Content Based File Type Detection Algorithms

Mason McDaniel and M. Hossain Heydari[1]
Computer Science Department
James Madison University
Harrisonburg, VA 22802

Abstract

Identifying the true type of a computer file can be a difficult problem. Previous methods of file type recognition include fixed file extensions, fixed "magic numbers" stored with the files, and proprietary descriptive file wrappers. All of these methods have significant limitations. This paper proposes algorithms for automatically generating "fingerprints" of file types based on a set of known input files, then using the fingerprints to recognize the true type of unknown files based on their content, rather than metadata associated with them. Recognition is performed by three different algorithms based on: byte frequency analysis, byte frequency cross-correlation analysis, and file header/trailer analysis. Tests were run to measure the accuracy of these algorithms. The accuracy varied from 23% to 96% depending upon which algorithm was used.

These algorithms could be used by virus scanning packages, firewalls, intrusion detection systems, forensic analyses of computer hard drives, web browsers, or any other program that needs to identify the types of files for proper operation. File type detection is also important to the operating systems for correct identification and handling of files regardless of file extension.

I. Introduction

Computers use a tremendous array of file formats today. All types of files are frequently transmitted through intranets and the Internet. Currently, operating systems, firewalls, and intrusion detection systems have very few methods for determining the true type of a file. Perhaps the most common method used is to identify the type of a file by the file's extension. This is an extremely unreliable method, as any user or application can change a file's name and extension at any time. As a result, some users are able to conceal files from system administrators simply by renaming them to a filename with a different extension. While this doesn't conceal the existence of a file, it can conceal the nature of a file and can prevent it from being opened by the operating system. In addition, many virus-scanning packages default to only scanning executable files. These packages may miss any viruses contained within executable files that had non-executable file extensions. This could introduce vulnerabilities into a network, even if it contained virus protection.

The other common method of identifying file types is through manual definition of file recognition rules. This is an extremely time-consuming process, whereby an individual examines a file type specification, if one is available, and identifies consistent features of a file type that can be used as a unique identifier of that type. In the absence of a specification, the individual must manually examine a number of files looking for common features that can be used to identify the file type. Not only is this time-consuming, but it can require an individual with a highly technical background that is capable of doing a hexadecimal analysis of files.

Manual rule definition is the method used by many Unix-based operating systems, as well as tools used in forensic analysis of computer disks during investigations. Regardless of the investigating authority, automated file type recognition is a critical part of this sort of computer forensic analysis.

An efficient, automated algorithm to perform this kind of file type recognition would be of tremendous benefit to organizations needing to perform forensic

[1] This research is supported in part by a grant from the Virginia Commonwealth Technology Research Fund and supported in part by a grant from the Department of Defense Information Assurance Scholarship program.

analyses of computer hard drives. It could also be used by virus protection software, intrusion detection systems, firewalls, web browsers, and security downgrading packages to identify the true nature of programs passing through the protected systems. Finally, this kind of algorithm could be of use to the operating systems themselves to allow for correct identification and handling of files regardless of file extension.

This paper describes an attempt to extend the concept of frequency analysis and apply it to the general case of generating a characteristic "fingerprint" for different computer file types, and subsequently using the fingerprint to identify file types based upon their characteristic signatures. The process could be almost entirely automated, and would not be affected if a user changed a file name or extension.

I.1 Previous work

To date, there have been relatively few methods for identifying the type of a file. One of the most commonly used methods is the use of file extensions. Microsoft's operating systems use this method almost exclusively. They come preset with associations between file extensions and file types. If different associations are desired, they must be manually reconfigured by the user [7]. As mentioned above, this approach introduces many security vulnerabilities. A user can change the extension of a file at any time, rendering the operating system unable to identify it. They can also change the file extension associations to fool the operating system into handling files in an inappropriate manner, such as trying to execute a text file.

Another approach is that taken by many Unix-based operating systems. These make use of a "magic number" which consists of the first 16 bits of each file. A file, such as */etc/magic* then associates magic numbers with file types [9]. This approach has a number of drawbacks as well. The magic numbers must be predefined before the files are generated, and are then built into the files themselves. This makes it very difficult to change them over time, since a change might interfere with the proper operation of many files that were generated using the old magic number. Furthermore, not all file types use magic numbers. The scheme was initially intended to assist with the proper handling of executable and binary formats. With only 16 bits allocated, a number of extensions had to be introduced over time, such as using the "#!" magic number to identify a command to execute on the rest of the file [8].

Another approach is to define a proprietary file format that encapsulates other files and provides information regarding their type. One example of this approach is the Standard File Format (SAF) developed by the Advanced Missile Signature Center (AMSC) [6]. There are many down sides to this approach. The specification must be written defining how to encapsulate and identify each file format. An individual or external system must identify the type of the file before it can be correctly encapsulated in the standard format in the correct manner. The most significant problem, however, is that this type of file can only be used within the small proprietary system that recognizes the "standard" format. The files cannot be exported to external systems such as the Internet without removing the encapsulation, and thus negating its benefit.

