# Fileprint Analysis for Malware Detection[1]

Salvatore J. Stolfo, Ke Wang, Wei-Jen Li
Columbia University

## Abstract

Malcode can be easily hidden in document files and embedded in application executables. We demonstrate this opportunity of stealthy malcode insertion in several experiments using a standard COTS Anti-Virus (AV) scanner. In the case of zero-day malicious exploit code, signature-based AV scanners would fail to detect such malcode even if the scanner knew where to look. We propose the use of statistical binary content analysis of files in order to detect suspicious anomalous file segments that may suggest infection by malcode. Experiments are performed to determine whether the approach of n-gram analysis may provide useful evidence of an infected file that would subsequently be subjected to further scrutiny. Our goal is to develop an efficient means of detecting suspect infected files for application to online network communication or scanning a large store of collected information, such as a data warehouse of shared documents.

## 1. Introduction

Attackers have used a variety of ways of embedding malicious code in otherwise normal appearing files to infect systems. Viruses that attach themselves to system files, or normal appearing media files, are nothing new. State-of-the-art COTS products scan and apply signature analysis to detect these known malware. For various performance optimization reasons, however, COTS Anti-Virus (AV) scanners may not perform a deep scan of all files in order to detect known malcodes that may have been embedded in an arbitrary file location. Other means of stealth to avoid detection are well known. Various self-encryption or code obfuscation techniques may be used to avoid detection simply making the content of malcode unavailable for inspection by an AV scanner. In the case of new "zero day" malicious exploit code, signature-based AV scanners would fail to detect such malcode even if the scanner had access to the content and knew where to look.

In this paper we explore the use of statistical content analysis of files in order to detect anomalous file segments that may suggest infection by malcode. Our goal is to develop an efficient means of detecting suspect infected files for application to online network communication such as file sharing or media streaming, or scanning a large store of collected information, such as a data warehouse of acquired content.

The first contribution of this paper is the observation that anti-virus systems can be easily deceived even given a signature for the hidden malcode. In our experiments, we simply inserted known malcode into normal PDF and DOC files. Although all these malcodes can be captured by the anti-virus system if they appear as stand alone files, quite a few poisoned PDF and DOC files carrying the malcode inside were not flagged by a popular COTS AV scanner. Furthermore, some of these infected files were successfully opened by Adobe Reader and MS Word. Thus, the file formats and application logic provides a ready made means of stealthily infecting a host with innocent appearing files. This implies sandboxing techniques to determine whether files are infected or fail in their execution would not be effective detectors in all cases.

We also note that an existing known vulnerability of certain windows executables [23] remains available for malware insertion while avoiding detection. We demonstrate a simple case of embedding malcode into the block padding portion of MS WINWORD.EXE creating an infected application that operates correctly as the original executable. This provides a stealthy means of

[1] This work was partially supported by a grant from ARDA under a contract with Batelle, Pacific Northwest Labs, and by a Department of Homeland Security Grant, HSARPA #0421001/H-SB04.2-002.

hiding malcode in otherwise normal appearing applications for later use by a crafty attacker. Such simple evasion of the anti-virus scanners is an obvious critical concern. Although a simple solution would be to perform a deep scan of all files, it is unknown whether this might create too many false positives by a traditional signature-based scanner (that may incorrectly deem portions of a large media file as infected). Certainly, new malcode would still go undetected even under a deep file scan. The problems with signature-based AV systems failing to detect new zero-day exploits are well known; a new generation of anomaly detection systems aimed at detecting zero-day exploits are beginning to appear in commercial products.

Towards this end, we also introduce an alternative means of detecting malware using an anomaly detector applied to the static binary content of a file. The conjecture is that we may model different types of files to produce a model of what all files of that type should look like. Any significant deviation form this model may indicate the file is infected with embedded malcode. Suspect files identified using this technique may then be more deeply analyzed using a variety of techniques under investigation by other researchers (e.g., [9,16.18].)

In our prior work [11, 19, 20], we demonstrated an efficient statistical n-gram method to analyze the binary contents of network packets and files. This work followed our earlier work on applying machine learning techniques applied to binary content to detect malicious email attachments [15]. The method trains n-gram models from a collection of input data, and uses these models to test whether other data is similar to the training data, or sufficiently different to be deemed an anomaly. The method allows for each file type to be represented by a compact representation we call a *Fileprint*, an effective means of representing all members of the same file type by a set of statistical n-gram models. Using this technique, we can successfully classify files into different types, or validate the declared type of a file, according to their content, instead of using the file extension only or searching for embedded "magic numbers" [11] (that may be spoofed).

We do not presume to replace other detection techniques, but rather to augment approaches with perhaps new and useful evidence to detect suspicious files. Under severe time constraints, such as real-time testing of network file share flows, or inspection of large amounts of newly acquired media, the technique may be useful in prioritizing files that are subjected to a deeper analysis for early detection of malcode infection. We explore the utility of the technique by the following experiments:

1. We insert known malware into a randomly chosen location of some randomly chosen files, and test whether we can identify a) the infected files, and b) the likely location of the malcode insertion. The results achieved indeed show promise. Reasonable detection accuracy of suspect files can be determined using the methods proposed when insertion appears at the head or tail of a file. However, reliable detection of embedded malcode within interior portions of files is hard to achieve using 1-gram models.
2. Whether virus files can be accurately classified as distinct from other ordinary executable files. The results indicate that virus executables can be reasonably well distinguished from standard Windows executables purely determined on the basis of their binary content 1-gram distributions.
3. One would expect a self-encrypted, or compressed file object should be easily discernible. Thus, we also compute the n-gram distance of a set of files against the uniform distribution to test whether purposely obfuscated self-encrypted files may be revealed as substantially different from other executable files. Indeed, such self-encrypted files are easy to observe.

