Applying Static Analysis to High-Dimensional Malicious Application Detection

Sean Semple, Stanislav Ponomarev, Jan Durand, Travis Atkison
Louisiana Tech University
Ruston, LA 71270
{sms079, spo013, jrd037, atkison} @latech.edu

ABSTRACT

Signature based anti-virus systems inherently restrict the detection of new and previously unknown types of malicious attacks. To that end researchers are searching for methodologies to combat this problem. One potential method is the use of static application analysis. Using this methodology the applications are not executed to determine whether or not they contain malicious functionality. This paper presents a static application analysis methodology using the information retrieval technique of *n*-gram analysis and the dimensionality reduction techniques of randomized projection and mutual information to create a malicious application detection model. For this effort, a data set was extracted from Microsoft Windows applications that were either benign or possessed malicious functionality. Dimensionality and prediction methodology was then applied. Initial results show promise when comparing the prediction to expected outcomes. In one performance evaluation, the Boosted J48 algorithm achieved an accuracy of 99.08%.

General Terms

Algorithms, Performance, Reliability, Experimentation, Security, Verification.

Keywords

Static analysis, malicious software detection, information retrieval, *n*-gram analysis, randomized projection, mutual information.

1. INTRODUCTION

With more than thirty years of propagation, computer viruses have become increasingly mainstream. As the internet evolved into a tool used by millions in everyday life, malicious software has taken advantage of this effective medium to infect victims' machines. Malware is a catch-all phrase used to refer to any program that is designed to "harm or subvert a system's intended functionality" [1] and falls under several categories including viruses, worms, and Trojans. Most consumers and companies commonly refer to any malicious application as a virus, though not all malicious software are viruses.

Traditionally, signature based detection is used to detect malicious software. This involves comparing the byte sequence of a suspected malicious application to the signatures of known threats stored in a virus signature database. If part of the application matches a signature in the database then the application is flagged as malicious. Signature-based detection is "effective when the virus code does not change significantly over time" [2]. In fact, a single virus signature can be used to match multiple variants of a particular virus if they all contain the base signature code [2]. Signature-based detection is a very practical method of malware detection and became popular due to its ease of use and low false-positive rates [3]. However, this approach is fundamentally limited to detecting only known threats. That is, signature-based detection tools are incapable of detecting new and previously unknown threats which do not contain any known signatures. This limitation leaves signature-based anti-virus tools completely ineffective against "zero-day" viruses until their signature database, or virus definitions, have been updated with the signatures of the new threats. As a result, signature-based detection tools require frequent updates of the signature database to keep up with new malware. This process can be very time consuming as suspect malicious applications must first be identified before they can be analyzed. If a malicious intent is found, the analyst must then find a unique signature that can be used to detect it in the wild.

Apart from being difficult to identify signatures, signature based detection can be easily bypassed by code obfuscation. Code obfuscation involves making changes to the syntactic properties of the malware byte sequence while preserving program behavior. Christodorescu et al. observed in an experiment that "three commercial virus scanners could be subverted by very simple obfuscation transformations." [2] Some more advanced viruses use several obfuscation methods to avoid signature identification and are referred to as polymorphic or metamorphic viruses. These viruses are equipped with mutation engines which mutate their code, while retaining behavior [2]. The techniques employed by metamorphic viruses are so powerful that they "make virus detection using search strings virtually impossible." [4]

Modern antivirus programs are employing new methods of malware detection along with signature matching in order to keep up with the rapidly evolving world of malware. Some antivirus programs are using a technique called heuristic analysis which monitors the behavior of programs to determine unknown malware. The static heuristic analysis method is the focus of this research. Unlike the dynamic analysis method which must run the application in question, the static analysis method determines behavior by examining the source code.

The purpose of this paper is to help answer the important question of "How do we know that the software we are using is doing what we ask it to do and nothing more?" The main focus of this effort is to design and develop a static analysis method to assist in the detection of malicious applications by combining techniques from both the information retrieval and data mining fields, with emphasis on Trojan horses.

