Using Executable Slicing to Improve Rogue Software Detection Algorithms

Jan Durand, Juan Flores, Travis Atkison*
Louisiana Tech University
{jrd037, jcf028, atkison}@latech.edu
Computer Science Department
Box 10348
Ruston, LA 71272
USA

Nicholas Kraft, Randy Smith
University of Alabama
{nkraft, rsmith}@cs.ua.edu
Department of Computer Science
Box 870290
Tuscaloosa, AL 35487
USA

Abstract

This paper describes a research effort to use executable slicing as a pre-processing aid to improve the prediction performance of rogue software detection. The prediction technique used here is an information retrieval classifier known as cosine similarity that can be used to detect previously unknown, known or variances of known rogue software by applying the feature extraction technique of randomized projection. This paper provides direction in answering the question of is it possible to only use portions or subsets, known as slices, of an application to make a prediction on whether or not the software contents are rogue. This research extracts sections or slices from potentially rogue applications and uses these slices instead of the entire application to make a prediction. Results show promise when applying randomized projections to cosine similarity for the predictions, with as much as a 4% increase in prediction performance and a five-fold decrease in processing time when compared to using the entire application.

Keywords: rogue software detection, executable slicing, information retrieval, n-gram analysis, cosine similarity, randomized projections

1. Introduction

With today's market globalization of software development and the proliferation of malicious attackers, it is becoming almost impossible to have any trust in the software that is loaded onto our systems. Rouge applications, or applications in which code has been added, modified or removed with the intent of causing harm or subverting a system's intended function (McGraw & Morrisett, 2000), are becoming more and more prevalent. To combat these

* corresponding author

infiltrations, consumers, as well as corporations, are turning to anti-virus software products, which contain virus detection engines. Though very good at what they do, virus detection engines rely on a database of signatures to detect known rogue applications. Signature based systems inherently limit the detection of new and previously unknown types of rogue attacks. To that end there have been several research attempts to overcome these limitations. In one of these attempts (Atkison, 2009) we have shown the value of using randomized projection algorithms in detecting malicious applications.

The purpose of this paper is to provide methods and techniques to overcome the limitations inherent in the signature-based systems mentioned above. Through this research effort, we will provide a methodology for detecting rouge applications by enhancing the random projection, dimensionality reduction concept by using executable slicing. Executable slicing is a strategic method of compartmentalizing applications, and is used as a pre-processor to the algorithm. It will be shown that by adding this pre-processing step a significant gain in accuracy as well as in precision and recall can be achieved.

The following section provides a background description of previous methods that involve static analysis, information retrieval and randomized projection. In Section 3, the experimental design of this work is discussed including software and data used. In Section 4, results achieved are described. Finally, in Section 5 the conclusion and future directions are presented.

2. Background

Developing effective potential solutions to the malicious software detection problem is an important direction in host security research. There have been few research papers, (Kang, Poosankam, & Yin, 2007; Perdisci, Lanzi, & Lee, 2008) are good examples, that pose the option of executable slicing while looking at malicious detection. Though their focus is directed toward packed executables, the focus of this paper is to show that statically analyzing sections or slices of an executable will improve prediction rates of non-packed, stand-alone executables. It is important to understand the methods and techniques that are used for these predictions. Since the randomized projection technique in this solution is used in conjunction with an information retrieval prediction algorithm we will include a small background on information retrieval as well as static analysis.

2.1. Static Analysis

Static analysis, sometimes referred to as static program analysis or static code analysis, is the examination of the source or object code of an application in order to identify patterns that indicate potential design errors and/or security threats (Food and Drug Administration [FDA], 2010). This analysis approach eliminates the need to execute an application in order to determine its behavior, contrary to its counter-part dynamic analysis, thus avoiding the potential compromise of the host system.

Static analysis has proven to be a very useful tool in detecting undesirable or vulnerable code in applications. There have been several research efforts such as (Bergeron, et al., 2001; Bergeron, Debbabi, Erhioui, & Ktari, 1999; Christodorescu & Jha, 2003; FDA, 2010; Jovanovic, Kruegel, & Kirda, 2006) that have incorporated the use of static analysis to detect malicious code in executable files.

Christodorescu et al. (Christodorescu & Jha, 2003) presented a static analysis framework for identifying malicious code patterns in executables and implemented SAFE, a static analyzer for executables. In their research, they show that SAFE is resilient to common obfuscation transformations on malicious code while three popular anti-virus scanners were susceptible to these attacks (Christodorescu & Jha, 2003).