In the next section, we briefly describe related research. Section 3 describes the experiments of inserting malware into normal files and how well a commercial AV scanner performed in detecting these infected files. Section 4 reviews our Fileprint n-gram analysis and section 5

discusses several experiments using these techniques to detected infected files. The work performed has been primarily focused on 1-gram analysis, i.e. the frequency distribution of byte values in a file's binary content. Some experiments have been performed using 2-grams. We plan to perform 3-gram analysis as well in our future work. Higher order grams pose two performance problems. Higher order grams produce an exponentially increasing range of feature values, and hence the model computation could grow rapidly. Further, far more data is necessary to compute effective centroids. That is to say, the space of training data is sparse as the size of the gram increases. Even so, the technique shows promise and hence such an exploration of higher order n-gram analysis appears to be warranted.

## 2. Related work

Malware detection is an important security problem under study for quite some time by many researchers. Various approaches may be partitioned into active run-time detection, static analysis of code, or a combination of the two. Dynamic run-time analysis includes methods that test whether code violates pre-specified security policies. Approached based on wrappers [2,7], or specification-based anomaly detection [14, 16, 18] and sandboxing [1] have been proposed. Other approaches to anomaly detection are based on learning normal system operation (via system call analysis, for example) rather than pre-specifying normal operation [5, 6, 21]. Besides this work on detecting malware directly, others have proposed the approach of protecting hosts from infection when executing possibly untrusted code. These methods include augmenting software as proof carrying code (eg., [13]) to guarantee safety of execution.

N-gram [4] analysis has been widely used in a number of text classification tasks. This language independent statistical analysis technique has also been applied to detect malicious software [8, 15 21]. In particular, 1-gram analysis (or byte value frequency distributions) has been successfully applied to the problem of detecting anomalous network payload [19, 20]. Instead of using any structural or run-time information about software, we conjecture that simple statistical static analysis can be applied to provide valuable evidence of possible malicious software without any specific information about the format of a set of objects being tested. It is this approach that is the subject matter of this paper, extending the prior work in [11, 12]. In [17] a simple technique is demonstrated to locate likely key material stored on disk. The attack estimates high entropy regions of binary code by a simple measure of the number of distinct byte values encountered within that region. In our work, we measure the entire byte value distribution of a file or portion of a file and measure its distance to the uniform distribution when determining if a file is likely encrypted, random byte values.

## 3. How easily can we deceive anti-virus software?

Malware may be easily transmitted among machines as (P2P) network shares. One possible stealthy way to infect a machine is by embedding the malicious payload into files that appear normal and that can be opened without incident. A later penetration by an attacker or an embedded Trojan may search for these files on disk to extract the embedded payload for execution or assembly with other malcode. Or an unsuspecting user may be tricked into launching the embedded malcode in some crafty way. In the latter case, malcode placed at the head of a PDF file can be directly executed to launch the malicious software. Social engineering can be employed to do so. One would presume that an AV scanner can check and detect such infected file shares if they are infected with known malcode for which a signature is available. The question is whether a commercial AV scanner can do so. Will the scanning and pattern-matching techniques capture such embeddings successfully? An intuitive answer would be "yes". We show that is not the case in all cases.

We conducted the following experiments. First we collected a set of malware [22], and each of them was tested to verify they can be detected by a COTS anti-virus system[2]. We concatenate each of them to normal PDF files, both at the head and tail of the file. Then we manually test whether the COTS AV can still detect each of them, and whether Acrobat can open the PDF file without error. These tests were performed on a Windows platform. The results are summarized in table 1. The COTS anti-virus system has surprisingly low detection rate on these infected files with embedded malware, especially when malware is attached at the tail. For those that were undetected, quite a few can still be successfully opened by Acrobat appearing exactly as the untouched original file. Thus, the malcode can easily reside inside a PDF file without being noticed at all. An example of the manipulated PDF file is displayed in figure 1. The apparent reason Adobe Acrobat Reader (version 7.0) opens infected files with no trouble is that it scans the head of a file looking for the PDF "magic numbers" signaling the beginning header meta-data necessary to interpret the rest of the binary content. Thus, the portions passed over by the reader while searching for its header data provides a convenient place to hide malcode.

**Table 1. COTS AV etection rate and Acrobat behavior on embedded malcode.**

| Total virus/worm | Virus at the head of PDF | | Virus at the tail of PDF | |
|---|---|---|---|---|
| | Can detect | Can open | Can detect | Can open |
| 223 | 162 (72.6%) | 4 /not detected | 43 (19.3%) | 17 /not detected |

