods struggle to accurately identify file types based on an analysis of the content only. The types found within a file are not solely indicative of file type but also of the component data types.

The following objectives embody the central process to the development of SÁDI:

- Analyze the pros and cons of previous methods of identifying file types and utilize this knowledge to assist in the development of a more efficient and accurate algorithm (much of this work was done in [4])
- Develop an algorithm to locate and identify data types within a block of digital memory
- Evaluate the effectiveness and accuracy of the algorithm, including calculating false positive and negative rates

- Utilize formal methods to verify the accuracy of the algorithmic method developed. This was done through precise documentation of procedures used and other adjustments made to the algorithm to obtain an increase in accuracy as well as a thorough testing of the algorithms to verify their integrity
- Evaluate the capability to automate the algorithm and implement the automation
- Analyze the effectiveness, efficiency and applicability of the automated algorithm

3.5 Methodology

3.5.1 Statistics and Data Types

As a starting point, only base data types were studied until the accuracy of the SÁDI method was verified. Base types are those for which the entire file can be considered to be of the same data type (ignoring header information). Some examples include jpg, png, dll, and mp3. The algorithms were later extended to include other data types and, additionally, to identify file types and the component data types within the files. The reason behind this segregation of types was to verify the techniques on simple cases and then expand to accurately handle container-like files, such as doc and zip files which present a more advanced and complicated problem. To develop this algorithm, we identified unique characteristics of data types in the various statistical measurements. We used the work of Erbacher and Mulholland [4] for the most useful statistics for data type identification (regarding currently studied data types). These include average, distribution of averages, standard deviation, distribution of the standard deviations and kurtosis, to which was added the distribution of byte values.

Average – (on a scale of 0-1, the normalized length of the file) the average of the byte values for each window *i*. A file-based average is the average of the set of window byte value averages. N denotes the number of bytes in the window or, in the file-based case, the number of windows.

$$\tilde{X}_j = \frac{1}{N}\sum_{i=1}^{N} X_i$$

Distribution of Averages – (on a scale from 0-255) the probability that an average chosen from all the averages of a memory block is of value B in the range 0-255

$$D_{\tilde{X}_B} = \Pr((B+1) > \tilde{X}_j \geq B)$$

Standard Deviation – (on a scale of 0-1, the normalized length of the file) the standard deviation of the byte values of a window from the average for the window

$$S_j = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(X_i - \tilde{X}_j\right)^2}$$

Distribution of Standard Deviations – (on a scale from 0-255) the probability that a standard deviation chosen from all the standard deviations of a file is the value B

$$D_{S_B} = \Pr((B+1) > S_j \geq B)$$

Kurtosis – (on a scale from 0-1, the normalized length of the file) the peakedness or consistency of the data calculated from two different modifications of the standard deviation. The numerator is the standard deviation squared except with a fourth power instead of just a square and the denominator is the standard deviation taken to the fourth power.

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

Distribution of Byte Values – (on a scale from 0-1) the probability that a byte chosen from all the bytes in a window is the value of B. Note: only unique values are used in the analysis.

$$D_{X_i} = \Pr((B+1) > X_i \geq B)$$

These statistical characteristics are then utilized in the algorithmic analysis of the digital data to uniquely identify data of each data type. In various cases, the other statistics mentioned in [4] are helpful in increasing accuracy and differentiating between very similar data types.

3.5.2 Identifying Unique Data Type Characteristics

To discover and identify the unique statistical characteristics for each data type, the results from test memory blocks were visually studied through a time intensive process, the outcome of this manual process being a textual file containing information about the unique characteristics for each data type. This information was then used in the automated algorithm to perform the statistical analysis identifying the various data types found within the analyzed memory blocks. The use of this textual data input file allows easy modification of these values to obtain greater accuracy. Due to the shifts in the statistical results for the different window sizes, this textual input file is only applicable for the window size used to generate the characteristic data found therein. Hence, for varying window sizes, multiple characteristic input files are required.

The advantage to allowing for different window sizes comes from the fact that each window size produces slightly varied statistical results for the data types and hence may highlight differing unique characteristics that may not be as clearly visible at alternate window sizes. Window sizes are of the form 2^n, where n represents some integer power. From [4], the most helpful window sizes are between 256 and 1024; therefore, we initially analyzed files utilizing these window sizes. One disadvantage to utilizing multiple window sizes is that the manual identification of the data type characteristic information needs to be repeated with each window size, as the memory block data is more obfuscated with larger window sizes and more detailed and varying with smaller window sizes. A second drawback to using the sliding windows of varying sizes is that each of these characteristic input files is only helpful in analyzing data that

has been preprocessed using the same window size, i.e., the window size used in the analysis must correspond to the window size used to preprocess the memory blocks to obtain viable results.

Due to this drawback of manually constructing each characteristic input file for each window size individually, we primarily used a window size of 256 bytes. As of yet, we have not expanded into detailed investigations into larger window sizes for two main reasons, the first being that the results from the smaller windows sizes are very promisingly accurate, and the second reason being that in identifying file fragments or obfuscated data, the smallest fragment or data piece that can be identified is the same as the window size. Hence, by utilizing the smaller window size, SÁDI is more successful in accurately identifying small pieces of hidden data within larger blocks of memory. It should be noted that the use of larger window sizes could reduce run time and decrease memory requirements and might be more helpful in applications that require faster run times or have limited memory available to the program. This is an aspect to be researched in the future.

Some of the data types also have very unique identifying patterns for some statistics. The most unique example of this phenomenon is seen in the graph of the average statistical results from the spreadsheet data (referred to as xls data in Figure 1). This particular data type and statistic has a stair step pattern which is unique to only the average statistical results for spreadsheet data [4].

Referring again to Figure 1, except for the red line corresponding to the file Food Storage.xls, all of the lines have the obvious stair step pattern which is unique to spreadsheet data. This particular file is very small and does contain one sheet with a limited amount of data in it, followed by two empty sheets.

The empty sheets are represented by the horizontal sections between approx. 0.5 and 0.7 and 0.7 and 0.9. Due to the normalization between these files of differing lengths, the short stair

step pattern for Food Storage.xls is not clearly visible but is recognized in the automated analysis process. The automated analysis process performs up to two passes over the data. The second, optional, pass performs the analysis on a per window basis and utilizes any available pattern information. A graph of the results from this second pass clearly shows SÁDI's capability to identify component data types within a large memory block, as seen in Figure 5. In Figure 5, the green ◊'s mark windows matched the corresponding type by more than 80% while the blue +'s mark windows only matched between 50-80% and the red □'s represent a match between 0-50%. By looking at this figure one can see the empty sheets represented, by null data, of the xls file. The intermediate spreadsheet data between the empty spread sheets can also be seen in the xls data column.

Techniques similar to pattern matching can be used to identify these unique patterns in the statistical results of the data types. Initially, SÁDI is using a simple recursive approach which recursively checks for a new matching pattern in the data or the continuation of a previously started pattern match, but more sophisticated techniques can later be tested and incorporated into SÁDI. Incorporating the ability to have pieces of the pattern be optional would be helpful for some patterns.