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

2. BACKGROUND

Malicious application detection is a popular topic and has gained a significant body of research. In essence though, all the different research ventures can be categorized as either dynamic or static analysis. Each approach has advantages and disadvantages. While dynamic analysis is more robust at handling code obfuscation because it monitors semantic characteristics, it can easily be bypassed by programs that only exhibit malicious behavior at certain times. It is also possible for a malicious application to gain access to and compromise the host machine from the virtual environment, as demonstrated by Ed Skoudis and Tom Liston from IntelGuardians. This negates any 'security' a researcher had by using the virtual environment for malicious detection. Static analysis allows for examination of malicious code without putting the system in danger of infection. It can also consider all possible execution paths, thus allowing it to identify malicious behavior that could evade dynamic analysis. However, static analysis may be vulnerable to subversion by polymorphic and metamorphic techniques as it examines the syntactic properties of malware.

There are many static analysis methods available for the examination of data. One of the earlier attempts of using static analysis to detect malicious applications produced the Malicious Code Filter (MCF) [25]. Lo et al. used what they called "tell-tale" signs, which were manually identified in malicious programs, to filter out other suspicious programs which possessed the same signs [2]. This research introduced the concept of tell-tale signs, but did not show any results to validate the effectiveness of the approach. Below are descriptions of major research efforts that use various methods of static analysis to solve this detection problem.

2.1 Information Retrieval

Information retrieval is a static analysis method available for the examination of data. It can be thought of as the "part of computer science which studies the retrieval of information (not data) from a collection of written documents." [10] This information is then used to "satisfy a user's information need." [10] It can be thought of as searching through a set of documents, called the corpus, to find information that is relatable to a given entity, called the query. When dealing with information retrieval, the features extracted play a crucial role. The technique of *n*-gram analysis has proven to be a successful and valuable tool for this. Several research efforts focus on the classification of malicious applications with the use of *n*-gram analysis [5, 6, 8, 9, 23, 24]. An *n*-gram is any substring of length *n* [10]. Since *n*-grams overlap, they do not just capture statistics about substrings of length *n*, but also capture frequencies of longer substrings [6].

There have been many efforts to use the information retrieval technique of *n*-gram analysis as a feature generator. Baeza-Yates and Ribeiro-Neto applied this technique to the field of document authorship to determine whether William Shakespeare wrote all the works attributed to him. Henchiri et. al. [7] and Abou-Assaleh et. al. [5, 6] both use the Common *N*-Gram (CNG) analysis method, which uses the most frequent *n*-grams to represent a class, to detect malicious applications. Henchiri further limits the number of features by imposing a "hierarchical feature selection process". [7] Marceau [8] suggests an interesting modification on the problem of using *n*-grams as features by having "multiple-length" grams instead of the tradition single *n*-length gram. Marceau does this by first creating and then compacting a suffix tree to a directed acyclic graph (DAG) [8]. Reddy et. al. [9] develop their own unique *n*-gram feature selection measure called, 'class-wise document frequency'.

2.2 Dimensionality Reduction

Malware detection using the static analysis method of information retrieval invariably creates data in extremely high dimensions. Computing in extremely high dimensions creates a problem known as the "curse of dimensionality." [28] Due to this "curse of dimensionality", techniques must be introduced to reduce the data size to allow for proper analysis. Two such dimensionality reduction techniques are feature extraction and feature selection. The feature selection technique of mutual information can be used to select a subset of the most relevant *n*-gram features. In addition to using mutual information, the feature extraction technique of random projection can be used to compare the performance between the two dimensionality reduction methods. Unlike feature selection, feature extraction transforms the original subset, either linearly or non-linearly, to a much smaller feature set while retaining the most relevant properties. Thus, feature extraction techniques consider the predictive information provided by all the features in the original feature set instead of just a select few.

There have been efforts [11, 12, 22] that use randomized projection techniques for dimensionality reduction. Hegedus et. al. [21] applied random projection techniques to the topic of malware detection through dynamic feature extraction. The authors proposed a way to detect files with the highest probability of being false negatives (malicious files that were classified as benign).

Minnila et. al. [11] used 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 human analyst previous situations that resemble the current one" [11] 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 et. al. [12] applies 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.