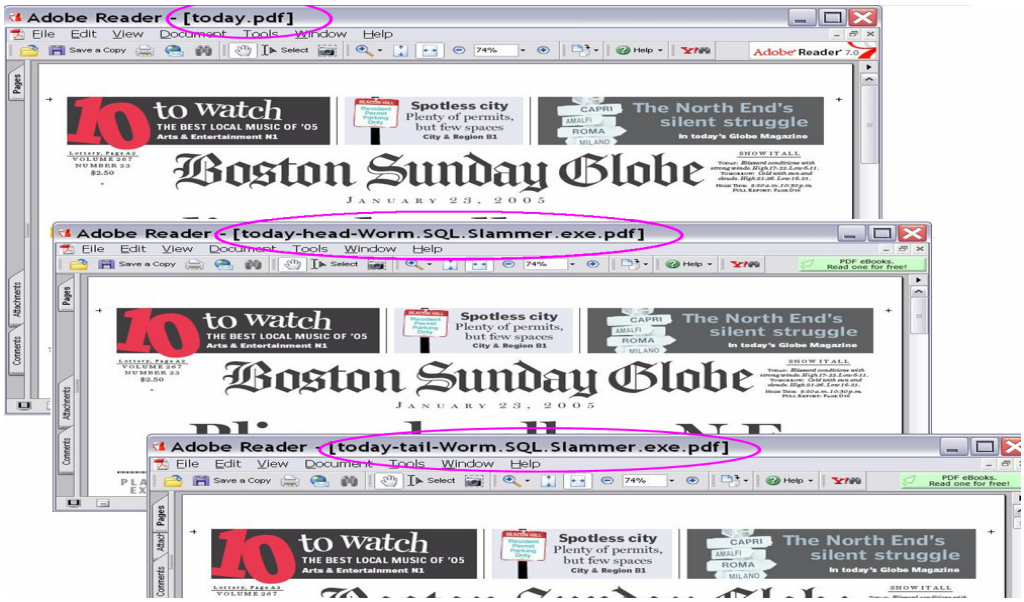


**Figure 1. Screenshot of original and malware embedded PDF file**

We also performed another experiment by inserting the malware into some random position in the middle of the PDF file. But since PDF has its own encoding and such blind insertion can easily break the encoding, generally this is easily noticed by the Acrobat Reader when opening the file. This was the case and hence malware simply appended to the head/tail is obviously easier without causing any errors by the reader. We repeated this experiment on DOC files using some selected malwares, and got a similar result. The following table provides the detailed results of several

[2] This work does not intend to evaluate nor denigrate any particular COTS product. We chose a widely used AV scanner that was fully updated at the time the tests were performed. We prefer not to reveal which particular COTS AV scanner was used. It is not germane to the research reported in this paper.

malware insertion experiments using well known malware. Only CRII can be reliably detected no matter where it is inserted, while Slammer and Sasser were missed.

**Table 2. Detailed example of insertion using several well-known malware**

| **Slammer** | | | |
|---|---|---|---|
| | Virus at head | In the middle | At tail |
| PDF file | Not detect/open fine | Not detect/open error | Not detect/open fine |
| DOC file | Not detect/open error | Not detect/open error | Not detect/open fine |
| **CodeRed II** | | | |
| Can be detected anywhere | | | |
| **Sasser** | | | |
| | Virus at head | In the middle | At tail |
| PDF file | Can detect | Not detect/open error | Not detect/open error |
| DOC file | Can detect | Not detect/open error | Not detect/open fine |

Another experiment focused on Windows executables, like WINWORD.EXE. After analyzing the byte value distributions of executables, we noticed that byte value 0 dominated all others. Application executables are stored on disk using a standard block alignment strategy of padding of executables (falling at addresses n*4096) for fast disk loading. These zero-ed portions of application files provide ample opportunity to insert hidden malcode. Instead of concatenating malcode, in this case we insert the malcode in a continuous block of 0's long enough to hold the whole malcode and store the file back on disk. Again, we tested whether a COTS AV scanner would detect these poisoned applications. It did not. We performed this experiment by replacing the padded segments of WINWORD.EXE, from byte positions 2079784-2079848. Figure 2 shows two versions of the application, the normal executable and the other infected with malcode, and both were able to open DOC files with no trouble.

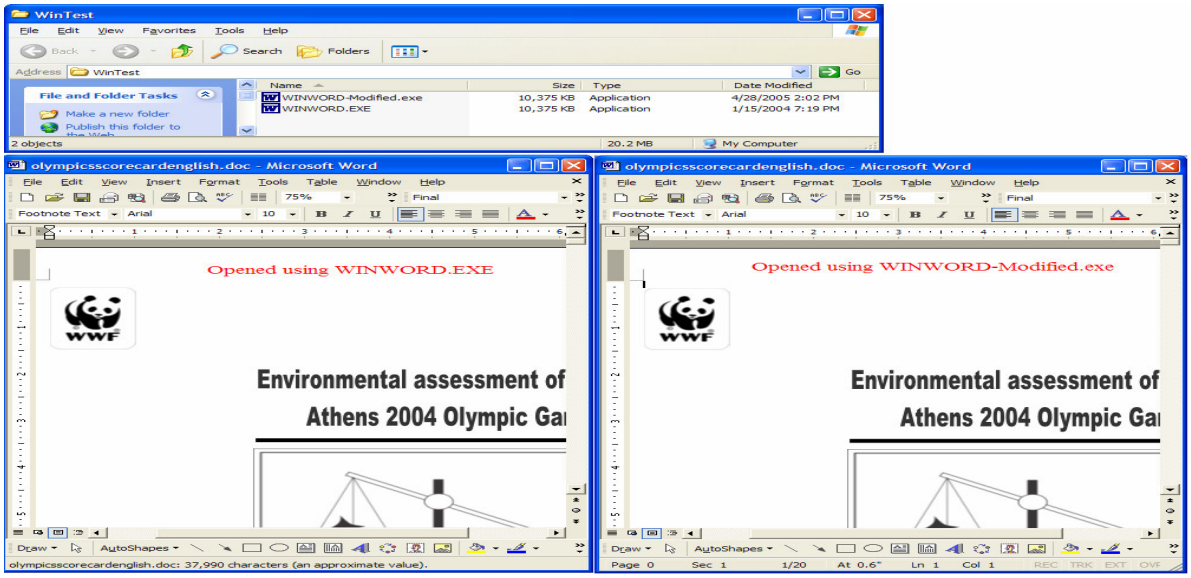


**Figure 2: Opening of a normal DOC file using the original WINWORD.EXE (left) and the infected one WINWORD-Modified.EXE (right).**

## 4. Analysis method

In this section, we introduce the n-gram analysis method and describe the fundamental test and evaluation methodology.

