SÁDI – Statistical Analysis for Data type Identification

Sarah J. Moody and Robert F. Erbacher
Department of Computer Science, UMC 4205
Utah State University
Logan, UT 84322
s.j.m@aggiemail.usu.edu and Robert.Erbacher@usu.edu

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 being examined. Typical file type identification is based upon file extension or magic keys. These typical techniques fail in many typical forensic analysis scenarios such as needing to deal with embedded data, such as with Microsoft Word files, or file fragments.

The SÁDI (Statistical Analysis Data Identification) 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. The development of SÁDI provides the capability to identify what digitally stored data actually represents and will also allow for the selective extraction of portions of the data for additional investigation; i.e., in the case of embedded data. Thus, our research provides a more effective type identification technique that does not fail on file fragments, embedded data types, or with obfuscated data.

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 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 are focusing on the location and identification of relevant information embedded or appended into other innocuous appearing files; there are many techniques that can be applied for the hiding of 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 that 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 will be focusing on the identification of data on hard drives, the discussed technique is applicable across the board to all forms of digitally stored data. This new technique is being referred to as SÁDI (Statistical Analysis Data

Identification). Of critical importance is the concept that SÁDI attempts to identify data types as opposed to file types. This becomes 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 becomes the location of relevant data types. For instance, when attempting to locate child pornography on a hard drive we must consider locating

- 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.

The technique used by SÁDI involves taking a block of memory, i.e., a single file, and performing 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. Research problem

There have been many techniques developed which attempt to identify file types; these techniques will be expounded upon in Section 3. 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 will not open a file that has been renamed with an incorrect extension and some virus scanners will not scan files unless they have the executable extension [13].

UNIX systems utilize the *file* command 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 attempt 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]. The reliance on header information prevents magic numbers and file extensions from being useful when dealing with file fragments or obfuscated files and data. 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 that can be found 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 that someone actually wants to hide 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 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 which might or might not be somewhere on a hard drive [1][3][5][6][16].

3. Background and related work

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

Other previous work in the area of type identification is found in [13]. In this work, three different techniques were used to identify file types. Note however that although these detection algorithms attempt to utilize file content to perform the identification, the overarching goal of these techniques was to identify the file type regardless of what data or data types might be contained or embedded therein. Consequently, the techniques by McDaniel et al. [13] would not be able to identify embedded data accurately. McDaniel et al.'s first technique used a file fingerprint created from the byte frequency distributions of files of the same type. Files were then identified depending upon their match with a type's fingerprint. This algorithm had an average accuracy rate of 27.5%. The second algorithm reported by McDaniel et al. is the Byte Frequency Cross-correlation Algorithm (BCA). This algorithm is similar except it extends the process to look at correlations between byte values. The authors reported its accuracy as 45.83% and that it was a much slower method than the BFA method. The third algorithm was the File Header/Trailer Algorithm (FHT). This algorithm just 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%, it reduces the problem to a non-content based approach that only relies upon headers and trailers; i.e. it will fail for many situations of interest to forensic analysts.

Karresand et al. [10] 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 in an earlier paper of their research [11]. A centroid, created from byte frequency distributions, byte averages, and the standard deviation for byte values, was used to identify file fragments. To match a type, the file fragment had 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 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 generally 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. While the technique was effective at identifying jpeg files, this is misleading as 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 raises issues as to the overall usefulness of the technique. These Oscar-based methods still focused on identifying file type and hence will have the same drawbacks as are mentioned for the methods developed by McDaniel et al.

Hall et al. [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 will aid in differentiating the type of data contained within the file, such as compressed versus uncompressed data.

4. Research project goals, approach and uniqueness

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

- Allow for the more accurate identification of file and data types
- Allow for the identification of obfuscated data and covert channels
- Allow for the locating and extraction of hidden data

As implied by the name ‘Statistical Analysis Data Identification’, the SÁDI method uses results from a statistical analysis to perform data type identification. While a full range of statistical techniques were examined, 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 will show 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 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 in the case of html, txt, and csv data. All of these data type are strictly textual data but each haa unique characteristics in their distributions that allows for differentiation between 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.

5. Methodology

5.1. Applied statistical techniques

From the previous work by Erbacher et al. [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 will show 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 with 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.

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 are then stored in a configuration file in association with each of the individual data types. The identified range characteristics are shown in Table 1.

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 of allowing for different window sizes comes from the fact that each window size will produce 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. We have primarily used a window size of 256 bytes. This size was chosen due to the benefit of the data not being obfuscated from too large a window and 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. These configuration values are valid for a 256 byte window.

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 will be identifying unique probability characteristics; namely a high appearance of “,” in csv and {“<”,”>”} in html.