II. Algorithms

The design goals for the proposed file recognition algorithm are as follows:

- Accuracy – The algorithm should be as accurate as possible at identifying file types.
- Automatic generation of file type fingerprints.
- Small fingerprint files – The fingerprint file sizes should be minimized.
- Speed – Comparisons should be as fast as possible for a given fingerprint file size.
- Flexibility – The algorithm should provide a customizable tradeoff between speed and accuracy.
- Independence from file size.

These design goals can be achieved by implementing the three algorithms described in this paper, each of which could be selected independently, or used together for increased accuracy. Due to space limitation, detailed explanation of these results is available in [1].

II.1 Byte frequency analysis (BFA) algorithm

A computer file is a collection of bytes, which correspond to eight-bit numbers capable of representing numeric values from 0 to 255 inclusive. By counting the number of occurrences of each byte value in a file, a frequency distribution can be obtained. Many file types have consistent patterns to their frequency distributions, providing information useful for identifying the type of unknown files.

Figure II.1 and Figure II.2 show the frequency distributions for a typical RichText (RTF) and a Graphics Interchange Format (GIF) file, respectively. Many file types likewise have characteristic patterns that can be used to differentiate them from other file formats.

This section describes the methods used to build the byte frequency distribution of individual files and

to construct a fingerprint representative of the file type.

II.1.1 Building the byte frequency distribution

The first step in building a byte frequency fingerprint is to count the number of occurrences of each byte value for a single input file. This is done by constructing an array of size 256 (indexed 0 to 255), and initializing all array locations to zero. For each byte in the file, the appropriate element of the array is incremented by one. Once the number of occurrences of each byte value is obtained, each element in the array is divided by the number of occurrences of the most frequent byte value. This normalizes the array to frequencies in the range of 0 to 1, inclusive. This normalization step prevents one very large file from skewing the file type fingerprint. Rather, each input file is provided equal weight regardless of size.

Some file types have some byte values that occur much more frequently than any other. If this happens, the normalized frequency distribution may show a large spike at the common values. Figure II.3 shows the frequency distribution for an executable file that demonstrates this. The file has large regions filled with the byte value zero. The resulting graph has a large spike at byte value zero, with insufficient detail to determine patterns in the remaining byte value ranges.

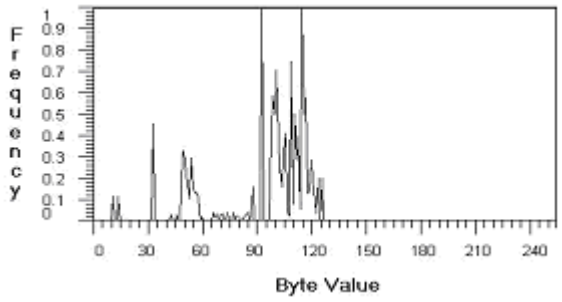

Figure II.1 - Byte frequency distributions for two RTF files.

A way to solve this problem would be to pass the frequency distribution through a companding function to emphasize the lower values. Common companding functions, such as the A-law and μ-law companding functions used in telecommunications [2], can be roughly approximated by the following function, which can be very rapidly computed.

The same file shown in Figure II.3, after being passed through this equation, produces the frequency distribution shown in Figure II.5. This graph shows more of the detail across all byte frequencies, and therefore may allow for more accurate comparisons. Experimental results indicated that $\beta = 1.5$ is the optimal β value for the most accurate file type recognition [1]. The optimal value of β is defined as the value that produces the greatest difference between the fingerprint with the highest frequency score and the fingerprint with the second-highest frequency score.

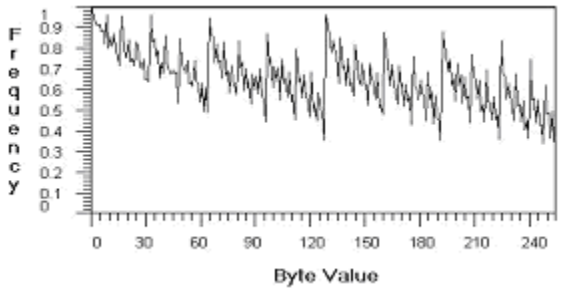

Figure II.2 - Byte frequency distributions for two GIF files.

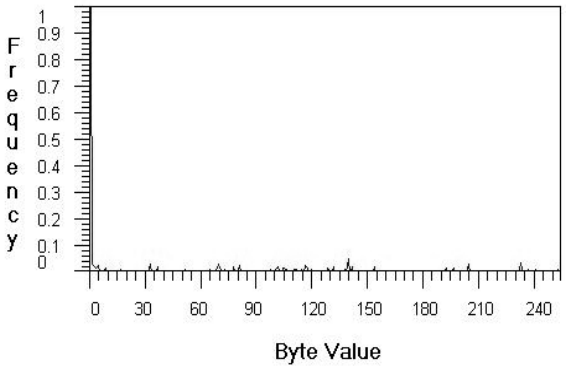

Figure II.3 - Frequency distribution for a sample executable file.

The companding function results in a frequency distribution that is still normalized to 1. This is true since the most frequent byte value was normalized to 1, and the companding function with an input value of 1 results in an output value of 1.