### 4.1 N-gram analysis

An n-gram [4] is a subsequence of n consecutive tokens in a stream of tokens. n-gram analysis has been applied in many tasks, and is well understood and efficient to implement. By converting a string of data to a feature vector of n-grams, one can map and embed the data in a vector space to efficiently compare two or more streams of data. Alternatively, one may *compare the distributions* of n-grams contained in a set of data to determine how consistent some new data may be with the set of data in question. In this work, we experimented with both 1-gram and 2-gram analysis of ASCII byte values. The sequence of binary content is analyzed, and the frequency and variance of each gram is computed. Thus, in the case of 1-grams, two 256-element vectors (histograms) are computed. This is a highly compact and efficient representation, but it may not have sufficient resolution to represent a class of file types. The following plot shows that different file types do indeed have significant distinct n-gram patterns. Thus, different file types can be reasonably well classified using this technique (see [11]).

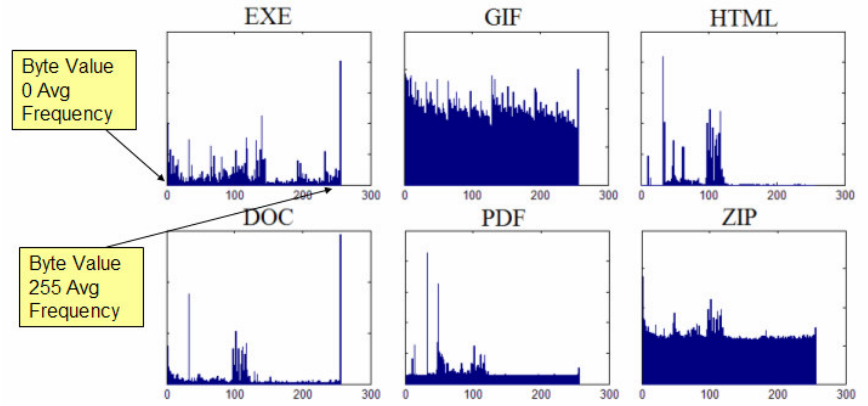


**Figure 3. 1-gram distribution for different file types.**

### 4.2 Truncation and multiple centroids

Truncation simply means we model only a fixed portion of a file when computing a byte distribution. That portion may be a fixed prefix, say the first 1000 bytes, or a fixed portion of the tail of a file, as well as perhaps a middle portion. This has several advantages. First, for most files, it can be assumed that the most relevant part of the file, as far as its particular type is concerned, is located early in the file to allow quick loading of meta-data by the handler program that processes the file type. Second, viruses often have their malicious code at the very beginning of a file. Hence, viruses may be more readily detected from this portion of the file. However, viruses indeed may also be appended to the end of a file, hence, truncation may also be applied to the tail of a file to determine whether a file varies substantially from the expected distribution of that file type. The last, truncation dramatically reduces the computing time for model building and file testing. In network applications this has obvious advantages.

On the other hand, files with the same extension do not always have a distribution similar enough to be represented by a single model. For example, EXE files might be totally different when created for different purpose, such as system files, games, or media handlers. Thus, an alternative strategy for representing files of a particular type is to compute "multiple models". We do this via a clustering strategy. Rather than computing a single model $M_A$ for files of type A, we compute a set of models $M^k_A$, $k>1$. The multiple model strategy requires a different test methodology, however. During testing, a test file is measured against all models to determine if it matches at least one. The collection of such models is considered a fileprint for the entire class. The multiple

model technique creates more accurate models, and separates foreign files from the normal files of a particular type in more precise manner. The multiple models are computed by the *K-Means* algorithm under *Manhattan Distance* as the similarity metric. The result is a set of $K$ centroid models, $M^k_A$ which are later used in testing files for various purposes.

## 5. N-gram experiments on files

To test the effectiveness of the n-gram analysis on files, we conduct several experiments to determine whether it can correctly classify files and whether it can detect malware. We report each of them as a subsection. First we apply the technique to detect the malware-embedded files from normal files of the same type. We then report the results of experiments to determine whether normal executables can be distinguished from viruses. Finally we demonstrate how the uniform 1-gram distribution can be used to readily detect self-encrypted files.

### 5.1 Data sets

The test files used in the experiments include 140 PDF files, 31 normal application executable files, 45 spyware, 331 normal Windows executable under folder System32 and 571 viruses/worms. The malicious files are collected from emails, internet sources [22] and some target honeypot machines setup for this purpose in our lab. The 31 normal applications are common third party vendor executables, such as Ad-aware, Adobe Reader, Firefox, etc. The PDF files were collected from the internet using a general search on *Google*. In this way, they can be considered randomly chosen as an unbiased sample.