There have been other efforts that use mutual information techniques for dimensionality reduction. Kolter [23, 24] introduced the technique of mutual information as a processing step to reduce the dimensionality of the program feature vectors. Kolter used the information retrieval technique of *n*-gram analysis to create binary feature vectors of malicious and benign programs, which were used with machine learning algorithms to unknown program instances.

2.3 Static Analysis Benefits

Static analysis, in this context, conventionally has been used as a method to comb through source code to assist in identifying and detecting malicious applications. Many examples of static analysis exist as well as static analysis overviews, namely [13 - 16]. More recently researchers [17 - 19] have attempted to apply these techniques to executable binaries with some success. These research efforts are looking at many aspects of a binary including detecting obfuscation [18] and detecting mimicry attacks [19].

Even though there are some advantages to using a dynamic analysis methodology which is normally used by running the possibly malicious application in a controlled or restricted environment, there are many dangers involved. Because of these dangers, most researchers will execute these potentially malicious applications in virtual environments, such as VMWare or Virtual Box. However, many malicious application writers have designed checks that can detect whether or not the malicious application is being executed inside of a virtual environment. If a virtual environment is detected, the malicious code will skip over the harmful parts of the application, therefore, going unnoticed by the detection system. Even with the potential pitfalls, dynamic analysis does provide real-time run-time analysis capabilities that static analysis cannot provide.

Static analysis provides many advantages when attacking a problem as diverse and difficult as malicious application detection. Since the potential malicious applications are not executed there is no chance for accidental infestation as is the case with dynamic analysis. Though this is a positive, due to undecidability it is “impossible to certify statically that certain properties hold.” [20] That aside, static analysis of potential malicious applications can be accomplished without the run-time overhead associated with dynamic analysis techniques. By using static analysis, an analyst can discover all possible execution paths. Because of the ability to analyze applications without the need to execute them, this research effort has attacked the malicious application detection problem using static analysis techniques. This work is done with the understanding that the solutions created here are only one piece of the larger solution that may include dynamic analysis or other methods.

3. EXPERIMENT

For these experiments, a set of software tools was created along with the Waikato Environment for Knowledge Analysis (WEKA) [26] to follow the static analysis methods of information retrieval in conjunction with mutual information and randomized projection. A description of the data sets used in the experiments is described below.

3.1 Data Set

The data set used for these experiments consisted of 1622 Windows formatted binary executable files. Of these, 303 were extracted from a fresh installation of Microsoft Windows XP, 406 were extracted from a fresh installation of Microsoft Windows Vista, and 78 were extracted from a fresh installation of Microsoft Windows 7. All data sets were obtained by installing the operating system in a virtual environment which was not connected to the internet to ensure no malicious infiltration. The remaining 835 files were malicious Trojan horse applications that were downloaded from various websites on the internet including http://www.trojanfrance.com and http://vx.netlux.org.

3.2 Classification Algorithms

This subsection describes the classification algorithms used in the performance evaluation of random projection as an effective dimensionality reduction technique. All machine learning algorithms were trained and tested using the WEKA machine learning software suite, with 10-fold cross validation on the training data set. Seven separate classification algorithms were applied in the context of this experiment. Instance Based Classifier, Naïve Bayes Classifier, Decision Tree, and Support Vector Machines (SVMs) were the classifiers used. The instance-based learner uses a *k*-nearest-neighbor algorithm which classifies an instance by a majority vote of the labels of the *k* most similar instances in the training set. Naïve Bayes classifiers use prior probability of a class and the conditional probability of each feature attribute for the specified class in order to determine the probability that an unknown instance belongs to a particular class. Decision trees consider all the instances in the training set and select a feature attribute which best splits the data set into its respective classes. The attribute that was previously selected is then removed from future consideration and the process is repeated recursively on each data subset. The last attribute split creates the leaf nodes which are labeled based on a majority vote of the class labels of its elements. For attribute selection, the J48 algorithm is used. SVMs produce a binary linear classifier which is able to separate a data set into two distinct classes, the positive and negative class. The method works by mapping feature vectors of data instances into a higher dimension so that the two classes of instances can be separated by a hyper plane. Classification is performed by mapping an unknown instance into a higher feature dimension and labeling it based on which side of the hyper plane it appears. Along with these, each classifier except the Instance Based Classifier, due to computation expense, was boosted using the Adaptive Boosting method implemented in WEKA. These new boosted classifiers were also used in the experiments. Different classification algorithms provide the researcher with a broader scope of results from the same amount of data.