One technique used for string matching with a similar capability, the Smith-Waterman algorithm discussed in [16], might be used as a basis for a similar type of matching technique in the automated data matching algorithm used in SÁDI. The Smith-Waterman algorithm has been show to also have the capability of being parallelizable, referred to as the Parallel SWAMP algorithm [17]. Thus, if the algorithm were used in this research, it would also inherit that advantage which would greatly improve performance. Even our initial simple recursive technique can be parallelized when analyzing multiple memory blocks, assigning each processor one memory block to analyze.

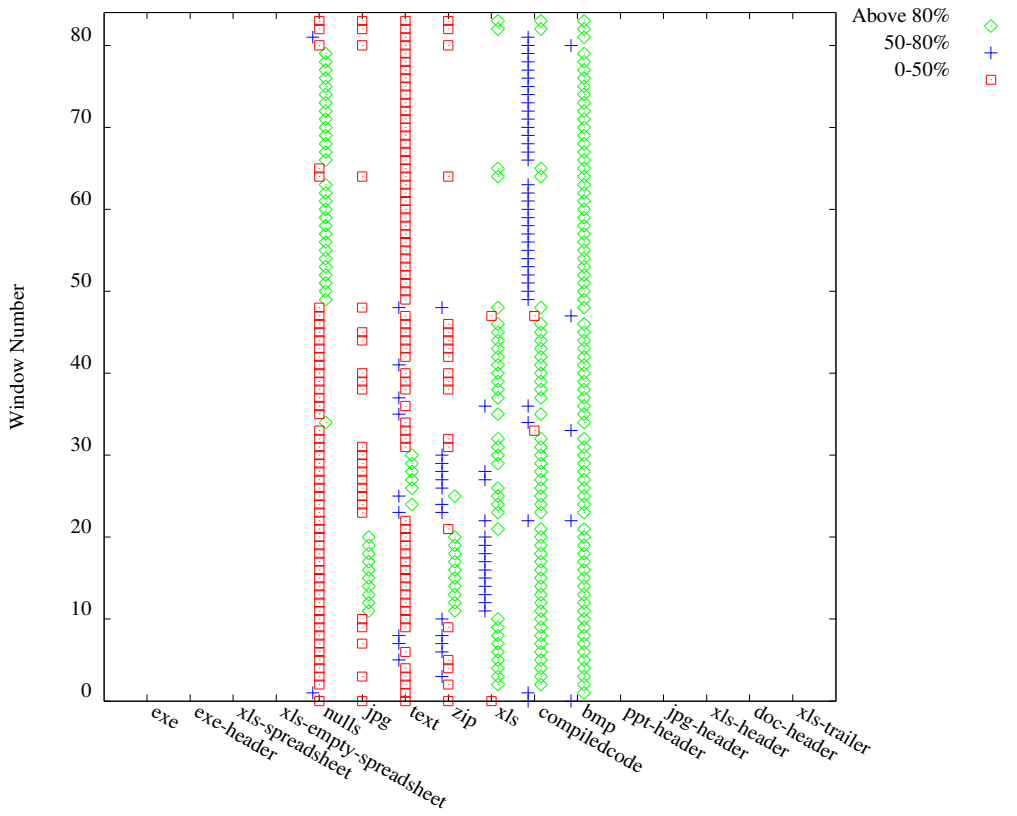

Figure 5. Secondary analysis of an xls file showing component sheets.

Work of other researchers in this research area was also studied to find other characteristics that might be useful in data type identification, specifically the research in [9] and [12]. As mentioned earlier, other statistical functions defined in [4] were also evaluated when working with data types not previously addressed.

The algorithm preprocesses the data to be analyzed by calculating each statistic for a memory block's windows. Then, these statistical values are analyzed using the characteristic information found in the corresponding input file (discussed earlier on in this section); based upon the results provided, an evaluation of what data types are found within the file and approximate locations of the start and end of each data type are presented to the analyst.

Output is provided to the user designating the most likely data types for each file, and any embedded or appended data can be identified on a per window basis similar to that shown for the component data types in the xls file shown in Figure 5. The algorithm does also locate the embedded and appended data down to the window; this will be extended to location down to the specific byte.

3.5.3 Evaluation of SÁDI

To evaluate the accuracy and efficiency of the algorithm, several sets of test data were generated that incorporate all the types included in the algorithm identification process, as well as some others to see how the algorithm handles the new data types. The algorithm was then used to analyze the set of test memory blocks and identify the data types within the blocks and the approximate starting and ending point of the data. This result was compared with the actual data types. The number of accurate positives, the number of false positives, and the number of false negatives were cataloged, and from these statistics we derived an efficiency rating and identified any weaknesses of the algorithm (i.e., does it have trouble identifying certain data types with or without certain circumstances; how does it handle previously unknown data types; are there specific data types that are consistently identified as another type and why, etc.). More work can be done later to better locate the data and allow it to be extracted from the memory block. The current work focuses on accuracy and the false positive/negative rate for the SÁDI technique. Initially, only base data types were analyzed to ensure the algorithm could handle the simple cases. Later, the study was expanded to include more types, including the types found in container-like files, such as doc, ppt, and pdf files.

This general process was repeated: re-evaluating the data type characteristic information in light of the false positive and false negative instances, adjusting the characteristic information

to obtain greater accuracy and avoid such false positives and negatives, and finally rerunning the test data as well as other data to evaluate the adjusted accuracy.

Throughout the development and testing of the automated algorithm, there was a continuous feedback loop so that each new memory block that was analyzed that did not fit the current set of characteristic information for the corresponding data types was added. Doing so enhanced the data type characteristics to obtain a more accurate set of characteristic statistical data for each data type. Also, through the use of this feedback loop, issues stated above regarding the weaknesses of the algorithm were addressed and are being alleviated. This feedback loop was and will continue to be the drive behind the iterative improvement process for SÁDI

This algorithm was compared with other current algorithms and techniques to evaluate its contributions to the field of computer forensics and how well it addresses the need for an accurate and reliable algorithm that performs type identification.

3.6 SÁDI Implementation

3.6.1 Implementation Overview

The implementation of SÁDI contained several integral steps which together complete the analysis technique. Figure 3 illustrates the correlation and connection between these component pieces, originally developed for [14] (a poster presentation of this same research at an earlier stage).

3.6.2 Data Preprocessing

For simplicity, we initially have all of our memory blocks be files; by sending these files through the data preprocessor, we generate statistical result files. These statistical result files are just plain text and contain the statistics for each window from the original file. At the beginning of each of these statistical result files are three lines of text containing the name of the original file (including any file extension), the size of window used, and other formatting information.

Following these three lines, the remainder of the file is organized as a very large table of values. Each column contains the values for a single statistic and in the cases of the distribution statistics, an array of values. Each row corresponds with the values for a single window from the original file. The statistical result files generated by the preprocessor are then used as input into the analysis code. These are also the results used to initially construct the list of data type characteristics used in the identification process.

3.6.3 Analysis Code

The analysis code itself went through several revisions throughout the testing and implementation of SÁDI and will undoubtedly continue to be refined over time. All versions have several nested loops to facilitate a complete analysis of the targeted memory blocks. In the initial implementation, the outermost loop dealt with the files to be analyzed with the next loop looping over the windows in each file and the inner loops looping over the various potential data types and their characteristic statistical information.