### 5.2 Detecting malware embedded files

First we revisit our malware embedding experiment. We've seen that the COTS AV system we used can easily miss the malcode hidden inside normal appearing files. Here we apply the 1-gram analysis and see how well it may be able to detect the malicious code sequences. We collected 140 publicly available PDF files randomly chosen using *Google*. 100 of these were used to build head and tail 1-gram models. Then we test the remaining 40 normal PDF files and hundreds of malware-embedded files against the model. Since we know ground truth, we measure the detection rate exactly when the false positive rate is zero, i.e., no normal PDF files been misclassified as malware-infected. The result is displayed in table 3, which is much higher than the COTS anti-virus software detection rate, which for these files is effectively zero. Notice that the total number of malware-embedded files is different for different truncation sizes. That is because the malware used in this study differ in size and we only consider the problem of classifying a pure malcode block fully embedded in a portion of the PDF file. We consider a concatenated PDF file as a test candidate only if the malcode size is equal or greater than the truncation size used for modeling.

**Table 3. Detection rate using truncated head and tail modeling**

| Models head N bytes | | | |
|---|---|---|---|
| Detect | 1000 bytes | 500 bytes | 200 bytes |
| | 49/56(87.5%) | 314/347(90.5%) | 477/505(94.5%) |
| Models tail N bytes | | | |
| Detect | 1000 bytes | 500 bytes | 200 bytes |
| | 42/56(75%) | 278/347(80.1%) | 364/505(72.1%) |

Usually it is easier to detect the malware if they are concatenated at the head or tail of the file, since different file types usually have their own header information and ending encoding. Malcode may be significantly different from these standardized encodings. However, we test whether malwares can effectively be hidden in some middle portion of a file (presuming that the file would still possibly be opened correctly). A reasonable assumption about such insertion is that the malware is inserted as a continuous whole block. So we apply the n-gram detection

method to each block of a file's binary content and test whether the model can distinguish PDF blocks from malware blocks. If so, then can we detect the malcode hidden inside PDF files.

We compute byte distribution model using N byte blocks from 100 PDF files, then test the blocks of the malware and another 40 PDF files against the model, using Mahalanobis distance [20]. Figure 4 shows the distance of the malware blocks and PDF blocks to the normal model, using 500-byte blocks and 1000-byte blocks, respectively. In the plot we display the distance of the malcode blocks on the left side of the separating line and the normal PDF on the right. As the plots show, there is a large overlap between malcode and PDF blocks. The poor results indicate that malware blocks cannot be easily distinguished from normal PDF file blocks using 1-gram distributions.

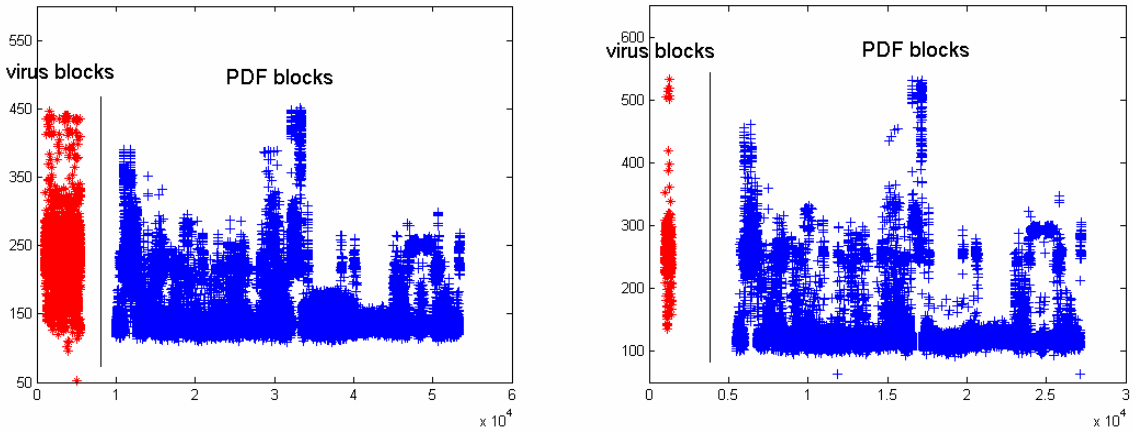


**Figure 4. The Mahalanobis distance of the normal PDF and malware blocks to the trained PDF block model. The left is 500-byte block and the right one is 1000-byte block**

In order to understand why the block based detection using 1-grams does not work well, we plot the byte distribution of each block of a normal PDF file and Sasser. The first 9 blocks of the PDF file and the first 6 blocks of Sasser are drawn in the following plots. These plots clearly show that different blocks inside a PDF file differ much in their byte distribution, and we cannot determine an absolute difference of the malcode blocks from PDF blocks. So a 1-gram statistical content analysis might not have sufficient resolution for malware block detection. Either higher order grams (perhaps 2-grams or 3-grams) may suffice, or we may need more syntax or semantic information to adequately distinguish malcode embedded in PDF files. This is part of our ongoing research.