In (Bergeron, et al., 1999) and (Bergeron, et al., 2001), Bergeron et al. present a three-step approach for detecting malicious code in applications, which they claim is capable of detecting unknown malicious code (Bergeron, et al., 2001). This approach consists of generating an intermediate representation, analyzing control and data flows to capture security-oriented program behavior, and performing static verification of critical behaviors against security policies (Bergeron, et al., 2001).

Jovanovic et al. (Jovanovic, et al., 2006) tackle the problem of vulnerable Web applications using static code analysis. They make use of a number of static analysis techniques including flow-sensitive, interprocedural and context-sensitive data flow analysis to locate vulnerable points in an application and then improve the accuracy of the search results via alias and literal analysis (Jovanovic, et al., 2006). This framework was then implemented as Pixy, an open-source Java tool which targets taint-style vulnerabilities such as SQL injection attacks and cross-site scripting (Jovanovic, et al., 2006).

The research proposed in this paper makes use of static analysis techniques such as executable slicing in conjunction with information retrieval techniques and randomized projection in order to detect malicious applications.

2.2. Information Retrieval

Information retrieval traditionally is the part of computer science, which from a collection of written documents studies the retrieval of information (not data) (Baeza-Yates & Ribeiro-Neto, 1999). These retrieved documents' aim is to satisfy an information need (Baeza-Yates & Ribeiro-Neto, 1999). The process can be thought of as combing through a set of documents, called the corpus, to find a certain piece of information that has a relationship to a given entity, called the query. That piece of information can either be an entire document, set of documents or a subset of a document. Within the information retrieval community several methods exist for finding these pieces of relevant information. These methods include vector space models, latent semantic indexing models and statistical confidence models as well as others. The first approach to represent a document as a set of terms were vector space models (Liu, et al., 2004). As their name implies vector space models represent their data as a vector with each dimension being defined as a term which may or may not have a weight associated with it (Salton, Wong, & Yang, 1975). One of the most common vector space models is cosine similarity. Cosine similarity determines the similarity between two data vectors by measuring the angular distance between them. The property of cosine is that it is 1.0 for identical vectors and 0.0 for orthogonal vectors (Singhal, 2001) The following is the formula used in our work for computing cosine similarity.

$$\text{Cosine Similarity } (Q, D) = \frac{\sum_i w_{Q,i} w_{D,i}}{\sqrt{\sum_i w_{Q,i}^2}\sqrt{\sum_i w_{D,i}^2}} \tag{2.1}$$

This formula computes the similarity between a query *Q* and a document *D*. It does so by summing the individual components of the two entities represented in the formula as *w*. The individual components for this research are defined as *n*-grams. An *n*-gram is any substring of length *n* (Baeza-Yates & Ribeiro-Neto, 1999), that can also be described as a feature. A feature in this context is an extracted piece of information that in part describes the item from which it was extracted. Here the gram (which will be the composite of the substring) is a byte in hexadecimal form extracted from a binary executable in the corpus. For example, the string '03 A4 EC 17' represents 4 bytes in hexadecimal form and '03A4' is an *n*-gram of length 2 of that string. Therefore, w_{Qi} is the weight of the i^{th} *n*-gram in the query and w_{Di} is the weight of the i^{th} *n*-gram in the document.

There have been other efforts (Abou-Assaleh, Cercone, Keselj, & Sweidan, 2004a, 2004b; Henchiri & Japkowicz, 2006; Kephart, et al., 1995; Marceau, 2000; Reddy & Pujari, 2006) to use the information retrieval concept of *n*-grams as features. Henchiri et al. (Henchiri & Japkowicz, 2006) and Abou-Assaleh et al. (Abou-Assaleh, et al., 2004a, 2004b) both use the Common N-Gram (CNG) analysis method, which uses the most frequent *n*-grams to represent a class, to detect rogue applications. Henchiri further limits the number of features by imposing a "hierarchical feature selection process" (Henchiri & Japkowicz, 2006). Marceau (Marceau, 2000) puts an interesting twist on the problem of using *n*-grams as features by having "multiple-length" grams instead of the traditional single *n*-length gram. Marceau does this by first creating and then compacting a suffix tree, a structure that allows fast string operations be provides suffixes of given strings, to a Directed Acyclic Graph (DAG). Reddy et al. (Reddy & Pujari, 2006) develop their own unique *n*-gram feature selection measure called, 'class-wise document frequency.'

2.3. Randomized Projection

Rogue application detection, following the genre of information retrieval, suffers from the problem that the data, once processed, is encoded in extremely high dimensions. This high-dimensional data limits the kind and amount of analysis that can be performed. One method for dealing with the reduction of this type of high-dimensional data is known as feature extraction. Feature extraction transforms, either linearly or non-linearly, the original feature set into a reduced set that retains the most important predictive information. Examples of this type include principle component analysis, singular value decomposition and randomized projection.