In later implementations, these loops were reordered so that instead of looping over the data types' statistical information multiple times (once for each window in the file), each file's window data is looped over multiple times (once for each statistic for each data type). Consequently, the innermost loop was over the window data. In the implementation used to generate the results here, both were used at different points. The initial implementation with the innermost loops going over the potential data types and statistics was used in the initial analysis and in the additional pass to differentiate textual data types. The pass that performs pattern matching utilizes the implementation with the window data as the innermost loop. By incorporating both implementations, the performance improved noticeably, and less computation is required. This also allows data to be written out to file during the initial analysis rather than being forced to be stored in memory until the analysis is completed. In the initial analysis, each

window is completely analyzed before moving to the next; therefore, the results for that window are written out to the resulting output file rather than being kept in memory until all windows have been analyzed.

The overall Big O complexity of both orderings is the same: O(f*d*s*m) where f is the number of files to be analyzes (or memory blocks in the more generic case), d is the number of potential data types involved in the analysis, s is the number of statistics for each data type, and m is the length of the files. Obviously, these values vary depending upon the exact files being analyzed and the data types and statistics used. However, in either implementation for the same set of inputs, the Big O complexity is identical until extra calculations are included. This is done in order to maintain the ability to identify matches by hierarchical orderings of data types as well as the capability to display matching percentages in regard to either a unique type match or multiple type matches. This concept is discussed later in reference to sample output files.

Throughout the revisions, various capabilities were added to the analysis code, the most rewarding being flexibility in setting weights for the values of the data types statistics, inclusion of a percent match total for the block as a whole (facilitates file identification) rather than only on a per window basis, addition of timing code for performance evaluations, and the extension to use of a two pass analysis. The benefit of a flexible weighting scheme for the data types' statistics is in the ability to weight more unique statistics higher, giving them more influence on the resulting identification process then less unique statistics. Clearly the most beneficial modification was the extension to a two pass analysis. The first pass does not deal at all with any of the pattern matching, since earlier versions had severely reduced performance due to the time consuming nature of the pattern matching process.

The challenge of the pattern matching process is that the pattern is not a simple character sequence; instead, it is an array of minimum-maximum pairs. Therefore, to match a pattern in our case meant that the statistical values for a sequence of windows had to fit into the pattern of

valid ranges. In addition, usually in a valid match, several windows matched one range before progressing to the next range. In terms of regular expressions wherein each letter represents a unique range (minimum-maximum) pair, the pattern in our data might look like a+b+c+d+. However, not only did some ranges require one or more matching windows, others required zero or more, or even a specific number of matching windows within a set range (such as in the case of matching header information which can only constitute a few consecutive windows). Thus, the use of a simple, although not extremely efficient, recursive method to implement the pattern matching was used. To the authors' knowledge, no previous work has been done with this kind of pattern matching effort, e.g. in which the pattern is so flexible and deals with ranges of numbers rather than characters.

By not including the pattern matching analysis in the first pass, the potential accuracy of the results was reduced, but only in cases in which the file being analyzed contained these data patterns. Therefore, this first pass was used in two distinct ways to improve the overall analysis: 1) it allowed an analysis of the block as a whole to be considered; thereby performing file identification on the block; and 2) only when the block was identified as some sort of textual data, could not be identified as a specific type, or contained enough differing data to warrant further analysis would the file be flagged for a secondary analysis. By only performing the secondary analysis, which incorporated pattern matching as well as textual differentiation analysis, on those memory blocks that could not be identified satisfactorily with the faster initial analysis pass performance was improved without sacrificing the accuracy of the entire analysis process.

3.6.4 Graphical Results

The results from the analysis are then utilized to generate graphs for ease of review. Graphs from the initial analysis pass are presented as histograms, such as the one given in Figure 6. In all of these histograms, a clustering technique is used to generate the two different columns

for each data type. The gray bars indicate the percentage match without regard to other matches, the overall match for that type. The blue bars indicate the percentage match when each window is only allowed to be matched to one of the data types. Therefore, even if a specific window can match more than one type, it must be selectively assigned to only one type. This is done by creating a hierarchy out of the available data types based upon their uniqueness. Several of the data types' statistical characteristics encapsulate that of others.

For example, the bmp data type can have valid averages anywhere from 0.0 to 255.0 while mp3 is only within the range from 100.0 to 150.0 and jpg data has averages between 103.0 and 148.4. Please note that mp3 is not used in the results for this paper as it is not satisfactorily refined yet. When considering just these three data types and the single average statistic, the most unique data type is jpg followed by mp3 and lastly bmp. Therefore, if a window has an average of say 102.0, it would be assigned to be of type mp3 since that is the most unique type that it matches; however, if the average were 125.6, the identified type would be jpg, and if the average were 205.7, the matching type would be identified as bmp. Therefore, each window type is assigned based upon the most unique type that it matches. From these window types, the block's overall type is identified as the most unique type that is still matched with a reasonable percentage of windows. In all cases, the overall percent match is calculated by taking the total number of matching windows and dividing by the total number of windows in the memory block. These overall percentages are represented in the resulting histograms from the initial analysis.

All of these graphs were generated by a two step process. First, a Python script was used to generate a .p file for each graph. These .p files contained all the gnuplot commands to generate the graphs utilizing the output from the analysis. In addition, a single .p file (loadFiles.p) was written with a load command for each other generated .p file. Second, gnuplot was run on the single loadFiles.p which then caused all graphs to be generated.

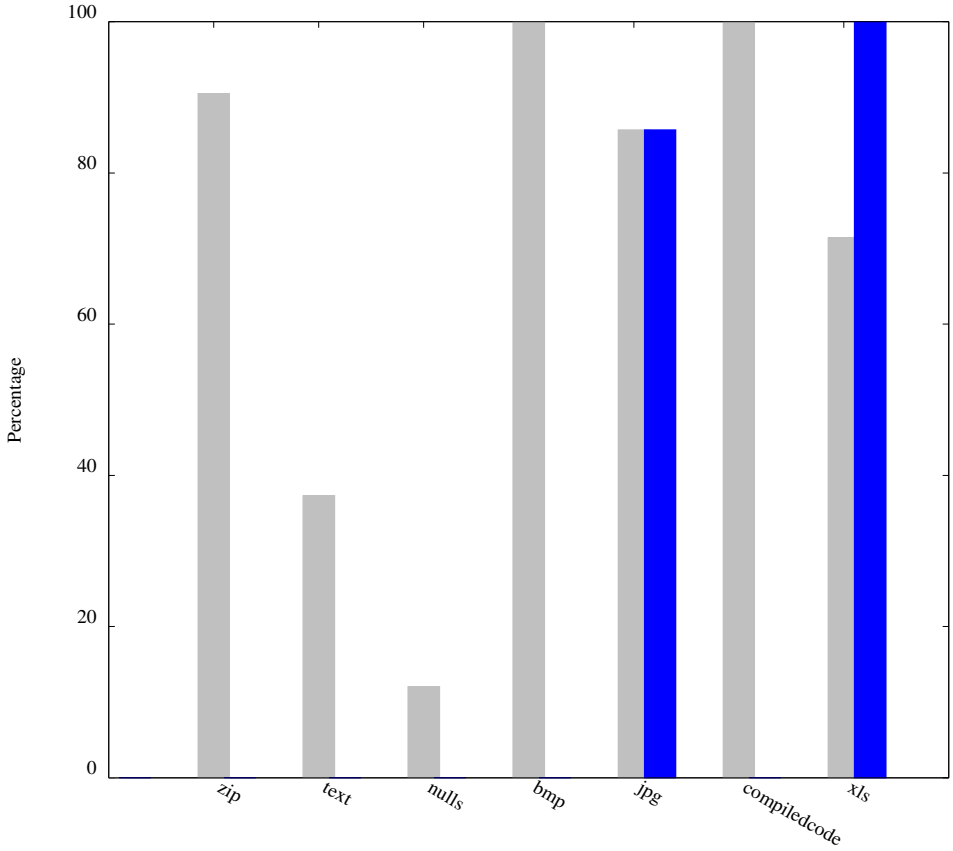

Figure 6. Sample result graph from initial analysis of Icon2.jpg.