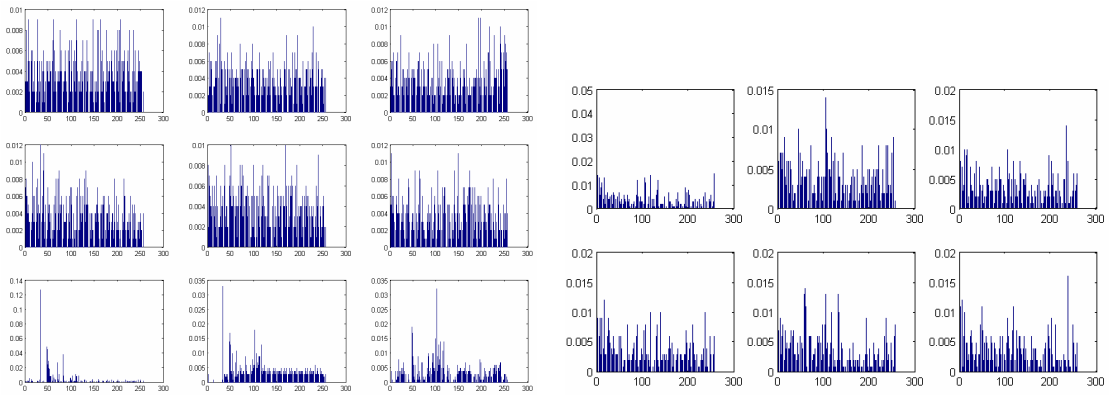

**Figure 5. Byte value distributions of blocks of the PDF file and Slammer worm.**

### 5.3 Classifying normal executables and viruses

In this experiment, we use a collection of malware executables gathered from other external sources, and compare the 1-gram and 2-gram distributions of these to the corresponding distributions of normal Windows executables to determine whether viruses exhibit any clear separating characteristics. We demonstrate that the Windows executables are generated by programming environments and compilers that may create standard headers different from those form viruses delivered via email or file shares. We apply three modeling methods to these experiments, which are one-centroid, multi-centroids and exemplar files as centroids. One centroid method trains one single model for each class (or type) of file. We build $t$ models $M_1$, $M_2$, …, $M_t$, from $n$ different file types. Then, we compute the distance of the testing file $F$ to each model, and $F$ is classified to the model with the closest distance. Alternatively, in the multi-centroids method, we build $k$ models $M^t_1$, $M^t_2$, …, $M^t_k$ using *k-means* algorithm for each file type $t$. There are $k*t$ models in total, where $t$ is the number of file types. $k$ is set to 10 in this test. The test strategy is the same as the one centroid method. The test file $F$ is classified to the model with the closest distance. We also use a set of exemplar files of each type as centroids. A set of randomly chosen normal files for each file type are used as centroids. There are $N$ models if there are $N$ training files. We also analyze the accuracy using different truncations – first 10, 50, 100, 200, 400, 600, 1000, 2000, 4000, 6000, and 8000 bytes, and the entire file. In this experiment, we evaluate both 1-gram and 2-gram analysis.

We trained on 80% of the randomly selected files of each group (normal and malicious) to build a set of models for each class. The remaining 20% of the files are used in testing. Note that all of the malicious file extensions are EXE. For each of the test files, we evaluate their distance from both the "normal model" and the "malicious model". 31 normal application executable files, 45 spyware, 331 normal executable under folder System32 and 571 viruses were tested. 3 "pairs" of groups of files are tested – Normal executable vs. spyware, normal application vs. spyware and normal executable vs. viruses. We report the average accuracy over 100 trials using cross validation for each of the modeling technique.

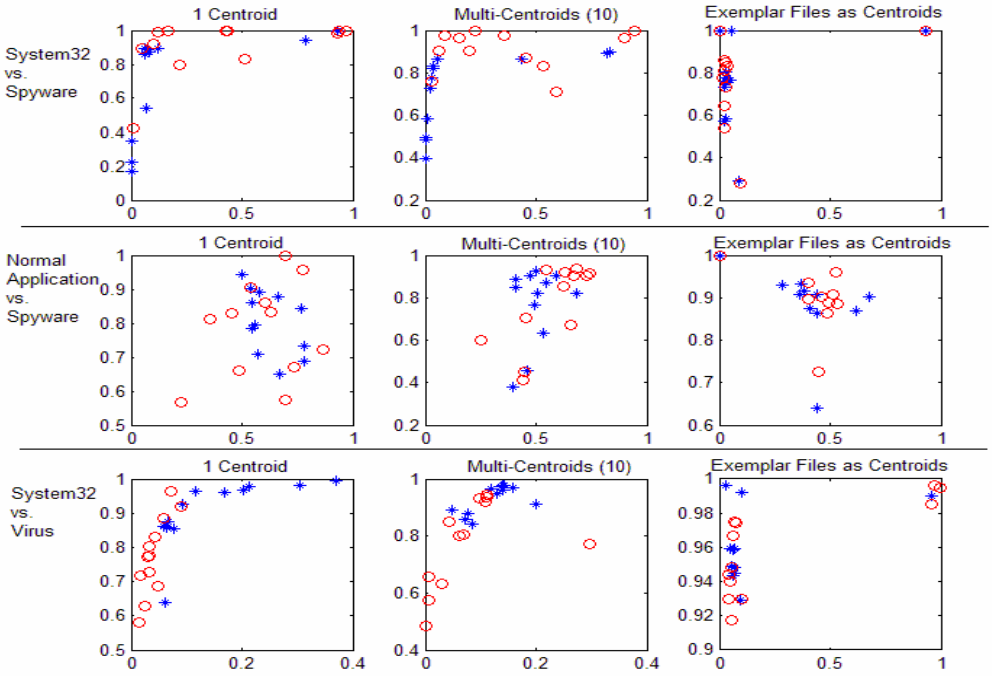


**Figure 6: 2-class classification of malware and normal EXE files. X-Axis: false positive, Y-Axis: detection rate. Asterisk marks: 1-gram test, Circle marks: 2-gram test.**

The results are shown in figure 6. Each column represents each modeling method, which are one-centroid, muli-centroids and exemplar files as centroids. The rows indicate the testing "pairs". In each plot, the X and Y-axis are the false positive rate and detection rate, respectively. The asterisk marks are 1-gram test using different sizes of truncation, and circles represent the 2-gram model results. In these plots, the sizes of truncation are not arranged in order. In these two dimensional plots, the optimum performance appears closest to the upper left corner of each plot. That is to say, a false positive rate of 0 and a detection rate of 1 is perfect performance.