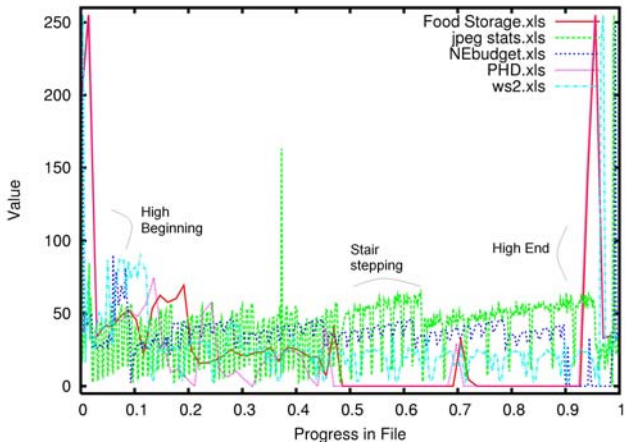

Figure 1*:* Averages values for xls data. These data streams are particularly unique with the stair-step pattern. Also of note are the peaks at the beginning and end of the data streams. Finally, an embedded graph is visible at the 38% mark for one of the data streams.

Some of the 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 that is unique to only the average statistical results for spreadsheet data [4]; this is exemplified in Figure 1.

This unusual pattern in the xls average statistical data engendered a need for a second pass within the analysis code. The first pass focuses solely on applying statistical analysis techniques

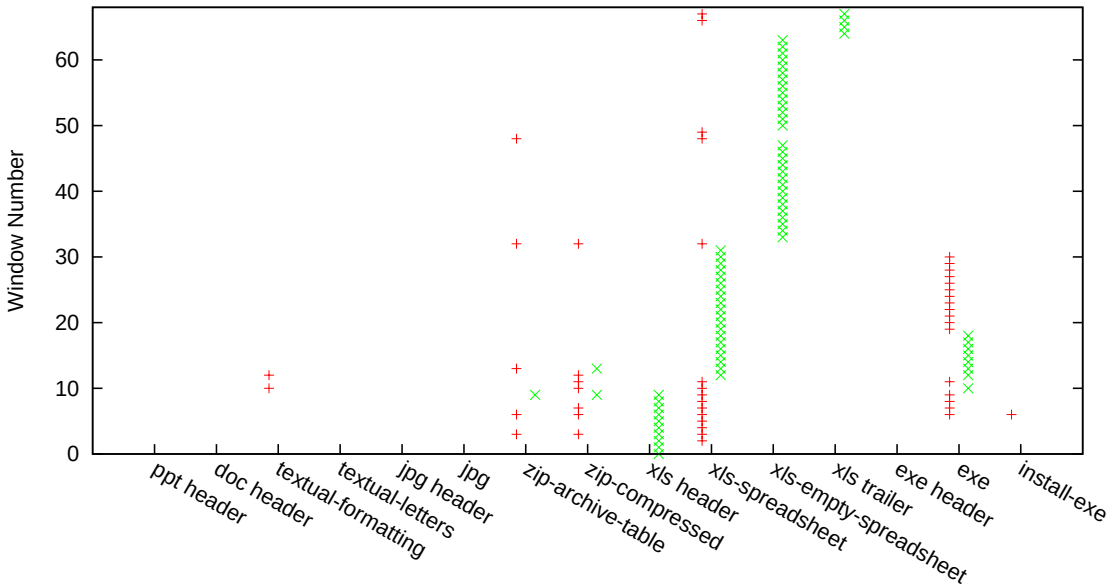

Figure 2: Secondary Analysis of 'Food_Storage.xls'. The characteristic pattern typical of xls spreadsheets can be clearly seen. This pattern remains identifiable even with larger numbers of conflicting matches in the first pass.

and attempts to identify data blocks that match the statistical structure of known data types. The second pass performs an analysis that attempts to identify unusual patterns in these computed statistics, as seen with Microsoft Excel spreadsheet files; this is exemplified in Figure 2. By looking at this figure, one can also see 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.

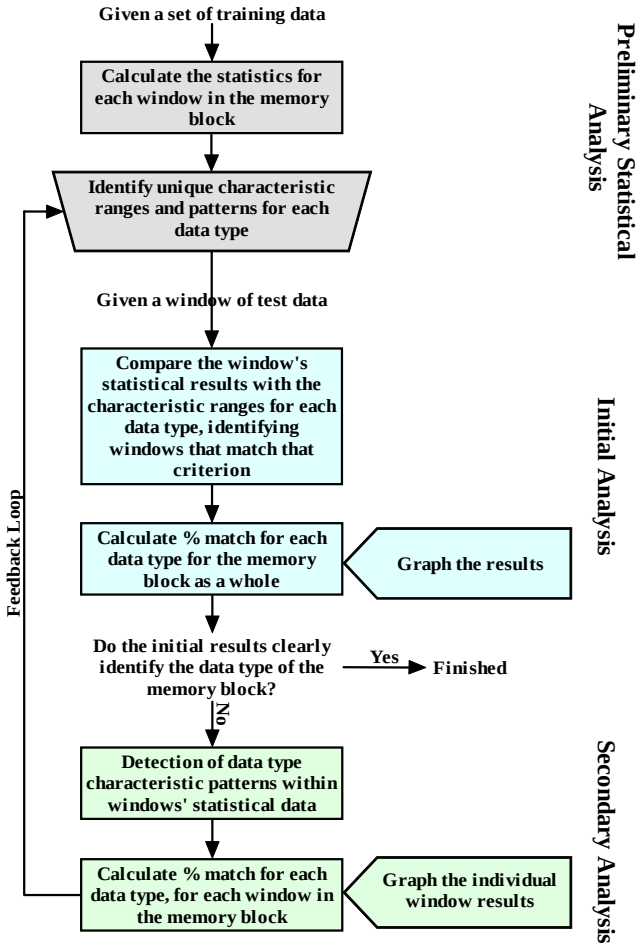

Figure 3: Analysis Process. This task flow diagram exemplifies the process of computing the initial statistical matching parameters and applying those parameters to test data for data type identification.