3.6.5 Identifying Characteristic Features and the Feedback Loop

Having already touched on the process used to generate the characteristic input file containing the unique identifying characteristics for the data types, this discussion will be brief. All previously analyzed files of each data type were used to initially identify these unique characteristics. A larger test set was then used to test accuracy and provide feedback to adjust the characteristics. Adjustments were made to avoid high false positive and high false negative rates and to increase accuracy. A mixed set of files, both those used in the earlier phases as well as completely new files, were next used to generate the results presented later in this paper.

The feedback loop, however, is essential to continual improvement of the accuracy and a reduction of the false positive and false negative rates for all types as more files and memory

blocks are analyzed. It will play an even more major role when other container-like files are analyzed to identify their data content.

3.7 Comparison to Previous Work

In the techniques found in both [11] and [12], various file types are combined to increase accuracy. Specifically, Microsoft Office files were combined (acd, doc, ppt, and xls in [13] and doc, ppt, and xls in [11]) into one file type, and in [12], dll and exe files were combined into a logically equivalent type. The other primary related work, [9], does not address any of these similar types, except for Windows executables. Instead, it focuses on jpeg and zip in addition to the Windows executable. Further, no comparison is made of similarities between Windows executable types and dlls.

In our work, constructing the data type characteristic input files, we discovered one reason why the combinations of these types increased accuracy for previous methods. Table 7 displays the average values for the first nine windows (with a window size of 256) of five files of each Microsoft Office type, doc, ppt, and xls. From this table, one can see the obvious similarities among the data found within files of these types, especially in the second window. The first and second windows are applicable to a header-based analysis, since these constitute the first 512 bytes of the file. Specifically, in regard to the second window's average, to have an average of exactly 255 requires every single byte in the window to be the value 255. If even one byte in a 256 byte window is not 255, the average can be no more than 254.9961. It is believed that these similarities among the header portions of these files is the primary reason why methods relying only upon a byte frequency distribution have an improved accuracy when combining these files into one type. This is especially applicable for techniques that rely heavily on the header portion of the files.

Table 7. Average Values for Beginning Windows in MS Office Files.

Window #	Doc Average	Ppt Average	Xls Average	Total Average
0	171.6836	173.2695	187.5969	177.5167
1	255	255	255	255
2	27.15232	94.64374	27.50156	49.76587
3	23.05312	108.4579	45.6789	59.06329
4	14.43362	75.01406	41.26564	43.57111
5	17.17422	80.41328	39.88284	45.82345
6	14.50312	70.83046	40.89688	42.07682
7	11.85236	138.5547	44.1875	64.86485
8	0	102.0336	45.01876	49.01745
9	0	98.42422	50.84452	49.75625

In Table 8, all but one of the five sample dll files has an average value of 0 in windows 5-15 and all but one exe file has an average of 0 in windows 3-15. So, each of these outlier files pulled the average from an even 0.0 up to what is listed in the table. If that proportion continues with larger sets of these files, the average for those windows will continue to get closer to zero for both types. Clearly, if one is only using the byte frequency distribution, these two types of files do look incredibly similar, and this why in [12] the combination of these two file types improved accuracy. In addition, as the number of files increases, the two distributions become increasingly similar, and the types are harder to distinguish when only relying upon a byte frequency distribution. From another perspective, see Figure 7-Figure 9 that display graphs originally generated in the research for [4], although not all were used in the paper itself. Each of these graphs displays the

Table 8. Average of Beginning Windows of Dll and Exe Files.

Window #	Dll Average	Exe Average	Total Average
0	64.8765624	65.5546876	65.21563
1	15.2273436	9.4171874	12.32227
2	15.1726564	13.6406248	14.40664
3	0	2.735156	1.367578
4	18.9328124	0	9.466406
5	18.6875	0	9.34375
6	15.2007812	18.9242188	17.0625
7	16.915625	18.0328124	17.47422
8	16.9054688	18.4359376	17.6707
9	14.2164062	18.8742188	16.54531
10	15.8929688	15.0601562	15.47656
11	15.0992188	18.65	16.87461
12	18.1289062	17.7671876	17.94805
13	14.775	15.9007812	15.33789
14	16.9828124	17.3578124	17.17031
15	14.0273438	16.0195312	15.02344
16	99.5929688	100.7562498	100.1746

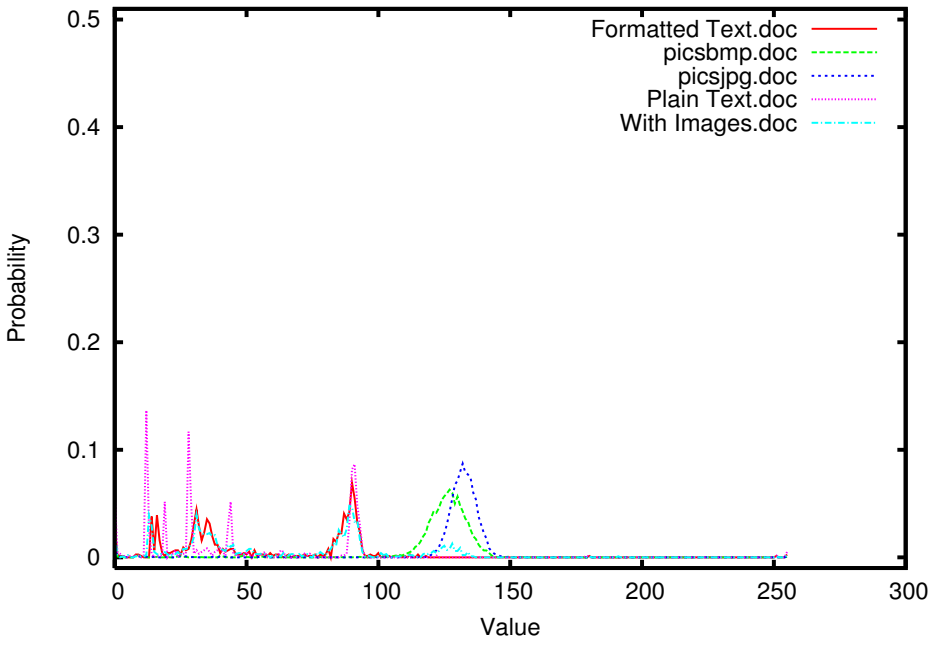

Figure 7. Byte distribution probability for MS doc files, generated originally for [4].