3.3 Experimental Design

Six different *n*-gram sizes ranging from 2 to 7 were used to produce multiple feature vectors representing the same executable file. A binary feature weighting scheme was used for this effort, whereby each unique *n*-gram was considered a feature and feature vectors were created for each document in the data set by assigning a ‘1’ to a vector dimension attribute if the corresponding *n*-gram was present in the executable or ‘0’ if it was not. Due to the large number of features produced through *n*-gram analysis, a reduced corpus had to be created. One set was reduced using the feature selection technique of mutual information and the other set was reduced using the feature extraction technique of randomized projection. For each set, three reduced versions were created by reducing each vector to a feature set size of 500, 1000, and 1500 using the respective dimensionality reduction technique. Various training algorithms were then used to classify data as either malicious or benign based off of the data given to it.

4. RESULTS

Using the methods and techniques described above, a software tool was created which received the raw executable files as inputs and produced corpuses of mutual information and random projection reduced *n*-gram feature vectors in a format consumable by WEKA. WEKA was then used to train and test classifiers using the various classification algorithms.

In order to make a comparison of performance values, a system for determining accuracy had to be introduced. These values include true positive rate (TPR), false positive rate (FPR), area under the curve (AUC) and accuracy. TPR, also known as recall, "is the proportion of retrieved applications that are relevant, measured by the ratio of the number of relevant retrieved applications to the total number of retrieved applications," [27] or the ratio of predicted true positive instances that were identified correctly. The TPR represents the number of malicious instances classified as malicious; the FPR represents the number of benign instances classified as malicious; the AUC measure represents the area under the receiver operating characteristic curve generated by the classifier; the accuracy represents the number of instances correctly classified by the classifier.

Presented below is a sample of the results of the experimental runs. Figure 1 illustrates the average accuracies of each classifier with respect to *n*-gram size. The total accuracy was greatest with an *n* = 3 and was lowest with values at the end of the spectrum. At *n* = 4, the accuracy dropped drastically but leveled out slightly at *n* = 5. Figure 2 illustrates the average accuracy of random projection classifiers against *n*-gram size. It shows that with an *n* = 4 gram size, peak accuracy can be achieved. Similarly to the mutual information classifier, after achieving peak performance, average accuracy dropped at *n* = 5 by over 1%. Using features with an *n*-gram size of 2 or 7 generally produced less accurate classifiers. This is possibly due to some important information not being provided by an *n*-gram of size 2, and extraneous information which could impair the classifiers with an *n*-gram of size 7. Results show that *n*-grams of length 3 and 4 were most accurate, particularly *n*-gram of length 3 for the mutual information trained classifiers and *n*-grams of length 4 for random projection trained classifiers. With an *n*-gram of length 3 used on the mutual information classifiers with 1500 features, the best results show a significant increase in detection accuracy over random projection and previous research efforts. Table 1 shows the performance results for a data set reduced by mutual information with a 3-gram window with 1500 features. This table shows how the static analysis technique of n-gram analysis yields extremely high true positive rates for both the boosted J48 classifier and SVM classifier. All classifiers except the Naïve Bayes classifier achieved a TPR of over 0.98. Peak accuracy was achieved with the SVM classifiers at 99.26%. The SVM classifiers also had the lowest FPR of 0.009. Though mutual information trained classifiers produced the most accurate results with 3-grams, random projection trained classifiers were most accurate with 4-grams. The SVM and Boosted SVM classifiers were able to accurately label malicious applications 99.6% of the time while labeling benign applications as malicious only 1.6% of the time. Both classifiers resulted with an accuracy of 99.08%.