In randomized projection, using a random matrix whose columns have unit lengths the original high-dimensional data is projected onto a lower-dimensional subspace (Bingham & Mannila, 2001). This type of projection attempts to retain the maximum amount of information embedded in the original feature set while substantially reducing the number of features required. This feature reduction will allow for greater amounts of analysis to be performed. The core concept has been developed out of the Johnson-Lindenstrauss lemma (Johnson & Lindenstrauss, 1984) which states that any set of n points in a Euclidean space can be mapped to $\Re^t$ where t = with distortion $\leq 1 + \varepsilon$ in the distances. Such a mapping may be found in random polynomial time. A proof of this lemma can be found in (Dasgupta & Gupta, 1999).

There have been some efforts (Bingham & Mannila, 2001; Mannila & Seppänen, 2001; Papadimitriou, Raghavan, Tamaki, & Vempala, 2000) that look at using randomized projection techniques for dimensionality reduction. Randomized projection refers to projecting a set of points from a high-dimensional space to a randomly chosen low-dimensional subspace (Vempala, 2004). Minnila et al. (Mannila & Seppänen, 2001) use random projection techniques to map sequences of events and find similarities between them. Their specific application is in the telecommunication field looking at how to better handle network alarms. Their goal is to show the analyst past circumstances that resemble the current one (Mannila & Seppänen, 2001) so that a more informed decision about the current situation can be made. Though their proposed solution is not perfect, it does show the promise of using randomized projections in a similarity based application.

Bingham and Mannila (Bingham & Mannila, 2001) apply randomized projections to an image and text retrieval problem. In comparison to this research problem, their dimensions are not as large, 2500 for images and 5000 for text but the results are still significant. The purpose of their work was to show that compared to other more traditional dimensionality reduction techniques, such as principle component analysis or singular value decomposition, randomized projections offered a greater detail of accuracy. The authors were also able to show that there was a significant computation saving by using randomized projections over other feature extraction techniques, such as principle component analysis.

In another text retrieval application, Kaski (Kaski, 1998) successfully applied randomized projections in his text retrieval application that used WEBSOM, a graphical self-organizing map. Again Kaski turned to randomized projection as a method to overcome the computation expense that made other dimensionality reduction techniques infeasible when handling high-dimensional data sets. After incorporating randomized projection into their tool the authors gained an additional 5% increase in classification and topic separation over previous methods used (Kaski, 1998).

The following efforts (Kurimo, 1999; Lin & Gunopulos, 2003; Papadimitriou, et al., 2000) use randomized projection in conjunction with latent semantic indexing. Papadimitriou et al. (Papadimitriou, et al., 2000), looking at another information retrieval technique, show positive results in using randomized projections as a pre-processor to the computationally expensive Latent Semantic Indexing. By simply applying randomized projection to their data before computing the Latent Semantic Indexing, their asymptotic running time for the overall system improved from $O(mnc)$ to $O(m(\log^2 n + c\log n))$, where m and n are the matrix size, c is the average number of terms per document (Papadimitriou, et al., 2000).

3. Experiment

For the experiments presented in this paper, a rogue application detection tool suite was developed. All of the experiments were run on commodity hardware running the Fedora Linux operating system. It is very significant that we were able to complete all of these experiments on commodity hardware. It shows that large, specialized machines are not needed to perform rogue application detection and that this work can be broadly applied across almost any level of architecture that researchers/developers may have and still gain the significantly positive results that were obtained and discussed below. In addition, this software and the methods that it supports can easily take advantage of commodity cluster hardware for substantial gains in performance.

3.1. Similarity Software

The rogue application detection tool suite created for this experiment provides functionality to input Windows formatted binary executables and then creates an m-dimensional data space that contains vectors representing those applications. It can create these vectors from the entire application or slices (sections) of the application. The sections used in these experiments were the data and code sections. In these experiments, m is the number of total possible n-grams that can be extracted from the ingested applications, one dimension for each possible n-gram. The information stored in each of the dimensions can take on one of several possible values: the absolute total number of occurrences of the particular n-gram in the application, the normalized value of the total number of occurrences of the particular n-gram in the application, or finally a 1 if the application contained the particular n-gram or a 0 if it did not. Once the m-dimensional vectors have been created, the randomized projection matrix algorithm is then applied. In the method of randomized projection via matrix multiplication, the original m-dimensional data, let's say a $d \times m$ matrix D, is projected to a k-dimensional ($k << m$) subspace through the origin, using a random $m \times k$ matrix R whose columns have unit lengths (Bingham & Mannila, 2001). Selecting vectors that are normally distributed, random variables with a mean of 1 and a standard deviation of 0, populates the random matrix. After the original feature matrix is multiplied by the random matrix, the resulting $d \times k$ matrix is a low-dimensional embedding of the original high-dimensional features. The cosine similarity algorithm is then applied to the query application's vector and the corpus applications' vectors. The cosine similarity algorithm followed is the same as shown in Eq. (2.1) above. A special feature of this software is that it has the ability to shift the n-gram window not only by the more traditional byte offsets but also by bit offsets. This allows for a more fine grain tuning of the vector values, e.g., if the malicious attacker performs bit shifting on the rogue applications. It also provides for more accurate similarity result calculations.