II.1.2 Combining frequency distributions into a fingerprint

A fingerprint is generated by averaging the results of multiple files of a common file type into a single fingerprint file that is representative of the file type as a whole. To add a new file's frequency distribution to a fingerprint we use the following simple averaging equation, where NFPS is the new fingerprint score, OFPS is the old fingerprint score,

PNF is the previous number of files, and NFS is the new file score.

$$NFPS = \frac{(OFPS \times PNF) + NFS}{PNF + 1}$$

Aside from the byte frequency distributions, there is another related piece of information that can be used to refine the comparisons. The frequencies of some byte values are very consistent between files of some file types, while other byte values vary widely in frequency. For example, note that almost all of the data in the files shown in Figure II.1 lie between byte values 32 and 126, corresponding to printable characters in the lower ASCII range. This is characteristic of the RichText format. On the other hand, the data within the byte value range corresponding to the ASCII English alphanumeric characters varies widely from file to file, depending upon the contents of the file.

This suggests that a "correlation strength" between the same byte values in different files can be measured, and used as part of the fingerprint for the byte frequency analysis. In other words, if a byte value always occurs with a regular frequency for a given file type, then this is an important feature of the file type, and is useful in file type identification.

A correlation factor can be calculated by comparing each file to the frequency scores in the fingerprint. The correlation factors can then be combined into an overall correlation strength score for each byte value of the frequency distribution.

The correlation factor of each byte value for an input file is calculated by taking the difference between that byte value's frequency score from the input file and the frequency score from the fingerprint. If the difference between the two frequency scores is very small, then the correlation strength should increase toward 1. If the difference is large, then the correlation strength should decrease toward 0. Therefore, if a byte value always occurs with exactly the same frequency, the correlation strength should be 1. If the byte value occurs with widely varying frequencies in the input files, then the correlation strength should be nearly 0.

A function that would provide more tolerance for small variations and less tolerance for larger variations is a bell curve with a peak magnitude of 1 and the peak located at 0 on the horizontal axis. The general equation for this type of bell curve is:

$$F(x) = e^{\left(\frac{-x^2}{2\sigma^2}\right)}$$

where *F(x)* is the correlation factor and *x* is the difference between the new byte value frequency and the average byte value frequency in the fingerprint.

Experimental results indicated that $\sigma = 0.0375$ is the optimal σ value for the most accurate file type recognition [1].

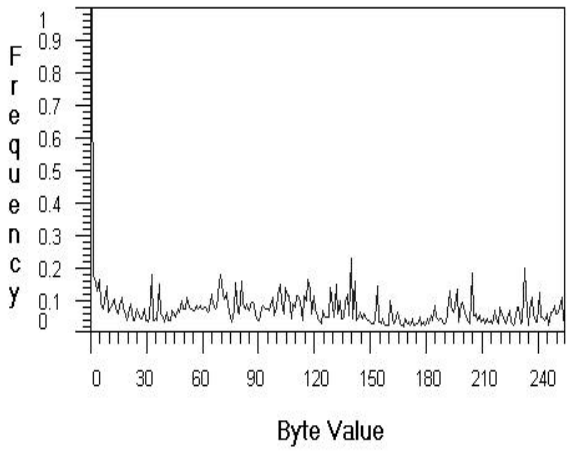

Figure II.5 - Frequency distribution for a the figure II.3 file after passing through the companding function.

Once the input file's correlation factor for each byte value is obtained, these values need to be combined with the correlation strengths in the fingerprint. This is accomplished by using the following simple averaging equation, which directly parallels the method used to calculate the frequency distribution scores, where NCS is the new correlation strength, OCS is the old correlation strength, PNF is the previous number of files, and NCF is tne new correlation factor.

$$NCS = \frac{(OCS \times PNF) + NCF}{PNF + 1}$$

II.1.3 Comparing a single file to a fingerprint

When identifying a file using the byte frequency analysis algorithm (BFA):

- Compute a score for each fingerprint identifying how closely the unknown file matches the frequency distribution in the fingerprint. The score is generated by comparing each byte value frequency from the unknown file with the corresponding byte value frequency from the fingerprint. As the difference between these values decreases, the score should increase toward 1. As the difference increases, the score should decrease toward 0.
- Compute an "assurance level" for each fingerprint, indicating how much confidence can be placed on the score. The file type's byte frequency correlation strengths are used to generate a numeric rating for the assurance level. This is because a file type with a characteristic byte frequency

distribution will have high correlation strengths for many byte values.

- Compare the unknown file's byte frequency distribution to the byte frequency scores and the associated correlation strengths stored in each file type fingerprint and pick the best match.

Figure II.7 shows the byte frequency distribution for the HTML fingerprint, with the frequency scores shown by a solid line and the correlation strengths shown by a dotted line. Figure II.8 shows the byte frequency distribution scores and correlation strengths for the ZIP fingerprint.

Using this scheme, the HTML file format would have a high assurance level for the byte frequency, since many byte values have high correlation strengths, whereas the ZIP file format would have a low assurance level for the byte frequency, suggesting that perhaps other algorithms should be used to improve accuracy for this type.