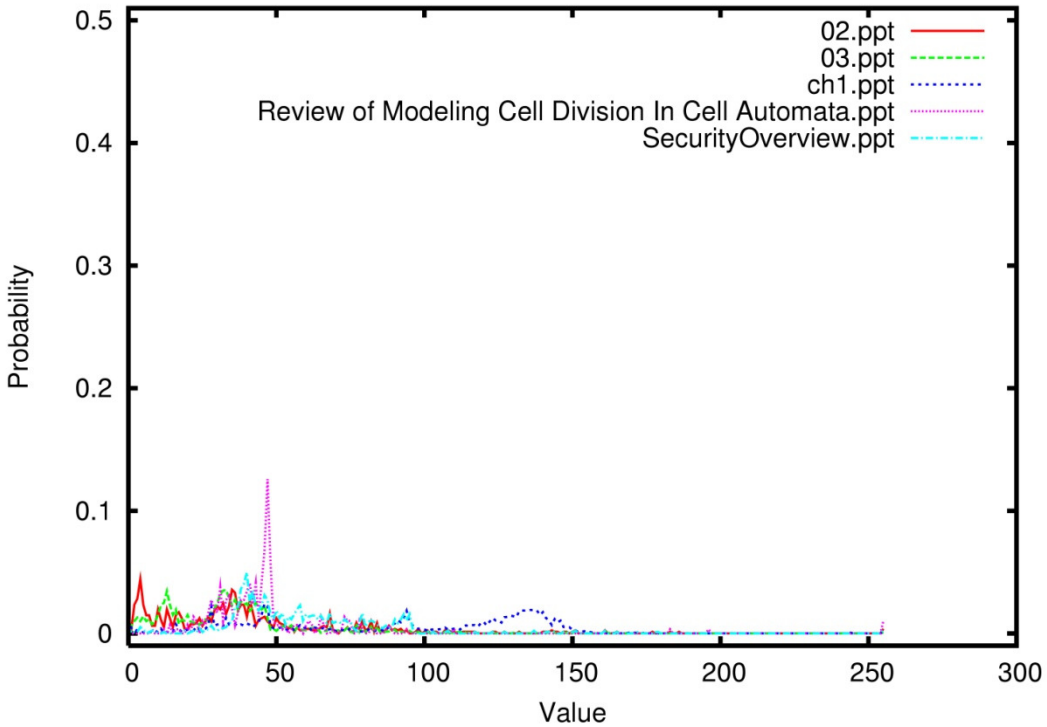

Figure 8. Byte distribution probability for MS xls files, generated originally for [4].

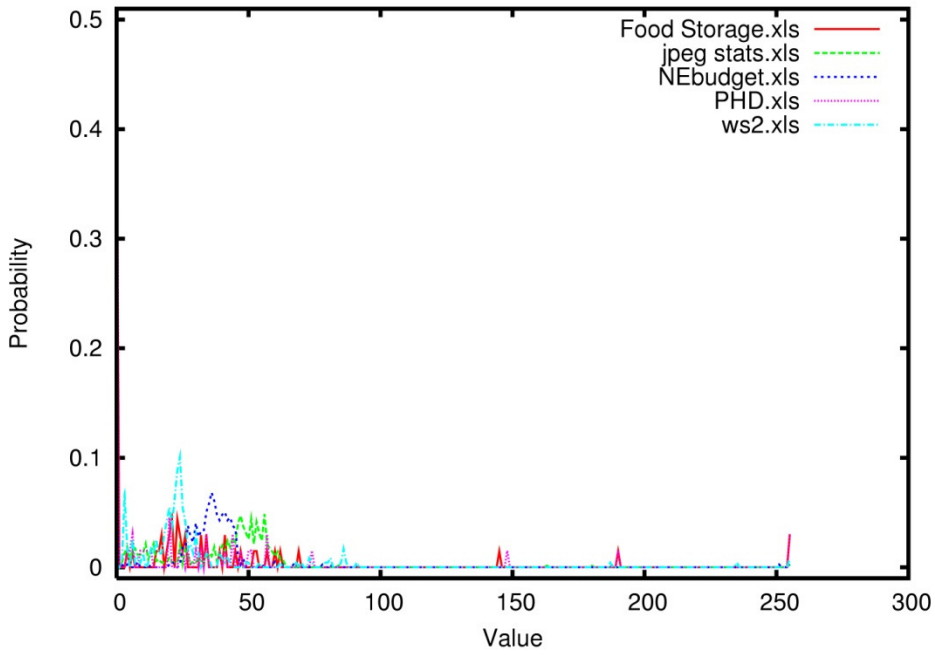

Figure 9. Byte distribution probability for MS ppt files, generated originally for [4].

distribution of averages for doc, xls, and ppt file types, respectively. Despite the variations in the individual files because of content, especially when the file contains a lot of image data and/or formatting and text, it is clear the distributions are reasonably similar. This similarity could prevent some techniques from adequately classifying the files as different file types when only utilizing a byte frequency distribution in the identification process. Consequently, for most previous techniques, the file characteristics are very similar and by combining them into one type increase a technique's accuracy. Therefore, by solely relying upon a byte frequency distribution or any other single measure to identify type, data type, or file type, challenges and problems arise. Indeed, as pointed out in [7], every measure has its drawbacks. Therefore, SÁDI takes the good from several statistical measures and incorporates all the different measurements, thereby alleviating the drawbacks of any one or two individual methods.

Therefore, by solely relying upon a byte frequency distribution or any other single measure to identify type, data type, or file type, challenges and problems arise. Indeed, as pointed out in [7], every measure has its drawbacks. Therefore, SÁDI takes the good from several statistical measures and incorporates all the different measurements, thereby alleviating the drawbacks of any one or two individual methods.

3.8 Evaluation

To evaluate the accuracy of SÁDI, we gathered 25 files of each data type, including bmp, csv, dll, exe, html jpg, txt, and xls, for a total of 8 different data types and 200 files. These files were run through the preprocessor to generate the statistical result files which were then used in the automated analysis to identify the data types contained in each. The results from the initial analysis pass are shown in Table 9. These initial results contain some false positives and false negatives. The more accurate results for xls data from both passes alleviated much of this inaccuracy caused by the patterned xls data, as shown in Table 10. The individual type matches were chosen by selecting the most unique type with a satisfactory percentage match.

To identify a positive match for a file's initial results, the type containing the highest bar in the histogram generated from the initial results is considered the matched type; we are only considering blue bars, as these are the bars containing all available data results and only allow each window to match one overall type, thus preventing the potential confusion from multiple type matches.

Thus we can see from Table 9 and Table 10 that SÁDI can accurately identify base data types. One challenge to note in reference to these results is the identification of zip data, as it is very easily confused with other compressed data such as jpg. In addition, zip data can be considered simply as other multiple types of data in a compressed format, hence, the potential to fool the identification analysis if enough data of a differing type is contained within a zip data file.

Table 9. Analysis Results from Initial Analysis with Multiple Matches (Gray Bars in Graphs).

		% Correct was calculated by taking the total correctly matched divided by the total analyzed.						
	#files	**bmp**	**text**	**Dll /exe**	**Jpg**	**Xls**	**Null**	**% Correct**
bmp	25	11		12		2		44.0%
csv	25		25					100.0%
dll	25	4		17		3	1	68.0%
exe	25	6		10		9		40.0%
html	25		25					100.0%
jpg	25	1		4	20			80.0%
txt	25	2	20	1		2		80.0%
xls	25	15	1	8		1		4.0%
Totals	200							64.5%
Updated average percentage ignoring xls.								73.1%

Table 10. Analysis Results for Xls Files from Pattern Matching Analysis.