3.2. Data Set

The data set that was compiled together for the experiments described in this section consisted of 1544 Windows formatted binary executable files. None of the files in the data set were larger than 950KB. Of these files 303 were extracted from a fresh installation of the Windows XP operating system. Another

406 were extracted from a fresh installation of Windows Vista operating system. Both of these sets were obtained by installing the respective operating system in a virtual environment on a commodity PC. These virtual environments were not connected to the Internet and therefore provided a safe location. This ensured that it would allow for application extraction without the worry of rogue infiltration during the gathering phase of the research effort. This process provided a total of 709 files that were in the data set and that were considered benign. The remaining 835 files for the data set were rogue, Trojan horse applications that were downloaded from various websites on the Internet including http://www.trojanfrance.com and http://vx.netlux.org.

3.3. Procedure

This section describes the overall flow of this experiment. The feature set (*n*-grams) was extracted from the corpus. The size of the *n*-grams was varied from a 3-byte, 5-byte and a 7-byte window. The randomized projection method described above in section 3.1 was applied to the original high-dimensional data set to produce three separate new low-dimensional embeddings, which contained 500, 1000 and 1500 features each. The cosine similarity algorithm was then applied between each vector in these reduced dimensional data sets over a range of cosine similarity threshold values, ranging from 0 to 1.0 in 0.05 increments, to produce prediction values. These prediction values were then used to classify each document vector as either malicious or benign. The results obtained from these experiments are presented below.

4. Results

To determine if executable slicing can be a useful pre-processing tool, multiple instantiations of the data set were created. The first instantiation involved using the entire or whole application itself. This instantiation of the data is the one that is used by all of the researchers that are mentioned in the literature survey described in section 2. The remaining three were created through extracting and combining well-known defined sections from the whole application. The second and third instantiations were created by extracting the code and data sections from each application using the PE Explorer tool from Heaventools Software (Heaventools Software, 2009). To confirm the accuracy of this tool several of the applications in the data set were hand dissected, comparing these to the results provided by PE Explorer and the tool proved to be very accurate. To create the fourth instantiation of the data set, the data and code sections were combined together via a string append operation. These additional instantiations were done to determine if extracted sections of each application could prove more fruitful in detection than just using the entire application. The thought process behind creating these multiple instantiations was as follows. Since all of the applications in the data set were valid Windows format executables, there would have to be an inherent similarity in all of them. This comes from both structure and header contents that may hamper attempts to produce valid and viable rogue application detection. By extracting the data and code sections, this inherent similarity was removed and allowed the detection methods to concentrate on the true differences in the applications. It must be noted that with the combined data and code data set instantiation a potentially 'false' set of features is created at the point of fusion. For example, consider the union of byte sequences '0F 1C A2' and '45 B0 12'. Extracting *n*-grams of length 3 from the resulting sequence '0F 1C A2 45 B0 12', where 1 gram is 2 contiguous characters, the *n*-grams '1CA245' and 'A245B0', are produced at the junction '<u>A2 45</u>'. This set of *n*-grams is considered 'false' since its members do not exist in the individual strings. However, the cardinality of this set is extremely small, at most 6 for these experiments, when compared to the entire set of features that are extracted and therefore will not hamper any detection capabilities of the tool suite.

4.1. Validation

As with any new method, technique or technology that is introduced, a system for determining its accuracy or validity must also be presented. Validation is a key component to providing feasible confidence that any new method is effective at reaching a viable solution, in this case a viable solution to the rogue application detection problem. Validation is not only comparing the results to what the expected result should be, but it is also comparing the results of our techniques and methodologies to other published methods.