II.2 Byte frequency cross-correlation (BFC) algorithm

While BFA algorithm compares overall byte frequency distributions, other characteristics of the frequency distributions are not addressed. One example can be seen in Figure II.7. There are two equal-sized spikes in the solid frequency scores at byte values 60 and 62, which correspond to the ASCII characters "<" and ">" respectively. Since these two characters are used as a matched set to denote HTML tags within the files, they normally occur with nearly identical frequencies.

This relationship, or cross-correlation, between byte value frequencies can be measured and scored as well, strengthening the identification process. This section describes the methods used to build the byte frequency cross-correlations of individual files, to construct a fingerprint representative of the file type, and to compare an unknown file to a file type fingerprint, obtaining a numeric score.

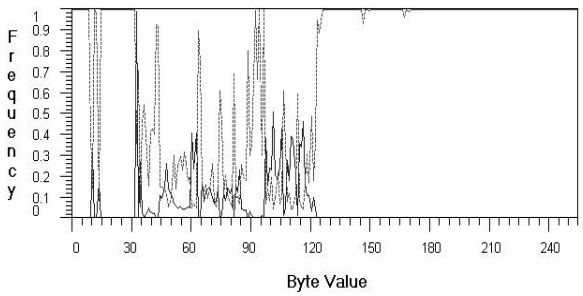

Figure II.7 - Byte frequency distribution with correlation strength for HTML fingerprint.

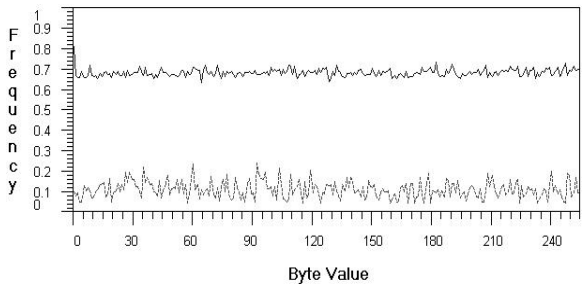

Figure II.8 - Byte frequency distribution with correlation strength for ZIP fingerprint.

II.2.1 Building the byte frequency cross-correlation

There are two key pieces of information that need to be calculated concerning the byte frequency cross-correlation analysis: the average difference in frequency between all byte pairs and a correlation strength similar to the BFA algorithm. Byte value pairs that have very consistent frequency relationships across files, such as byte values 60 and 62 in HTML files, as mentioned above, will have a high correlation strength score. Byte value pairs that have little or no relationship will have a low correlation strength score.

In order to characterize the relationships between byte value frequencies, a two-dimensional 256×256 cross-correlation array is built (byte values are between 0 and 255), with indices ranging from 0 to 255 in each dimension.

Note that if byte value i is being compared to byte value j, then array entry (i, j) contains the frequency difference between byte values i and j while array entry (j, i) contains the negative of the corresponding (i, j) location. Hence, half of the array contains redundant information and storing both of them is unnecessary. We use the lower half of the array to store the correlation strengths of each byte value pair. So now if byte value i is being compared to byte value j, then array entry (i, j) contains the frequency difference between byte values i and j while array entry (j, i) contains the correlation strength for the byte pair. Furthermore, a byte value will always have an average frequency difference of 0 and a correlation strength of 1 with itself, so the main diagonal of the array can be used to store any other information that is needed for the comparisons. We use the first entry of the main diagonal (0, 0) to store the number of files that have been used to compute the fingerprint.

Calculating the difference between the frequencies of two bytes with values i and j involves simply subtracting the frequency score of byte value i from the frequency of byte value j. Since byte value

frequencies were normalized, with a range of 0 to 1, this results in a number with a possible range of –1 to 1. A score of –1 indicates that the frequency of byte value i was much greater than the frequency of byte value j. A score of 1 indicates that the frequency of byte value i was much less than the frequency of byte value j. A score of 0 indicates that there was no difference between the frequencies of the two byte values.

II.2.2 Combining cross-correlations into a fingerprint

Once the frequency differences between all byte-value pairs for an input file have been calculated, the new fingerprint can be calculated using the following equation, similar to the one used in BFA algorithm, where NFPD is the new fingerprint difference, OFPD is the old fingerprint difference, NFD is the new frequency difference, and PNF is the previous number of files.

$$NFPD = \frac{(OFPD \times PNF) + NFD}{PNF + 1}$$

A correlation factor can be calculated for each byte value pair, by comparing the frequency differences in the input file to the frequency differences in the fingerprint. The correlation factors can then be combined with the scores already in the fingerprint to form an updated correlation strength score for each byte value pair. As more files are added to construct the fingerprint, the correlation strengths more accurately reflect the file type.

If at least one file has been previously added into a fingerprint, then the correlation factor for each byte value pair is calculated by subtracting the pair's frequency difference from the new file and the same pair's average frequency difference from the fingerprint. This results in a new overall difference between the new file and the fingerprint. If this overall difference is very small, then the correlation strength should increase toward 1. If the difference is large, then the correlation strength should decrease toward 0. New correlation strength is calculated using the same equations as the BFA algorithm.

After the average frequency differences and correlation strengths for each byte value pair of the new input file have been updated in the fingerprint, the Number of Files field is incremented by 1 to indicate the addition of the new file.