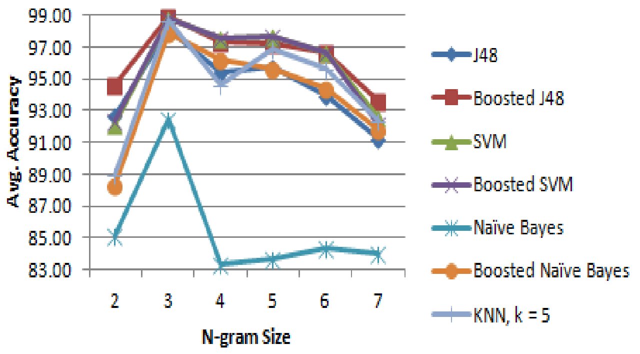

Figure 1: Average accuracy of each mutual information classifier vs. *n*-gram size.

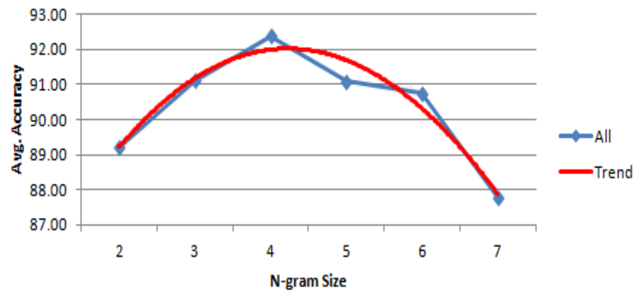

Figure 2: Average accuracy of all random projection classifiers vs. *n*-gram size.

Table 1: Performance results for mutual information, 3-grams, 1500 features.

Algorithm	TPR	FPR	AUC	Acc. %
J48	0.984	0.019	0.983	98.27
Boosted J48	0.99	0.01	0.996	99.01
SVM	0.994	0.009	0.992	99.26
Boosted SVM	0.994	0.009	0.992	99.26
Naïve Bayes	0.931	0.096	0.959	91.98
Boosted Naïve Bayes	0.994	0.019	0.996	98.83
KNN, k = 5	0.993	0.026	0.997	98.46

5. Conclusions

The results of these experiments indicate that using static analysis techniques along with classification algorithms and information retrieval techniques has merit and can be used to determine if an application may contain malicious functionality. These experiments have shown that the results are accurate and better than some suggested solutions in literature.

This is by no means a perfect solution yet, but a tool which when combined with other tools could yield a better way to defend data and develop more accurate prediction software. This research could be used by an analyst to narrow down the amount of potentially malicious applications, saving valuable time. This technique of static analysis is a useful pathway for research not only in malicious application removal, but other applications where large data sets are compiled.

Future efforts for this research include expanding information retrieval techniques and creating new ways to analyze data. New dimensionality reduction techniques will also be explored in order for the researchers to gain the highest result accuracy. Determining the right correlation of techniques is key to a high success rate in the experimentation with static analysis.

6. ACKNOWLEDGMENTS

This material is based upon work supported by the U.S. Air Force, Air Force Research Laboratory under Award No. FA9550-10-1-0289.

7. REFERENCES

[1] G. McGraw and G. Morisett, "Attacking Malicious Code: A Report to the Infosec Research Council," IEEE Software, vol. 17, no. 5, pp. 33-41, Sep/Oct 2000.

[2] M. Christodorescu and S. Jha, "Static Analysis of Executables to Detect Malicious Patterns," in Proceedings of the 12th Conference on USENIX Security Symposium, Berkeley, CA, USA, 2003, p. 12.

[3] M. Christodorescu, S. Jha, M. D. Preda, and S. Debray, "A Semantics-Based Approach to Malware Detection," in Proceedings of the 34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Nice, France, Jan. 2007, pp. 377-388.

[4] E. Konstantinou and S. Wolthusen, "Metamorphic Virus: Analysis and Detection," Information Security Group, Royal Holloway, University of London, Technical Report RHUL-MA-2008-02, 2008.

[5] T. Abou-Assaleh, N. Cercone, V. Keselj, and R. Sweidan, "Detection of New Malicious Code Using *N*-grams Signatures," Proceedings of the 2nd Annual Conference on Privacy, Security and Trust, New Brunswick, Canada, 2004, pp. 193 - 196.