For this research, the authors are comparing multiple data slices to determine their usefulness in the prediction process. To that end several performance values were used to measure and compare the performance of the experiments conducted in this research effort. These values include true positive rate (TPR), false positive rate (FPR), accuracy and precision. TPR, equation 4.1 below, also known as recall, is defined as the proportion of relevant applications that are retrieved, calculated by the ratio of the number of relevant retrieved applications to the total number of relevant applications that are in the data set (Salton & Buckley, 1988). In other words TPR is the ratio of actual positive instances that were correctly identified. FPR, equation 4.2 below, is the ratio of negative instances that were incorrectly identified. Accuracy, equation 4.3 below, is the ratio of the number of positive instances, either true positive or false positive, that were correct. Precision, equation 4.4 below, is defined as the proportion of retrieved applications that are relevant, calculated by the ratio of the number of relevant retrieved applications to the total number of retrieved applications (Salton & Buckley, 1988), or the ratio of predicted true positive instances that were identified correctly. All of these values are derived from information provided from the truth table. A truth table, also known as a confusion matrix, provides the actual and predicted classifications from the predictor. The following are the mathematical definitions of the performance formulas as well as the truth table (Table 4.1) where, a (true positive) is the number of rogue applications in the data set that were classified as rogue applications, b (false positive) is the number of benign applications in the data set that were classified as rogue applications, c (false negative) is the number of rogue applications in the data set that were classified as benign applications, and d (true negative) is the number of benign applications in the data set that were classified as benign applications (Schultz, Eskin, Zadok, & Stolfo, 2001). Below are the formulas for the four performance calculations that were used in this research effort for validation of the predicted results.

		Actual	
		Positive	**Negative**
Predicted	**Positive**	a	b
	Negative	c	d

Table 4.1: Definition of Truth Table

$$TPR = \frac{a}{a+c} \tag{4.1}$$

$$FPR = \frac{b}{b+d} \tag{4.2}$$

$$Accuracy = \frac{a+d}{a+b+c+d} \tag{4.3}$$

$$Precision = \frac{a}{a+b} \tag{4.4}$$

Using these calculated performance values this work can be validated and show that the proposed executable slicing method performed "better" than not using executable slicing. Better is defined in terms of absolute comparison of the validation methods presented above.

4.2. Instantiation Performance

As discussed above the pre-processing of the data set produced four data slices: whole, data, code and a combination of data and code. It is important to note that the results presented in this paper are just samples of the entire breadth of experiments that were performed on this data set.

Figures 4.1 and 4.2 depict a 3-gram experiment where the dimensionality was reduced to 500 from a range of ~500,000 to ~7,000,000, and a 4-gram experiment where the dimensionality was reduced to 1500 from a range of ~650,000 to ~13,000,000, respectively. The upper left quadrant contains the validation accuracy calculation results for the range of cosine similarity threshold values. By cosine similarity threshold value, we mean that two documents with a cosine similarity below this cut-off point are considered dissimilar. The lower left quadrant contains the TPR calculation while the lower right contains the calculations for precision. For each of the quadrants in the figure we are looking for the highest peak. For example in the upper left quadrant the highest peak would equate to the highest accuracy value for the range of threshold values. The upper right quadrant is defined as the FPR, for this value the lower value is the better result.

Beginning with the accuracy values (upper left quadrant) it can be seen that a 4% increase in total accuracy can be reached by using the slicing method (data – green line, code – blue line, combination – purple line) when compared to not using the slicing method (whole – red line). This value can be seen in Figure 4.1 when comparing the whole set (95%) to either the code or combination of code and data sets (both at 99%), each with threshold values of 0.25. Continuing to use those threshold values we can turn our attention to the TPR rate where the executable slicing provides a 2% increase. The precision is approximately the same but there is a 5% decrease in overall FPR. Similar results are seen as well in Figure 4.2.

This important result of the extracted instantiations outperforming the whole application can be seen throughout the experiment. This is a positive and significant step in that this type of slicing of applications to make a rogue application detection determination has not been published before at this level. By extracting these sections from an application, the data search space becomes much smaller and therefore allows for a faster detection time and a more accurate detection because of the ability to include more applications in the detection corpus. The slicing process adds a very minimal time to the preprocessing stage. Through the entire set of experiments the prediction processing time was decreased by as much five-fold, excluding the feature extraction phase.

When the results are examined from a data set instantiation viewpoint holding the remaining variables of *n*-gram size and dimensionality reduction size constant, it is clear that using the extracted data set instantiations provided a considerable increase in accuracy when compared to using the entire or whole application. It can be further derived that the code instantiation provides better results than the data instantiation. Even better results can be obtained by the combination of the data and code instantiations. However, with a minimal loss in overall prediction performance, about 1%, one could use just the code instantiation and gain in time performance.