It is interesting to compare the frequency distribution graphs of BFA algorithm to the byte frequency cross-correlation plots generated from BFC algorithm. Figure II.7 shows the frequency distribution for the HTML file format, and Figure II.9 shows a graphical plot of the HTML fingerprint cross-correlation array. Note that there are ranges of byte values in the frequency distribution that never occurred in any files (they appear with 0 frequency.) These regions appear in the cross-correlation plot as mid-tone gray regions of 0 frequency difference. Furthermore, the intersection of 60 on the vertical axis with 62 (corresponding to the ASCII values for the "<" and ">") shows a dark dot representing a correlation strength of 1, as expected.

Looking at a graphical plot of a GIF fingerprint cross-correlation array (not shown to save space), a sawtooth pattern in the frequency distributions is a characteristic feature of the GIF file type, and it manifests in the cross-correlation plot as a subtle grid pattern in the frequency difference region.

II.2.3 Comparing a single file to a fingerprint

When identifying a file using the byte frequency cross-correlation algorithm (BFC):

- Compute a score, similar to BFA, for each fingerprint identifying how closely the unknown file matches the fingerprint. The score is generated by comparing the frequency difference for each byte value pair from the unknown file with the average frequency difference for the corresponding byte value pair from the fingerprint. As the difference between these values decreases, the score should increase toward 1. As the difference increases, the score should decrease toward 0.
- Compute the assurance level, indicating how much confidence can be placed on the score. File types that have characteristic cross-correlation patterns should have high assurance levels, others should have low assurance levels. As with the BFA algorithm, the correlation strengths are used to generate a numeric rating for the assurance level. The higher the assurance level, the more weight can be placed on the score for that fingerprint.
- Compare the unknown file's cross-correlation array to the cross-correlation scores and correlation strengths stored in each file type fingerprint and pick the best match.

II.3 File header/trailer (FHT) algorithm

BFA and BFC make use of byte value frequencies to characterize and identify file types. While these characteristics can effectively identify many file types, some do not have easily identifiable patterns. To address this, the file headers and file trailers can be analyzed and used to strengthen the recognition of many file types. The file headers and trailers are patterns of bytes that appear in a fixed location at the beginning and end of a file

respectively. These can be used to dramatically increase the recognition ability on file types that do not have strong byte frequency characteristics.

This section describes the methods used to build the header and trailer profiles of individual files, to combine the ratings from multiple files into a fingerprint for the file type, and to compare an unknown file to a file type fingerprint, obtaining a numeric score.

II.3.1 Building the header and trailer profiles

The first step in building header and trailer profiles is to decide how many bytes from the beginning and end of the file will be analyzed. If H is the number of file header bytes to analyze, and T is the number of trailer bytes to analyze, then two two-dimensional arrays are built, one of dimensions H × 256 and the other of dimensions T × 256. For each byte position in the file header (trailer), all 256 byte values can be independently scored based upon the frequency with which the byte value occurs at the corresponding byte position.

An individual file's header array is initially set to 0. For each byte position in the header, from byte 0 (the first byte in the file) to byte H – 1, the array entry corresponding to the value of the byte is filled with a correlation strength of 1 (each row has 255 zeros and a single one). The only exception occurs when an input file is shorter than the header or trailer lengths. In this case, the fields in the missing byte position rows will be filled with the value -1 to signify no data. (Note that if a file length is greater than the header and trailer lengths, but less then the sum of the two lengths, then the header and trailer regions will overlap.) The trailer array is similarly constructed.

II.3.2 Combining header and trailer Profiles into a fingerprint

A fingerprint is constructed by averaging the correlation strength values from each file into the fingerprint using the following equation, which is similar to the ones used in BFA and BFC algorithms, where NFPA is the new fingerprint array entry, OFPA is the old fingerprint array entry, PNF is the previous number of files, and NA is the new array entry.

$$NFPA = \frac{(OFPA \times PNF) + NA}{PNF + 1}$$

A sample graphical plot of the file header fingerprint array is shown in Figure II.11 (please note the very light markings on the figure) for the GIF file type. The first few bytes of the GIF header show high correlation strengths (represented by dark marks,) indicating that this type has a strongly characteristic file header. The specification for the GIF format states that the files shall all begin with the text string "GIF87a" for an earlier version of the format, or "GIF89a" for a later version. Further inspection of Figure II.11 shows that rows 0-3 and 5 have correlation strengths of 1 for the byte value positions corresponding to ASCII values of "GIF8" and "a". In row four, byte values 55 and 57 (ASCII values for "7" and "9") both show correlation strengths roughly balanced. This indicates that approximately equal numbers of files of each version of the GIF format were loaded into the fingerprint. Beyond byte position six, there is a much broader distribution of byte values, resulting in lower correlation strengths and lighter marks on the plot.