[6] T. Abou-Assaleh, N. Cercone, V. Keselj, and R. Sweidan, "*N*-gram-based Detection of New Malicious Code," Proceedings of the 28th Annual International Computer Software and Applications Conference, COMPSAC. vol. 2, 2004.

[7] O. Henchiri and N. Japkowicz, "A Feature Selection and Evaluation Scheme for Computer Virus Detection," 6th International Conference on Data Mining, ICDM'06, 2006, pp. 891-895.

[8] C. Marceau, "Characterizing the Behavior of a Program Using Multiple-Length *N*-grams," Proceedings of the 2000 Workshop on New Security Paradigms, Ballycotton, County Cork, Ireland: ACM, 2000.

[9] D. K. S. Reddy and A. K. Pujari, "*N*-gram analysis for computer virus detection," *Journal in Computer Virology,* vol. 2, no. 3, 2006, pp. 231 - 239.

[10] R. Baeza-Yates and B. Ribeiro-Neto, Modern Information Retrieval, Harlow, England, Addison Wesley, 1999.

[11] H. Mannila and J. K. Seppänen, "Finding similar situations in sequences of events," 1st SIAM International Conference on Data Mining, 2001,

[12] E. Bingham and H. Mannila, "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,* 2001, pp. 245-250.

[13] P. Chandra, B. Chess, and J. Steven, "Putting the tools to work: How to succeed with source code analysis," IEEE Security & Privacy, vol. 4, no. 3, 2006, pp. 80-83.

[14] D. Hovemeyer and W. Pugh, "Finding bugs is easy," *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications* Vancouver, BC, CANADA: ACM, 2004.

[15] D. Wagner and R. Dean, "Intrusion detection via static analysis," 2001, pp. 156-168.

[16] Y. Zhang, J. Rilling, and V. Haarslev, "An Ontology-based Approach to Software Comprehension-Reasoning about Security Concerns," 2006.

[17] J. Bergeron, M. Debbabi, M. M. Erhioui, and B. Ktari, "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,* 1999, pp. 184-189.

[18] J. Lin and D. Gunopulos, "Dimensionality reduction by random projection and latent semantic indexing," Proceedings of the Text Mining Workshop at the 3rd SIAM International Conference on Data Mining, 2003.

[19] C. Kruegel, E. Kirda, D. Mutz, W. Robertson, and G. Vigna, "Automating mimicry attacks using static binary analysis,"

[20] J. Bergeron, M. Debbabi, J. Desharnais, M. M. Erhioui, Y. Lavoie, N. Tawbi, and M. Erhioui, "Static Detection of Malicious Code in Executable Programs," *Symposium on Requirements Engineering for Information Security,* Indianapolis, IN, 2001.

[21] J. Hegedus, Y. Miche, A. Ilin, and A. Lendasse, 2011, "Methodology for Behavioral-based Malware Analysis and Detection using Random Projections and K-Nearest Neighbors Classifiers," Hainan, 2011.

[22] C. H. Papadimitriou, P. Raghavan, H. Tamaki, and S. Vempala, "Latent Semantic Indexing: A Probabilistic Analysis," *Journal of Computer and System Sciences,* vol. 61, no. 2, 2000, pp. 217-235.

[23] J. Z. Kolter and M. A. Maloof, "Learning to Detect Malicious Executables in the Wild," in Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, WA, Aug. 2004, pp. 470-478.

[24] J. Z. Kolter and M. A. Maloof, "Learning to Detect and Classify Malicious Executables in the Wild," The Journal of Machine Learning Research,

[25] R. W. Lo, K. N. Levit, and R. A. Olsson, "MCF: A Malicious Code Filter," *Computers & Security*, vol. 14,1995.

[26] I. H. Witten and E. Frank, Data Mining: Practical Machine Learning Tools and Techniques, 2nd ed. San Francisco, CA, USA: Morgan Kaufmann, 2005.

[27] G. Salton, A. Wong, and C. S. Yang, "A vector space model for automatic indexing," Communications of the ACM, vol. 18, no. 11, 1975, pp. 613-620.

[28] R. Bellman, Adaptive Control Processes: A Guided Tour.: Princeton University Press, 1961.