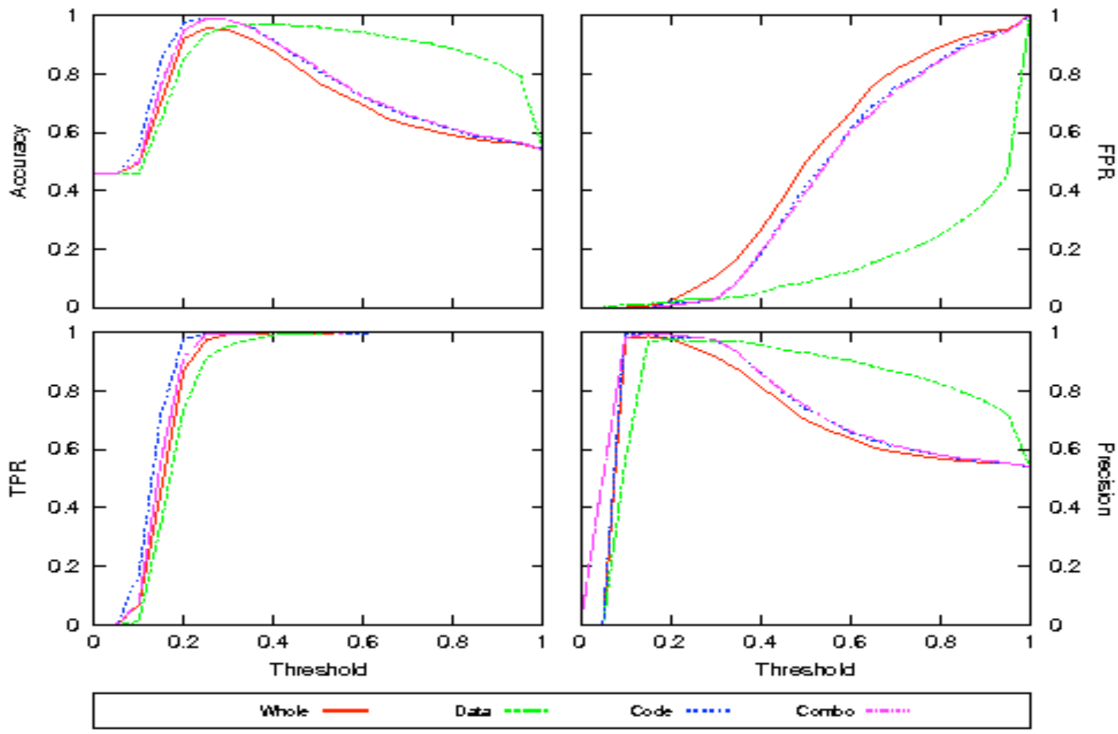

Figure 4.1: 3-gram, 500-features.

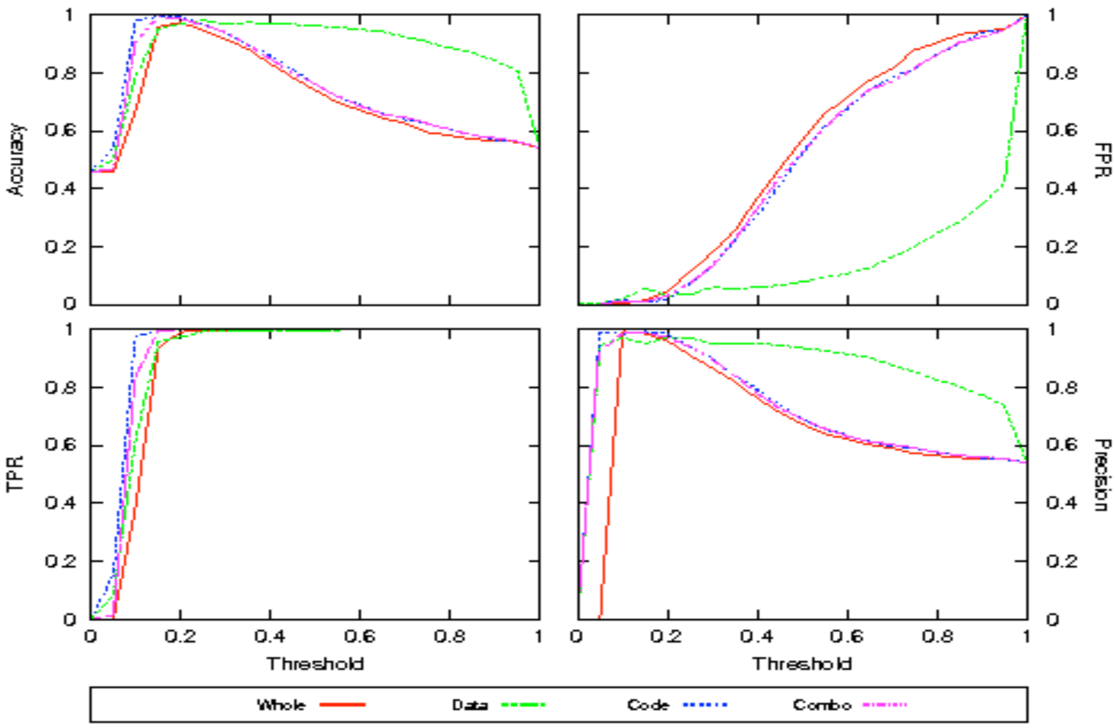

Figure 4.2: 4-gram, 1500-features.