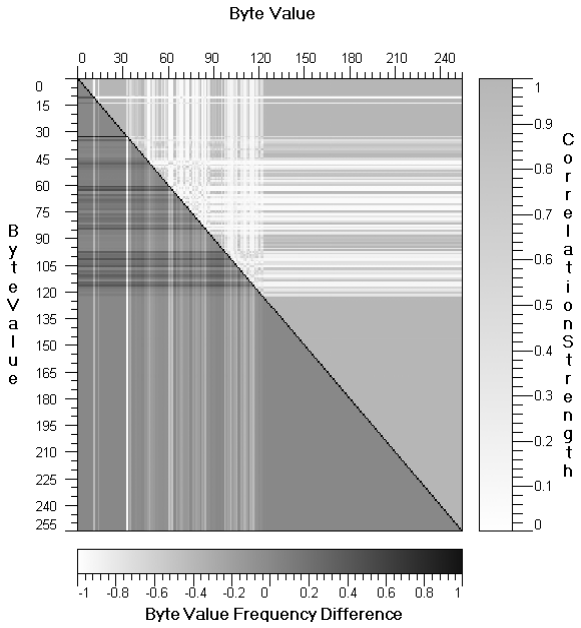

Figure II.9 - Byte frequency cross-correlation plot for the HTML file type

Figure II.12 shows a very similar plot of the file trailer for the MPEG file type fingerprint, where the end of the file is represented by byte position 0 at the bottom of the plot. This plot shows a broad distribution of byte values (resulting in extremely faint marks) up until four bytes from the end of the file. These final four bytes show a characteristic pattern similar to the pattern described above for the GIF file header.

Note that for file types that do not have a characteristic file header or trailer, the corresponding plots would appear essentially empty, with many

scattered dots with very low correlation strengths (therefore producing almost white dots.)

II.3.3 Comparing a single file to a fingerprint

When identifying a file using the file headers and trailers algorithm (FHT):

- Construct the file header and trailer arrays for the unknown file as described above.
- Use the following equation to generate the score for the file header and trailer, where C is the correlation strength for the byte value extracted from the input file for each byte position, and G is the correlation strength of the byte value in the fingerprint array for the corresponding byte position. This equation produces an average of the correlation strengths of each byte value from the input file, weighted by the greatest correlation strength at each byte position. This results in placing greatest weight on those byte positions with a strong correlation, indicating that they are part of a characteristic file header or trailer, and placing much less weight (ideally no weight) on values where the file type does not have consistent values.

$$\overline{S} = \frac{C_1 G_1 + C_2 G_2 + \ldots + C_n G_n}{G_1 + G_2 + \ldots + G_n}$$

- The assurance level for the file header and file trailer is simply set equal to the overall maximum correlation strength in the header and trailer arrays, respectively. This is different from the approach used in BFA and BFC algorithms, where the average of all correlation strengths was used.
- Compare the unknown file's header/trailer information to the cross-correlation scores and correlation strengths stored in each file type fingerprint and pick the best match.

The GIF file header provides a clear example. The first four byte positions each have a correlation strength of 1 for a single byte. This indicates that all input files of the GIF file type had the same byte values for these positions. If an unknown file has different bytes in these positions, it is a very strong indicator that it is not a GIF file. On the other hand, if the unknown file has a differing byte value at position 20, which shows a very low correlation strength, this offers no real information about whether the unknown file is a GIF file or not since there are no bytes in this position with a high correlation strength.

Setting the assurance level equal to the maximum correlation strength allows even a few bytes with very high correlation strength, such as those in the GIF file format to provide a strong indication of file type. Therefore even a few bytes can produce a strong influence in recognition. On the other hand, if a file type has no consistent file header or trailer, the maximum correlation strength will be very low. This means little weight will be placed on the header or trailer with the low assurance level.

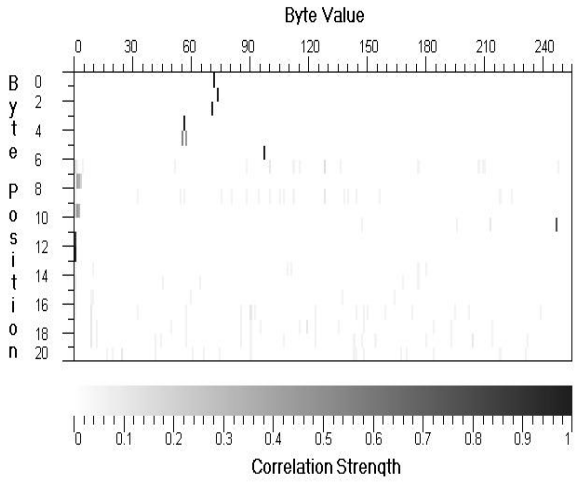

Figure II.11 - File header plot for the GIF file fingerprint

The optimal header (trailer) length is the value that results in the highest average level of differentiation across all file types. Our experimental results indicate that the optimum header and trailer length for file type identification is five [1].

III. Experimental results

In this section we describe our experimental results, using each of the 3 above-mentioned algorithms to identify file types. Thirty file type fingerprints are constructed and used for this test. To run the accuracy test, four test files are selected for each file type, resulting in a total library of 120 files. Combining the file types ACD, DOC, PPT, and XLS into a single OLE DOC fingerprint, using the average of the four type fingerprints, resulted in a more accurate type recognition for BFC and FHT algorithms and a slight decrease for BFA. Following shows the accuracy test results for each of the 3 algorithms. Type recognition reports were generated for each of the 120 test files:

- Figure III.1 shows the resulting file type identification grid for BFA algorithm. BFA's accuracy is only 27.50%. This is better than purely random guesses but not accurate enough for practical use. We should note that the accuracy of this algorithm increases to 29.17% if separate

fingerprints are used for ACD, DOC, PPT, and XLS files, not a significant improvement.