	#files	bmp	text	Dll /exe	Jpg	Xls	Null	% Correct
xls	25	2	1	3	0	19	0	76.0%

In reference to the NULL data type, this occurs in several places for several different reasons, such as in the case of xls files. In xls files, empty or blank sheets are generally stored as several windows of NULLS. In earlier implementations, these were labeled as xls-empty-spreadsheet data; however, this is inaccurate for other types of files that use nulls to separate data.

This alternative use of NULL data is seen in the first 10 windows of doc files, in which information at the beginning of the file is separated from the main data portion with usually at least one window of NULL data. However, even in other data types, NULL can be valid value, such as in bmp images which are primarily black (for example a screenshot of a terminal window or a DOS prompt). This can cause some bmp windows to match the NULL data type if they contain only bytes of value 0. This is a rare case but still can be considered an exception to the identification of NULL as a data type. However, due to the flexible use of NULLs in data memory, they are considered a separate type. When encountered, there is no way to accurately identify what they represent without additional information.

Figure 10 does show an example graph, containing both a spreadsheet and various pictures embedded within it, from the initial pass of the analysis of a doc file. In this figure, it is obvious that the file contains mostly pictures. When compared with the results from the same document but with the images removed, the presence of the spreadsheet is clearly seen as well as the textual data, both of which were hidden by the large number of pictures (see Figure 11). In Figure 11 the overall type cannot be determined; thus, a secondary pass is needed to demonstrate the technique's ability at identifying and localizing data types.

Figure 12 shows the results from the secondary analysis. Here, the textual data is located at the beginning of the file while a spreadsheet can clearly be identified in the second half of the file. It should be pointed out that these two types are the more unique types that still match for those windows. Hence, for the apparently matching compiled code and bmp types, these are more general types and less unique and would not be identified as the actual matching type. This figure is representative of the potential of SÁDI to accurately identify component types and locate them within a file or memory block.

In the current case, the type can be located down to the set of windows involved. Further analysis would be needed to break down the windows into byte values before being able to specify starting and ending byte locations.

3.9 Additional Results and Future Work

This research generated other results not incorporated into this thesis. Other file formats and data types were studied, and several things of note were discovered. In regard to wmf files, there is a characteristic difference between wmf files and compressed wmf files. This is especially seen in graphs of the byte distribution averaged across all windows of a file. Compressed wmf had a specific pattern of peaks compared with a regular wmf file (see Figure 13 and Figure 14).

The addition of varying window sizes would allow the technique to be more useful in other situations outside of the forensic field; therefore, further study is warranted. Some potential applications of SÁDI outside of forensics include its use in virus scanners and copyright and privacy issues. Many companies have trade secrets that are not allowed to be transmitted outside of the company network; yet malicious insiders have the capability to disguise the information

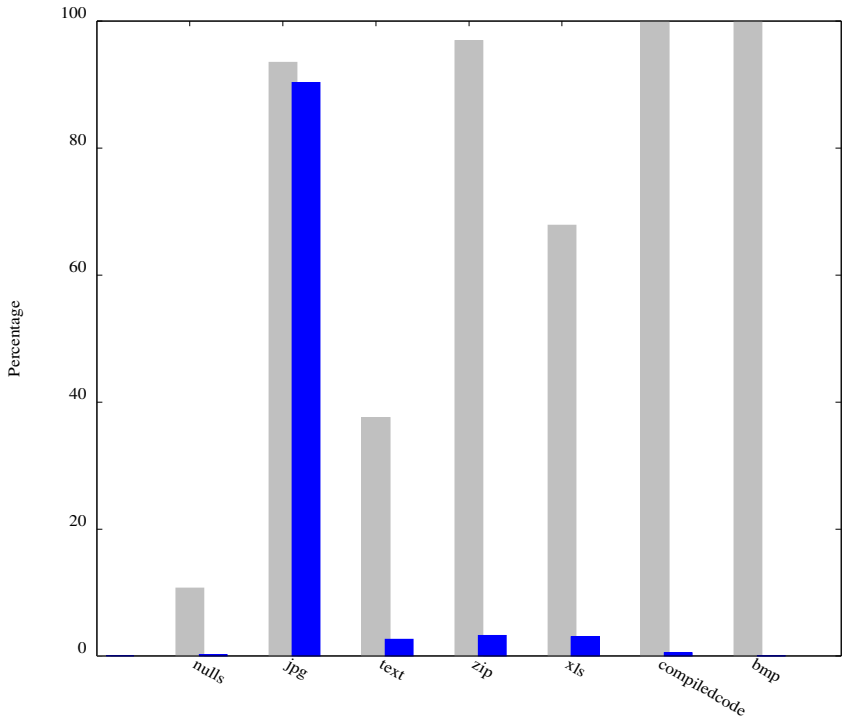

Figure 10. Results from initial analysis of CyberSecurity-2.doc, contains image data.

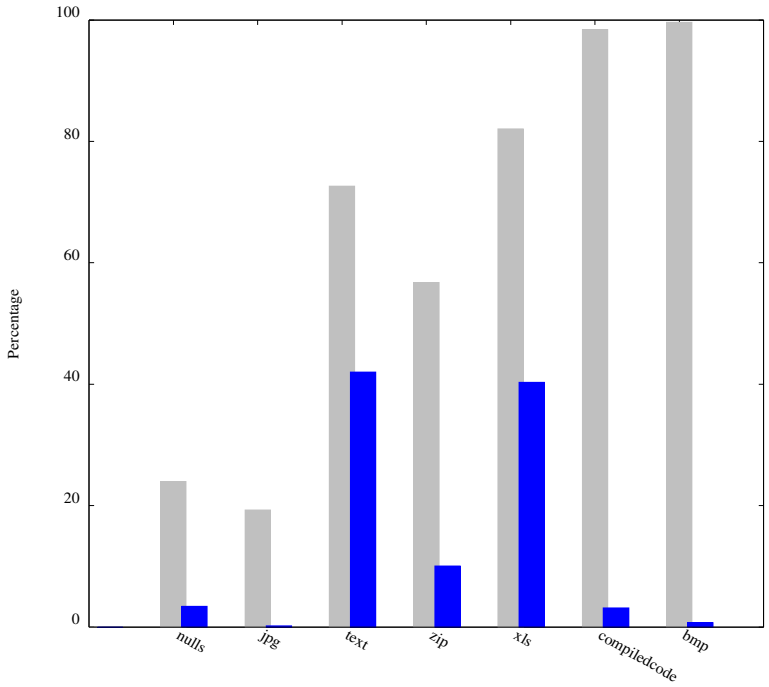

Figure 11. Results from initial analysis of CyberSecurity2-noImg.doc, with images removed.