5. Conclusions

The results support the idea that a better malicious software classifier can be created by applying an executable slicing technique as a pre-processing step to the technique of randomized projection. It has been shown through direct comparison that adding the executable slicing step generates results that have a higher accuracy value as well as better precision and recall values when compared to the randomized projection without using the executable slicing, pre-processing step.

There is no claim that this is a complete solution but rather a tool designed to fit into the security administrator's toolbox as a data point or first pass to help reduce the number of applications needing review. This potential reduction in the number of applications to sort through can provide an administrator or analyst with valuable time savings by not having to analyze applications that clearly do not contain rogue software. With more and more applications not being developed "in-house," this is a positive result for those responsible for providing secure solutions.

Future efforts for this research are to expand it with the addition of prediction algorithms from the data mining realm, for example decision trees. Also the author plans to investigate additional dimensionality reduction methods and techniques in order to further expand and enhance the analysis capability. It would be very interesting to determine if similar gains can be seen using executable slicing on other techniques. It is worth noting that this approach may be able to detect the slight variances in different instances of a polymorphic virus; however, this still needs to be tested. While detecting viruses which use self-encryption is out of the scope of this effort, it would be a notable path for future research. Additional research is also planned for determining the threshold values for the similarity algorithm. As seen in the results above, determining the key factors in choosing an optimal threshold value is crucial to gaining high confidence and to the success rate of the algorithm.

6. Acknowledgement

This material is based upon work supported by the U.S. Air Force, Air Force Research Laboratory under Award No. FA9550-10-1-0289. The authors would also like to thank Mr. Richard Libby, from Intel for equipment donation and Dr. Box Leangsuksun for high performance computing services.

References

Abou-Assaleh, T., Cercone, N., Keselj, V., & Sweidan, R. (2004a). *Detection of New Malicious Code Using N-grams Signatures.* Paper presented at the Proceedings of the 2nd Annual Conference on Privacy, Security and Trust, New Brunswick, Canada.

Abou-Assaleh, T., Cercone, N., Keselj, V., & Sweidan, R. (2004b). *N-gram-based Detection of New Malicious Code*. Paper presented at the Proceedings of the 28th Annual International Computer Software and Applications Conference, COMPSAC.

Atkison, T. (2009). *Applying Randomized Projection to aid Prediction Algorithms in Detecting High-Dimensional Rogue Applications.* Paper presented at the Proceedings of the 47th ACM Southeast Conference, Clemson, SC.

Baeza-Yates, R., & Ribeiro-Neto, B. (1999). *Modern Information Retrieval*. Harlow, England: Addison Wesley.

Bergeron, J., Debbabi, M., Desharnais, J., Erhioui, M. M., Lavoie, Y., Tawbi, N., et al. (2001). *Static Detection of Malicious Code in Executable Programs*. Paper presented at the Symposium on Requirements Engineering for Information Security,.

Bergeron, J., Debbabi, M., Erhioui, M. M., & Ktari, B. (1999). Static analysis of binary code to isolate malicious behaviors. *IEEE 8th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 1999.(WET ICE'99) Proceedings*, 184-189.

Bingham, E., & Mannila, H. (2001). Random projection in dimensionality reduction: applications to image and text data. *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 245-250.

Christodorescu, M., & Jha, S. (2003). Static analysis of executables to detect malicious patterns. *Proceedings of the 12th Conference on USENIX Security Symposium, 12*, 12.

Dasgupta, S., & Gupta, A. (1999). *An elementary proof of the Johnson-Lindenstrauss Lemma*. Berkley, California, USA: International Computer Science Institute.

Haventools Software. (2009). Heaventools: PE Explorer. Retrieved 14 March 2009, from http://www.heaventools.net

Henchiri, O., & Japkowicz, N. (2006). A Feature Selection and Evaluation Scheme for Computer Virus Detection. *6th International Conference on Data Mining, ICDM'06*, 891-895.

Food and Drug Administration. (8 September 2010). Infusion Pump Software Safety Research at FDA. Retrieved 9 February 2011, from www.fda.gov/MedicalDevices/ProductsandMedicalProcedures/GeneralHospitalDevicesandSupplies/InfusionPumps/ucm202511.htm

Johnson, W. B., & Lindenstrauss, J. (1984). Extensions of Lipschitz mappings into a Hilbert space. *Contemporary Mathematics, 26*, 189-206.