- Figure III.2 shows the resulting file type identification grid for BFC algorithm. BFC's accuracy is only 45.83%. This is a significant improvement over BFA, but not accurate enough for practical use.
- Figure III.3 shows the resulting file type identification grid for FHT algorithm. FHT's accuracy is 95.83%. This is a significant improvement over BFA and BFC and may be accurate enough for some fault-tolerant applications. We should note that using separate fingerprints for ACD, DOC, PPT, and XLS files decreases FHT's accuracy to 85%, most of the errors occurred between the ACD, DOC, PPT, and XLS file type identification.

	File 1	File 2	File 3	File 4	Score
3TF	3TF	3TF	3TF	3TF	4
ACD	3TF	3TF	OLE	OLE	2
AVI	3TF	CRP	RM	3TF	0
BMP	3TF	3TF	FNT	3TF	0
CAT	CAT	CAT	CAT	CAT	4
CRP	CRP	CRP	CRP	CRP	4
DOC	WPD	3TF	3TF	3TF	0
EXE	FNT	3TF	3TF	CRP	0
FNT	3TF	3TF	3TF	GIF	0
GIF	RM	ZIP	RM	RM	0
GZ	MP3	TAR	ZIP	CRP	0
HTML	RTF	TXT	CAT	CAT	0
JPG	JPG	GZ	MP3	MP3	1
MDL	CAT	CAT	CAT	CAT	0
MOV	CRP	CRP	RM	RM	0
MP3	MP3	GZ	MP3	MP3	3
MPEG	CRP	CRP	MP3	CRP	0
PDF	CRP	PDF	EXE	TXT	1
PPT	3TF	3TF	3TF	3TF	0
PS	TXT	TXT	CAT	TXT	0
RTF	RTF	RTF	RTF	CAT	3
RM	RM	CRP	RM	CRP	2
RPM	GZ	CRP	GZ	GZ	0
TAR	CRP	CAT	TXT	ZIP	0
TXT	TXT	CAT	TXT	TXT	3
TTF	TTF	TTF	TTF	WPD	3
WAV	CAT	TXT	FNT	3TF	0
WPD	3TF	3TF	WPD	TXT	1
XLS	WPD	FNT	3TF	3TF	0
ZIP	GIF	ZIP	ZIP	GIF	2
				TOTAL CORRECT:	33
				TOTAL FILES:	120
				Accuracy:	27.50%

Figure III.1 Identified type of each test file with a combined OLE DOC fingerprint, BFA algorithm.

IV. Conclusions and future work

The BFA algorithm proved to be the fastest of the three algorithms. An unknown file takes an average of 0.010 seconds to compare to 25 fingerprints and identify the closest match (All times were taken on an 800 MHz Pentium III laptop with 512 MB RAM). Because of its poor accuracy, BFA would be of a very limited use. The calculations performed in this algorithm, though, are used as the basis for the BFC.

The BFC algorithm proved to be by far the slowest and only moderately more accurate than BFA. An unknown file takes an average of 1.19 seconds to compare to 25 fingerprints and identify the closest match. This algorithm offers slightly improved accuracy over BFA but its accuracy is still too low to be of practical use in most applications.

The FHT algorithm provides the best combination of speed and accuracy. An unknown file takes an average of 0.015 seconds to compare to 25 fingerprints and identify the closest match, which is almost as fast as BFA. This algorithm had by far the highest accuracy at 95.83% for a combined OLE DOC fingerprint and 85% for separate ACD, DOC, PPT, and XLS fingerprints.

	File 1	File 2	File 3	File 4	Score
3TF	3TF	3TF	3TF	3TF	4
ACD	OLE	OLE	OLE	OLE	4
AVI	OLE	CAT	OLE	3TF	0
BMP	3TF	3TF	TTF	3TF	0
CAT	CAT	CAT	CAT	CAT	4
CRP	CRP	CRP	CRP	CRP	4
DOC	OLE	OLE	OLE	OLE	4
EXE	OLE	OLE	OLE	OLE	0
FNT	3TF	3TF	3TF	3TF	0
GIF	GIF	GIF	3TF	3TF	2
GZ	3TF	3TF	3TF	3TF	0
HTML	RTF	TXT	CAT	CAT	0
JPG	3TF	3TF	3TF	3TF	0
MDL	MDL	CAT	MDL	MDL	3
MOV	GIF	GIF	3TF	3TF	0
MP3	3TF	MP3	MP3	MP3	3
MPEG	3TF	MPEG	3TF	OLE	1
PDF	PDF	PDF	PDF	TXT	3
PPT	OLE	OLE	OLE	OLE	4
PS	TXT	TXT	CAT	TXT	0
RTF	RTF	TXT	RTF	TXT	2
RM	OLE	RM	RM	RM	3
RPM	RPM	OLE	RPM	RPM	3
TAR	OLE	CAT	3TF	RPM	0
TXT	TXT	CAT	TXT	TXT	3
TTF	TTF	TTF	OLE	OLE	2
WAV	TXT	TXT	TXT	TXT	0
WPD	WPD	WPD	WPD	TXT	3
XLS	3TF	OLE	OLE	OLE	3
ZIP	3TF	3TF	3TF	3TF	0
				TOTAL CORRECT:	55
				TOTAL FILES:	120
				Accuracy:	45.83%

Figure III.2 Identified type of each test file with a combined OLE DOC fingerprint, using BFC algorithm

Although FHT performs considerably better than the other algorithms, 95.83% accuracy, there would be a tradeoff in only using this algorithm. Not all file types have consistent file headers or trailers and would most likely not be correctly recognized if only FHT were used. BFA and BFC could help with the identification of the few files FHT was unable to identify. We are working on developing algorithms that use a combination of these techniques to improve type identification accuracy.