6. SÁDI implementation

SÁDI's task flow [15], exemplified in Figure 3, is fundamentally based on a continuous feedback loop paradigm to allow for the continuous improvement and refinement of the data

differentiation parameters. Such improvement 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.

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.

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 using 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 and hence they will have more effect on the resulting identification process than less unique statistics.

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.

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; 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. During testing, adjustments were made to avoid high false positive rates, high false negative rates, and to increase accuracy.

Table 2: Analysis Results – only including data generated from the initial analysis. Data windows were required to have a minimum 92% match with a data type to be associated with that data type.

	# 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%

The files were then 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 that in turn alters the false positive and false negative rates. The ideal minimum percentage varies depending upon type. It can also be of note that although xls is included here a secondary pass has not yet been done and therefore much of this data will be incorrectly identified.

The results for xls data when a secondary pass is taken into account are presented in Table 3. The most influential reason for the terrible results for xls data when only considering the initial pass was 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.

Table 3: Secondary analysis results for xls data. The incorporation of the pattern-based analysis greatly improves the results over the single pass only analysis.

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

The results of the analysis of the 25-bmp files are shown in Table 4. When considering the dissimilarity of bmp and textual data, it was very odd to see so many files identified as textual data.

Table 4: Secondary analysis results for bmp data. The identification of matches with textual data was key to improving the accuracy of bmp matches.

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

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 fact that the files were not actually 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.

```
#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, essentially embedded into a source code file. This data likely acts as an icon.

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

In the consideration of jpg data, although most jpg files are correctly matched, there were several which matched the wider range found within the dll and exe data types. This percentage can be 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 that will fall just outside of the jpg data characteristic range and hence will fail to match jpg data as accurately as other types of binary data, such as the dll and exe data types.

Table 5 gives the results with a varying minimum percentage; by varying the minimum percentage required for a match, we obtain the most promising accuracy 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 is 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%, then 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 is 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.

Table 5: Varied minimum percentages. This table identifies the changes (improvements) in accuracy achieved by letting the minimum percentages float to optimal values independently for each data type.

	# 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%

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, some preliminary results have been obtained to verify this differentiation will be possible for most files. Each data type has a 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. 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, for csv files, more 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 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 another example of the benefit of the technique to identify embedded data since the presence of larger amounts of non-html data caused these four files to match plain text data rather than html.

Table 6: Percentage of matched files from textual distribution analysis. Here we applied distribution analysis in order to differentiate csv, html, and txt files. This relies on the presence of unique frequently occurring bytes in each of the types, such as comma in csv files.

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

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 meta data.

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, down to 0% for jpg and xls data. One reason for the higher false negative percentages is that for some files they match a more unique type or a slightly less unique type. For example, most dll and exe data that was incorrectly identified was instead identified as bmp data that has wider ranges and hence allows for data with more varying values. Also, most incorrectly identified bmp data alternatively is identified as dll and exe data because it happens to fall within the smaller ranges of the characteristics of the dll and exe data types.

Even in cases where 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]; this embedded or appended data would be identified using the SÁDI technique simply because of its existence and its differing statistical structure from the rest of the file.

Bmp data presents its own 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 of this could be a bmp that is simply a black background. This would be stored as all zeros and hence would be identified as NULL data. This is such an extreme example because NULL is at the opposite end of the spectrum of data types because of the single value range for all statistics compared with bmp that has the broadest range for all statistics.

9. Future work

There could be benefit from further studying the effect of varying window sizes, both in identifying an optimal window size and in identifying applications outside of forensics. For instance, SÁDI could have beneficial use in virus scanners and in the identification of copyright or privacy violations. Similarly, simply identifying covert channels or the 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, identifying the data types embedded therein. This 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 at all. This will require applying SÁDI to fragments of data rather than to entire files and identify dynamically when the data type appears to be 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.

10. References

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

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

[3] *Encase® Enterprise*, online information found at: 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), http://www.porcupine.org/forensics/tct.html, accessed on 2007-11-30.

[6] *FTK®* AccessData, online information found at: 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," http://www.mantech.com/cfia2/SlidingWindowMeasurementforFileTypeIndentification.pdf.

[8] 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, 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," *Proceedings of the IEEE Information Assurance Workshop*, West Point, NY, June 2006, pp. 140-147.

[11] 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.

[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. www.aic.gov.au/

[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.

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

[17] G.J. Simmons, "The Prisoner's Problem and the Subliminal Channel," In *Proceedings 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, http://www.brothersoft.com/trid-20596.html, accessed on 2007-10-30.

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