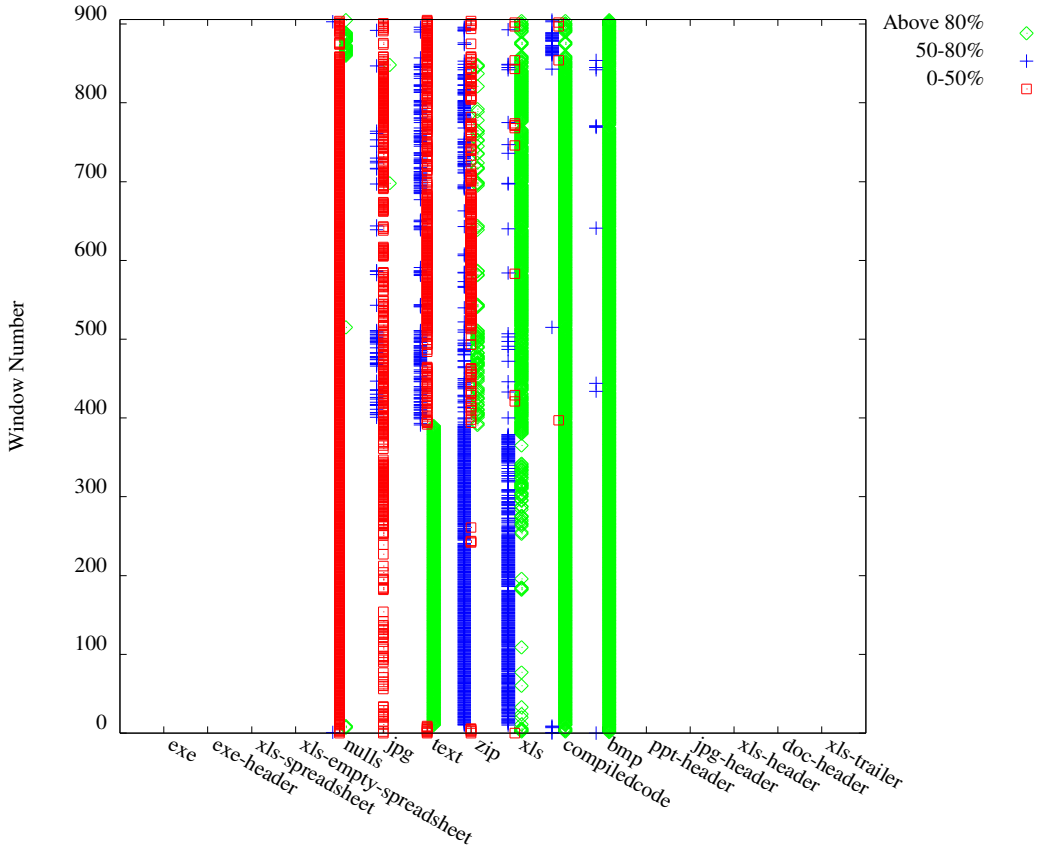

Figure 12. Results from pattern matching analysis of CyberSecurity2-noImg.doc.

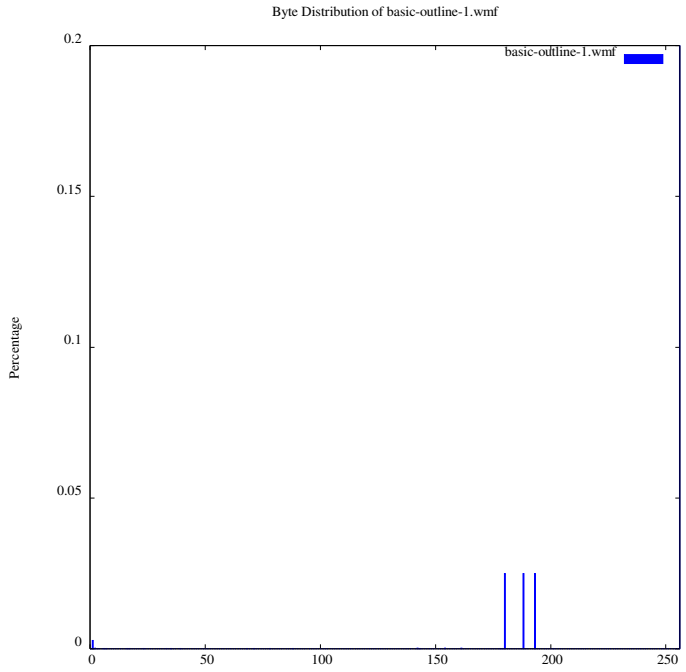

Figure 13. Byte distribution of compressed wmf file.

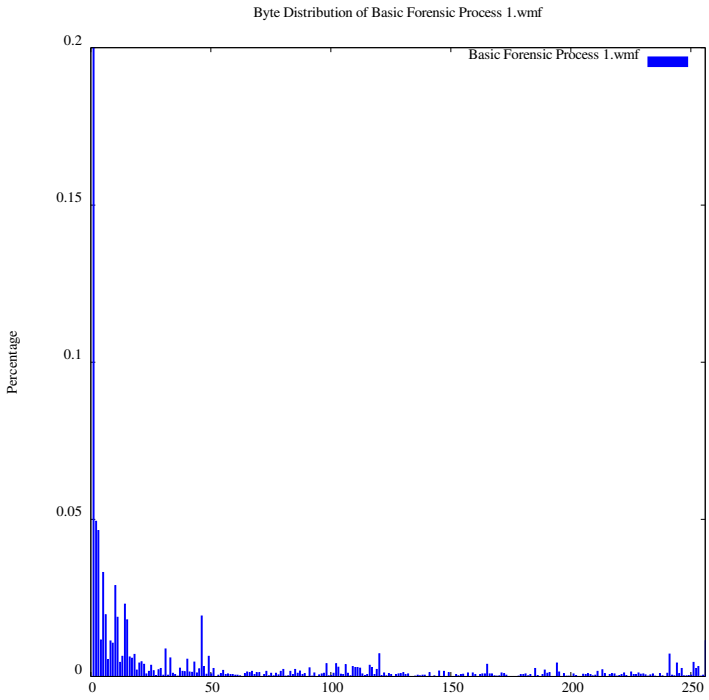

Figure 14. Byte distribution for a regular wmf file.

using steganography or other data hiding techniques to send the information outside the company. SÁDI can be useful in overcoming these malicious attempts and help in other steganalysis fields.

In the short term perspective, the most beneficial additional research currently being done to make SÁDI better is in expanding the current array of data types the analysis is able to handle, for example container-like files. Without this extension, the technique is not nearly useful enough to pursue, even in the case of forensics for general use. It would only be applicable in a subset of cases. The power of SÁDI is in the general use case and its flexibility in meeting the demands required of many different scenarios both inside and outside of the forensics field.

Other long term goals can include evaluation of overall performance with a much larger set of data types, the feasibility of its use in live forensics situations, feasibility of incorporating it into a current or future forensic tool such as Encase or the Forensics Toolkit (FTK), potential development of a GUI interface to increase usability and user friendliness, and the identification of other uses of such an algorithm (such as its potential inclusion in virus protection software, firewalls, etc.)

3.10 References

[1] Bryan Carrier, “The Sleuth Kit.” Available: http://www.sleuthkit.org/sleuthkit/ desc.php, accessed on 2007-11-30.

[2] Darwin, “*file(1)*,” Online Linux main page for the file command. Available: http://linux.die.net/ man/1/file, accessed on 2007-11-30.

[3] Encase® Enterprise. Available: http://www.guidancesoftware.com/products/ee_index.aspx, accessed 2007-11-30.