Other improvements could be investigated in the methods used to compute the score for BFA and BFC. Perhaps more sophisticated curve-matching (or other) algorithms could be tested to see if they would improve the accuracy of these options. Improvements could, also, be made in computing the score and correlation strength for header and trailer analysis as well. The header and trailer tests both showed degradation in performance as longer header and trailer lengths were used. It should be possible to modify the scoring algorithm to prevent this degradation

Overall, the algorithm proved effective at correctly identifying the file types of files based solely upon the content of the files. FHT algorithm identified executable files with 100 percent accuracy. This option could therefore be of use to virus scanning packages that are configured to only scan executable files. FHT is extremely fast and for header and trailer lengths of five bytes, the total fingerprint size for an executable fingerprint would be only 53 bytes. The algorithm could possibly be of use to cryptanalysts as well. It could be used to automatically differentiate between real data and "random" encrypted traffic.

A number of other systems could also benefit from the described file recognition approach. These include forensic analysis systems, firewalls configured to block transfers of certain file types, web browsers, and security downgrading systems. Further refinements would be required, however, before the algorithm would be fast enough or accurate enough to be used by an operating system that must reliably deal with a large number of varied file types.

Bibliography

[1] Mason McDaniel, Automatic File Type Detection Algorithm, Masters Thesis, James Madison University, 2001.

[2] Bellamy, John, Digital Telephony, Second Edition, John Wiley & Sons, Inc., New York, New York, 1991, pp 110-119.

[3] The Binary Structure of OLE Compound Documents, available online from: http://user.cs.tu-berlin.de/~schwartz/pmh/guide.html

[4] Kyler, Ken, Understanding OLE Documents, Delphi Developer's Journal, September 1998, available online from: http://www.kyler.com/pubs/ddj9894.html

[5] Stallings, William, Cryptography and Network Security, Prentice Hall, upper Saddle River, New Jersey, 1999, p. 32.

[6] *The Advanced Missile Signature Center Standard File Format*, available online from: http://fileformat.virtualave.net/archive/saf.zip

[7] *To Associate a File Extension with a File Type*, Windows 2000 Professional Documentation, available online from: http://www.microsoft.com/WINDOWS2000/en/professional/help/win_fcab_reg_filetype.htm

[8] *Why do some scripts start with #!*, Chip Rosenthal, available online from: http://baserv/uci/kun.nl/unix-faq.html

[9] */etc/magic Help File*, available online from: http://qdn.qnx.com/support/docs/qnx4/utils/m/magic.html

	File 1	File 2	File 3	File 4	Score
3TF	3TF	3TF	3TF	3TF	4
ACD	OLE	OLE	OLE	OLE	4
AVI	AVI	AVI	AVI	AVI	4
BMP	BMP	BMP	BMP	BMP	4
CAT	CAT	CAT	CAT	CAT	4
CRP	CRP	CRP	CRP	CRP	4
DOC	OLE	OLE	OLE	OLE	4
EXE	EXE	EXE	EXE	EXE	4
FNT	FNT	FNT	FNT	RPM	3
GIF	GIF	GIF	GIF	GIF	4
GZ	GZ	GZ	GZ	GZ	4
HTML	HTML	HTML	HTML	HTML	4
JPG	JPG	JPG	JPG	JPG	4
MDL	MDL	CAT	MDL	MDL	3
MOV	MOV	MOV	MOV	MOV	4
MP3	RM	MP3	MP3	MP3	3
MPEG	MPEG	MPEG	MPEG	MPEG	4
PDF	PDF	PDF	PDF	PDF	4
PPT	OLE	OLE	OLE	OLE	4
PS	PS	PS	PS	PS	4
RTF	RTF	RTF	RTF	RTF	4
RM	RM	RM	RM	RM	4
RPM	RPM	RPM	RPM	RPM	4
TAR	TAR	TAR	TAR	TAR	4
TXT	TXT	TXT	TXT	TXT	4
TTF	TTF	TTF	TTF	TTF	4
WAV	AVI	WAV	WAV	AVI	2
WPD	WPD	WPD	WPD	WPD	4
XLS	OLE	OLE	OLE	OLE	4
ZIP	ZIP	ZIP	ZIP	ZIP	4
				TOTAL CORRECT:	115
				TOTAL FILES:	120
				Accuracy:	95.83%

Figure III.3 Identified type of each test file with a combined OLE DOC fingerprint, using FHT algorithm.