The results show relatively good performance of normal executable vs. spyware and normal executable vs. virus. Because viruses and worms usually target the System32 folder, we can reasonably well detect non-standard and likely malicious files in that folder. Moreover, the performance results varied under different truncation sizes. Thus, we have considerable additional analysis to perform in our future work to identify appropriate file sizes and normalization strategies to improve detection performance. Notice that there is a high false positive rate in the normal application vs. spyware test. This is due to two reasons. First, the range of the normal application file size is too large, from 10KB to 10MB. It is hard to normalize the models when the data ranges so widely. Second, the spyware files are somewhat similar to normal application files. They are both MS Windows applications, and they may be used for similar purposes. Hence, other features may be necessary to explore ways of better distinguishing this class of files.

In the experiments performed to date, there is no strong evidence to indicate that 2-gram analysis is better than 1-gram analysis. Even though the 1-gram memory usage is much smaller and the computation speed is much faster, we may need to analyze far more many files to determine whether 2-gram analysis will perform better. As mentioned earlier, the space is far sparser when the gram size increases, and we may not have used sufficient training data to reveal good enough 2-gram models.

### 5.4 Uniform Distributions of 1-gram analysis: encrypted files and spyware

In this experiment we scan Windows files to determine whether any are close to a uniform 1-gram distribution. We thus test whether spyware obfuscated by self-encryption technology may be revealed as substantially different from other executable files on a Windows host platform. We conjecture that self-encrypted files, such as stealthy Trojans and spyware, may be detectable easily via 1-gram analysis.

The normal EXE from System32, spyware and virus files used in the experiments reported in the previous section are used here again. Moreover, we randomly select 600 files (DOC, PPT, GIF, JPG, PDF, DLL) from *Google*, 100 for each type. Since the models are normalized, the uniform distribution is an array with uniform value *1/L*, where *L* is 256 in the 1-gram test. For each of the test files, we compute the Manhattan distance against the uniform model and plot the distance in figure 7. The files that are closest to uniform distribution are listed in table 7.

As the plot shows, JPG, GIF and PDF files are compressed, so they are more similar to the uniform distribution. System32 files and DLL files are not self-encrypted, and most of the virus and spyware tested are also not self-encrypted. However, some of the normal files are self-encrypted and quite similar to the random distribution. An interesting example is the application ad-aware.exe, which is a COTS adware detection application that apparently uses self-encryption perhaps to protect its intellectual property.

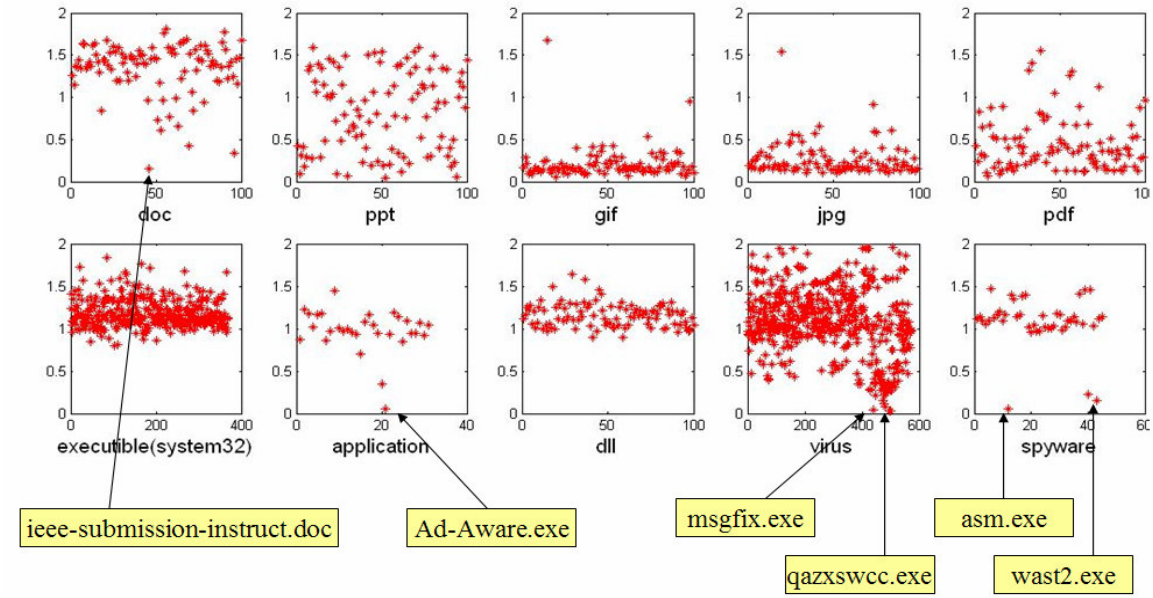


**Figure 7: the distance of testing files against the uniform distribution. X-Axis: the test files, Y-Axis: the distance.**