[4] Robert F. Erbacher and John Mulholland, "Identification and Localization of Data Types within Large-Scale File Systems," *Proceedings of the 2nd International Workshop on*

Systematic Approaches to Digital Forensic Engineering, Seattle, WA, April 2007, pp. 55-70. *Best Paper Award.*

[5] Dan Farmer and Wietse Venema, "The Coroners Toolkit (TCT)." Available: http://www.porcupine.org/forensics/tct.html, accessed on 2007-11-30.

[6] FTK® AccessData. Available: http://www.accessdata.com/common/pagedetail.aspx?PageCode=ftk2test, accessed 2007-11-30.

[7] Douglas J. Hickok, Daine R. Lesniak, and Michael C. Rowe, "File Type Detection Technology," in *Proceedings from the 38th Midwest Instruction and Computing Symposium*, Apr. 2005. Availabe: http://www.micsymposium.org/.

[8] Neil F. Johnson and Sushil Jajodia, "Steganalysis: The Investigation of Hidden Information," IEEE Information Technology Conference, Syracuse, New York, 1998, pp. 113-116.

[9] M. Karresand, and N. Shahmehri, "File Type Identification of Data Fragments by Their Binary Structure," *Proceedings of the IEEE Information Assurance Workshop*, West Point, NY, June 2006, pp. 140-147.

[10] M. Karresand and N. Shahmehri, "Oscar – file type identification of binary data in disk clusters and ram pages," *Proceedings of IFIP International Information Security Conference: Security and Privacy in Dynamic Environments (SEC2006)*, LNCS, 2006, pp. 413-424.

[11] Wei-Jen Li, Ke Wang, Salvatore J. Stolfo, and Benjamin Herzog, "Fileprints: Identifying file types by n-gram analysis," *Proceedings from the Sixth IEEE Systems, Man and Cybernetics Information Assurance Workshop*, June 2005, pp. 64–71.

[12] Mason McDaniel and M. Hossain Heydari, "Content Based File Type Detection Algorithms," *Proceedings of the IEEE 36th Hawaii International Conference on System Sciences (HICSS '03)*, Washington, DC, 2003, pp. 332.1.

[13] R. McKemmish, "What is Forensic Computing?," Trends and Issues in Crime and Criminal Justice, June 1999, published by the Australian Institute of Criminology. www.aic.gov.au/ , accessed 2007-11-30.

[14] Sarah Moody and Robert F. Erbacher, "Automated Identification of Data Types for Use in Computer Forensics," Poster presented at the 2007 Grace Hopper Conference, Oct. 17 Poster Session.

[15] ProDiscover® for Windows. Available: http://www.techpathways.com/ prodiscoverWindows.htm, accessed 2007-11-30.

[16] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology,* vol. 147, no. 1, pp. 195-197.

[17] Shannon I. Steinfadt, "Massively Parallel Sequence Alignment for the Bioinformatics Domain," Presentation given at the 2007 Grace Hopper Conference, Oct. 19 Session 2 – PhD Forum: Algorithms for Parallel and Distributed Computing. Available: http://www.cs.kent.edu/%7Essteinfa/files/GH07/SteinfadtPhD.ppt.html.

[18] *TrID,* published by Marco Pontello. Available: http://www.brothersoft.com/trid-20596.html, accessed on 2007-10-30.

[19] *To Associate a File Extension with a File Type,* Windows 2000 Professional Manual. Available: http://www.microsoft.com/technet/prodtechnol/ windows2000pro/proddocs/probook/prof14.mspx, accessed on 2007-11-30.

CHAPTER 4

SUMMARY AND CONCLUSION

In both preceding chapters, the results presented compare well with other recently published techniques for file identification. In fact, in most cases, the accuracy rates are higher and with fewer false positives. The technique has also been shown to handle embedded data and is applicable to file fragments. This research has produced a viable technique that is generally applicable to all sorts of digital data and produces comparable, and usually better, results. Although the technique is well on its developmental path, there are still many more data types it must be able to handle, as well as needing the incorporation of container file types.

The utilization of a multiple pass analysis provides the flexibility to handle the many and varied challenges to the vast amount of data and data types available. The ability to perform an initial analysis that can categorize the data while keeping run times low is immensely helpful for the forensics field. The additional use of a separate analysis to perform pattern matching and textual differentiation is of utmost importance when applying the technique to a real world circumstance as would be seen by forensic professionals in the field. The power and flexibility of this technique to handle data fragments and embedded data while avoiding the use of metadata and keeping the accuracy high is monumental.

This research has met the goals listed earlier, namely, the development of a data type identification technique and the analysis of the effectiveness of the developed technique. The preceding chapters describe in detail the development of the technique and the implementation used, and they provide an analysis of its effectiveness. As a measure of the effectiveness of the technique, comparisons were made to other file type identification techniques to gauge the accuracy of the novel approach used by SÁDI. Those results are again summarized in Table 11. Note the much more favorable results for all data types. Also refer to Table 12 for a comparison

of the advantages between the various techniques. Combine this with the added ability to identify embedded data, and this technique is obviously very powerful and useful for the researchers and analysts in the fields of computer and digital forensics.

Table 11. Comparison of Accuracy across Techniques.

		(Accuracy, False + Rate)						
	Avg.	**Jpg**	**Dll/ Exe**	**Zip**	**Bmp**	**Text**	**Xls**	**Doc**
SÁDI	74.2%	72%, 0%	76%, 10.9%	NR	*64%, 23.4%*	*93.3%, 0.6%*	*76%, 0%*	NR
BFA (McDaniel et al.)	27.5%	NR	NR	NR	NR	NR	NR	NR
BCA (McDaniel et al.)	45.83%	NR	NR	NR	NR	NR	NR	NR
FHT (McDaniel et. al.)	95.83%	NR	NR	NR	NR	NR	NR	NR
Extended Oscar (Karresand et al.)	NR	92.1%, 20.6%	12.6%, 1.9%	*46-80%, 11-37%*	NR	NR	NR	NR
File Header Identification (Li et al.)	NR	*100%, NR*	NR	NR	NR	NR	73.6% NR	83.9% NR
Centroid based classification (Li et al.)	*99.6%*	*100%, NR*	*100%, NR*	NR	NR	NR	NR	*98.9% NR*

It is clear from this comparison that the overall applicability of SÁDI as a forensics technique is high. While it is still not as refined as it eventually will be, it has many advantages lacking in other techniques. This novel technique provides the computer science field with a more accurate and more widely applicable analysis technique, and it will only get better with future research.

Table 12. Technique Comparison.

	Advantages	**Disadvantages**
SÁDI	Content-based, handles embedded and fragmented data	Can still be refined more, some types still less accurate
BFA (McDaniel et. al.)	Fast	Inaccurate
BCA (McDaniel et. al.)	Better than BFA	Inaccurate, Not fast
FHT (McDaniel et. al.)	High accuracy	Non-content-based
Extended Oscar (Karresand et. al)	Some w/ good accuracy	Not accurate for all types, high false positive
File Header Identification (Li et. al)	Mostly high accuracy	Non-content based, unknown False positive rates
Centroid based classification (Li et. al.)	High accuracy	Only looks at first 200 bytes (non-content based), unknown false positive rates