Jovanovic, N., Kruegel, C., & Kirda, E. (2006). Pixy: A Static Analysis Tool for Extracting Web Application Vulnerabilities. *IEEE Symposium on Security and Privacy*.

Kang, M. G., Poosankam, P., & Yin, H. (2007). *Renovo: A Hidden Code Extractor for Packed Executables*. Paper presented at the Proceedings of the 2007 ACM Workshop on Recurring Malcode.

Kaski, S. (1998). Dimensionality Reduction by Random Mapping: Fast Similarity Computation for Clustering. *The 1998 IEEE International Joint Conference on Neural Networks. IEEE World Congress on Computational Intelligence, 1*.

Kephart, J. O., Sorkin, G. B., Arnold, W. C., Chess, D. M., Tesauro, G. J., & White, S. R. (1995). *Biologically inspired defenses against computer viruses.* Paper presented at the Proceedings of the 14th International Joint Conference on Artificial Intellgence, San Francisco, CA.

Kurimo, M. (1999). Indexing Audio Documents by using Latent Semantic Analysis and SOM. *Kohonen Maps*, 363-374.

Lin, J., & Gunopulos, D. (2003, May). *Dimensionality reduction by random projection and latent semantic indexing.* Paper presented at the Proceedings of the Text Mining Workshop at the 3rd SIAM International Conference on Data Mining.

Liu, N., Zhang, B., Yan, J., Yang, Q., Yan, S., Chen, Z., et al. (2004). *Learning Similarity Measures in Non-Orthogonal Space*. Paper presented at the Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management.

Mannila, H., & Seppänen, J. K. (2001). Finding similar situations in sequences of events. *1st SIAM International Conference on Data Mining*.

Marceau, C. (2000). *Characterizing the Behavior of a Program Using Multiple-Length N-grams*. Paper presented at the Proceedings of the 2000 Workshop on New Security Paradigms,.

McGraw, G., & Morrisett, G. (2000). Attacking malicious code: a report to the Infosec Research Council. *IEEE Software, 17*(5), 33-41.

Papadimitriou, C. H., Raghavan, P., Tamaki, H., & Vempala, S. (2000). Latent Semantic Indexing: A Probabilistic Analysis. *Journal of Computer and System Sciences, 61*(2), 217-235.

Perdisci, R., Lanzi, A., & Lee, W. (2008). Classification of Packed Executables for Accurate Computer Virus Detection. *Pattern Recognition Letters, 29*(14), 1941 - 1946.

Reddy, D. K. S., & Pujari, A. K. (2006). N-gram analysis for computer virus detection. *Journal in Computer Virology, 2*(3), 231 - 239.

Salton, G., & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Information Processing and Management: an International Journal, 24*(5), 513-523.

Salton, G., Wong, A., & Yang, C. S. (1975). A vector space model for automatic indexing. *Communications of the ACM, 18*(11), 613-620.

Schultz, M., Eskin, E., Zadok, E., & Stolfo, S. (2001). *Data Mining Methods for Detection of New Malicious Executables*. Paper presented at the Proceedings of the IEEE Symposium on Security and Privacy,.

Singhal, A. (2001). Modern Information Retrieval: A Brief Overview. *Bulletin of the Technical Committee on Data Engineering, 24*(4), 35 - 43.

Vempala, S. S. (2004). *The Random Projection Method*: American Mathematical Society.

Bios

Jan Durand is currently a graduate student in the Computer Science Department at Louisiana Tech University. He received his B.S. degree in computer science from Grambling State University in 2010. His current research interests include the detection of malicious applications via static analysis and data mining.

Juan Carlos Flores is currently a graduate student in the Computer Science Department at Louisiana Tech University. He received his B.S. degree in computer engineering from the University of Houston Clear Lake in 2010. His research interests are in the areas of virtualization technologies, network security and digital forensics.

Nicholas Kraft is currently an assistant professor in the Department of Computer Science at the University of Alabama. He received his Ph.D. degree from the Clemson University in 2007. His research interests include reverse engineering, program comprehension, grammar engineering and software evolution.

Randy Smith is currently an associate professor in the Department of Computer Science at the University of Alabama. He received his Ph.D. degree from the University of Alabama in 1998. His current research interests include software process measurement and data mining.

Travis Atkison is currently an assistant professor in the Computer Science Department at Louisiana Tech University. He received his Ph.D. degree from Mississippi State University in 2009. His current research interests include computer security, computer forensics, software engineering and information retrieval.