**Table 7. Description for several representative self-encrypted files**

| File name | Description |
|---|---|
| Ieee-submission-instruct.doc | An IEEE submission format instruction word file |
| Ad-Aware.exe | Ad-Aware.exe: ad-aware from lavasoft, searches and removes spyware and/or adware programs from your computer. |
| msgfix.exe | msgfix.exe is a process which is registered as the W32.Gaobot.SN Trojan. This Trojan allows attackers to access your computer, stealing passwords and personal data. |
| Qazxswcc.exe | qazxswcc.exe is a process which is registered as a backdoor Trojan. |
| Asm.exe | asm.exe is an advertising program by Gator. This process monitors your browsing habits and distributes the data back to the Gator for analysis. This also prompts advertising pop-ups. |
| wast2.exe | wast2.exe is an adware based Internet Explorer browser helper object that deliver targeted ads based on a user's browsing patterns. Twain-Tech does not provide any other relevant purpose other than pop-ups. |

## 6. Conclusion

In this paper, we demonstrate that simple techniques to embed known malcode in normal files can easily bypass signature-based detection. We successfully inserted known malcode in both non-executable (PDF and DOC) and executable (WINWORD.EXE) files without being detected, and several were normally opened or executed. Various code obfuscation techniques can also be used by crafty attackers to avoid inspection by signature-based methods. We propose an alternative approach to augment existing signature-based protection mechanisms with statistical content analysis techniques. Rather than only scanning for signatures, we compute the statistical binary content of files in order to detect anomalous files or portions of files which may indicate a malcode embedding. We tested sampled normal MS Windows system files, normal applications, spyware, viruses and worms. The results are encouraging. The normal system files and malware

can be well classified. This work is preliminary in nature; much additional analysis is needed to improve the results including exploration of higher order grams, and other modeling techniques. If the methods continue to perform well, we will also be faced with counter-measures by crafty attackers. Work is also under way to study *counter-evasion* techniques; attackers may craft malcode to mimic the distributions of normal files. Such mimicry attacks may be prevented by various modeling techniques under study and that will be reported in future papers.

## References:

[1] K. G. Anagnostakis, S. Sidiroglou, P. Akritidis, K.Xinidis, E. Markatos, and A. D. Keromytis. "Detecting Targeted Attacks Using Shadow Honeypots", Proceedings of the 14th USENIX Security Symposium, 2005.

[2] R. Balzer and N. Goldman, "Mediating Connectors", Proceedings of the 19 IEEE International Conference on Dis-. tributed Computing Systems Workshop, 1994**.**

[3] M. Christodorescu and S. Jha, "Static Analysis of Executables to Detect Malicious Patterns", In Proceedings of the 12th USENIX Security Symposium, August 2003

[4] M. Damashek. "Gauging similarity with n-grams: language independent categorization of text." Science, 267(5199):843--848, 1995

[5] E. Eskin, W. Lee and S. J. Stolfo. ``Modeling System Calls for Intrusion Detection with Dynamic Window Sizes.'' *Proceedings of DISCEX II*. June 2001.

[6] J. Giffin, S. Jha, and B. Miller. "Detecting Manipulated Remote Call Streams". In the 11th USENIX Security Symposium, 2002

[7] C. Ko, M. Ruschitzka, and K. Levitt. "Execution monitoring of security-critical programs in distributed systems: A specification-based approach". In Proceedings of the IEEE Symposium on Security and Privacy, 1997.

[8] J. Kolter and M. Maloof. "Learning to Detect Malicious Executables in the Wild." In the Proceedings of ACM SIGKDD, 2004

[9] C. Krugel, W. Robertson, F. Valeur, G. Vigna. "Static Disassembly of Obfuscated Binaries". In Proceedings of USENIX Security Symposium, 2004.

[10] A. Kurchuk and A. Keromytis. "Recursive Sandboxes: Extending Systrace to Empower Applications". In Proceeding of the 19th IFIP International Information Security Conference (SEC), Aug. 2004

[11] W. Li, K. Wang, S. Stolfo and B. Herzog. "Fileprints: identifying file types by n-gram analysis". 6th IEEE Information Assurance Workshop, West Point, NY, June, 2005.

[12] M. McDaniel, M. Heydari. "Content Based File Type Detection Algorithms". 36th Annual Hawaii International Conference on System Sciences (HICSS'03).

[13] G. Necula, P. Lee. "Proof-Carrying Code" In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Langauges (POPL)*, 1997

[14] F. B. Schneider. "Enforceable security politics". Technical Report 98-1664, Cornell University, 1998

[15] M. Schultz, E. Eskin, and S. Stolfo. "Malicious Email Filter - A UNIX Mail Filter that Detects Malicious Windows Executables." In Proceedings of USENIX Annual Technical Conference - FREENIX Track. Boston, MA: June 2001.

[16] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang and S. Zhou. "Specification-Based Anomaly Detection: a new approach for detecting network intrusions", RAID 2002

[17] A. Shamir and N. van Someren, Playing Hide and Seek with stored keys, Financial Cryptography 1999.

[18] D. Wagner and D. Dean. "Intrusion Detection via Static Analysis". In IEEE Symposium in Security and Privacy, Oakland, CA, 2001

[19] K. Wang, G. Cretu and S. Stolfo. "Anomalous Payload-based Worm Detection and Signature Generation". To appear in Proceedings of the Eighth International Symposium on Recent Advances in Intrusion Detection, Sept. 2005.

[20] K. Wang and S. Stolfo. "Anomalous Payload-based Network Intrusion Detection". In Proceedings of the Seventh International Symposium on Recent Advance in Intrusion Detection (RAID), Sept. 2004.

[21] C. Warrender, S. Forrest, and B. Pearlmutter."Detecting intrusions using System calls:alternative data models, Proc.of 1999 IEEE Symposium on Security and Privacy, 1999.

[22] VX Heavens http://vx.netlux.org/

[23] Net Alliance http://www.shellcode.com.ar/docz/bof/pe.pdf