N-Gram Analysis in SVM Training Phase Reduction Using Dataset Feature Filtering for Malware Detection

Pagidimarri Venu[1], **Dasu Vaman Ravi Prasad**[2]

[1]Computer science, CVSR College of Engineering, Venkatapur, RR dist, India

[2]Computer Science and Engineering, CVSR College Of Engineering, Venkatapur, RR dist, India

Abstract: An n-gram is a sub-sequence of n items from a given sequence. Various areas of statistical natural language processing and genetic sequence analysis are using N-gram Analysis. In which sequence analysis is the process of comparing the sequence or series of attributes in order to find the similarity. Malicious software that is designed by attackers for disturbing computers is called as malware. The principal belong to the same family of malware eventhough Malware variants will have distinct byte level representations. The byte level content is different because small changes to the malware source code can result in significantly different compiled object code. In which programs are used as operational code (opcode) density histograms obtained through dynamic analysis. The process of testing and evaluation of application or a program during running time is called as dynamic analysis. A SVM is used for classification or regression problems. Kernel trick is a technique by SVM to transform your data and then based on these transformations it finds an optimal boundary between the possible outputs. We employ static analysis to classify malware which is identified a prefilter stage using hex values of files, that can reduce the feature set and therefore reduce the training effort. The result shows that the relationships between features are complex and simple statistics filtering approaches do not provide a Practical approach. One of the approaches, hex decimal based produces a suitable filter. The entire system will be implemented in WEKA tool.

Keywords: n-gram analysis, malware variants, kernel trick, SVM, WEKA tool.

1. Introduction

Recent years malware has been growing rapidly, with signature detection and monitoring suspected code for known security vulnerabilities becoming ineffective and intractable. In response, researchers need to adopt new detection approaches that out manoeuvre the different attack vectors and obfuscation methods employed by the malware writers. Detection approaches that use the host environment's native opcodes at run-time will circumvent many of the malware writers' attempts to evade detection. One such approach, as proposed in this paper, is the analysis of opcode density features using supervised learning machines performed on features obtained from run-time traces. In future research we intend to expand the detection methods by investigating N-gram size, which will dramatically increase the number of features. With this anticipated explosion of features we have chosen to investigate methods to prune irrelevant features. While Principle Component Analysis (PCA) is a popular method to reduce features in subspace, this paper aims to identify feature reduction in the original dataset space. For large datasets, or costly (computation) distance functions, the training process associated with learning machines can become immense. Thus, the feature explosion that occurs with N-grams for large values of N needs to be addressed. This paper investigates three approaches to filtering out irrelevant features and starts, in Section II, with a discussion on related research. In Section III, the experiments are placed into context with an overview of the experimental approach. Section IV specifies the environment used to capture the dataset and introduces anti-analysis approaches taken by malware writers. This is followed, in Section V, with an explanation of how the dataset is created. The Support Vector Machine (SVM) is introduced in Section VI and describes the creation of a reference model that is used to validate the successfulness of the subsequent filter experiments. Three filters: Firstly, a simple hypotheses test is considered to determine the likelihood that the benign and malicious dataset do not belong to the same distribution; secondly, an in-depth look at the distribution by calculating the area of intersect between the benign and malicious distributions; and finally, a look at the projection of the dataset into a subspace using eigen values. The results and key characteristics recorded during these experiments. Finally, the paper by comparing the results with other research and details future work that will be carried out as part of this research.

2. Literature Survey

Extensive research has been undertaken into the detection of malicious code using both static and dynamic analysis. Malware research can be categorized, not only in terms of static and dynamic analysis, but also in how the information is processed after it is captured. Popular research methods include: Control Flow Graphs (CFG) for both course and fine grain analysis, state machines to model system behavior, the mapping of stack operations and N-gram analysis. Bilar [2] used static analysis to obtain opcode distributions from PE files that could be used to identify polymorphic and metaphoric malware. Bilar's findings show that many prevalent opcodes (*mov, push, call*, etc.) did not make good indicators of malware. Lesser frequent opcodes such *ja, adc, sub, inc* and *add* proved to be better indicators of malware. In other research, Bilar [3] compared the statically generated CFG of benign and malicious code. Their findings showed a difference in the basic block count for benign and malicious code. Bilar concluded that malicious code has a lower basic block count, and implying a simpler structure: Less interaction, fewer branches and less functionality. N-grams are based on a signature approach that relies on small sequences of strings or byte codes that are used to detect malware. Santos *et al.* [4] demonstrated that n-gram signatures could be used to detect unknown malware. The experiment extracted code and text fragments from a large database of program executions to form signatures that are classified using machine learning methods.

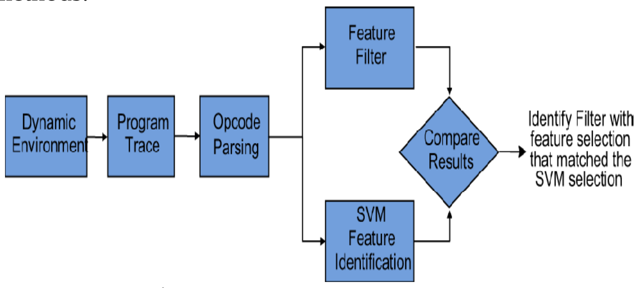

Dynamic Environment
Program Trace
Opcode Parsing
Feature Filter
Compare Results
SVM Feature Identification
Identify Filter with feature selection that matched the SVM selection

Figure 1: Experiment overview.

The motivation for this research is to reduce the computational overhead required when N-gram analysis is performed on low-level fine grain data. Therefore, developing a lightweight filter that will reduce the number of features to be processed will in turn reduce the computational overhead; thus making the training phase of the SVM approach a viable solution for N-gram analysis where large feature sets are generated. Fig. 1 illustrates an overview of the experimental approach taken in this paper. The programs under investigation are run in a test environment with a debug tool monitoring the runtime opcodes. After completion, the data is parsed into opcode histograms and after some conditioning the dataset is passed to the SVM to construct a reference model. It is constructed by configuring the SVM to perform an exhaustive search by traversing through all the features, searching for those opcodes that have a positive impact on the classification of benign and malicious software. To evaluate the various filtering algorithms, each filter processes the original dataset in an attempt to reproduce the same reference model produced by the SVM.

3. System Overview

The motivation for this research is to reduce the computational overhead required when N-gram analysis is performed on low-level fine grain data. Therefore, developing a lightweight filter that will reduce the number of features to be processed will in turn reduce the computational overhead; thus making the training phase of the SVM approach a viable solution for N-gram analysis where large feature sets are generated. Fig. 1 illustrates an overview of the experimental approach taken in this paper. The programs under investigation are run in a test environment with a debug tool monitoring the runtime opcodes. After completion, the data is parsed into opcode histograms and after some conditioning the dataset is passed to the SVM to construct a reference model. The reference model is constructed by configuring the SVM to perform an exhaustive search by traversing through all the features, searching for those opcodes that have a positive impact on the classification of benign and malicious software. To evaluate the various filtering algorithms, each filter processes the original dataset in an attempt to reproduce the same reference model produced by the SVM.

4. Dataset Creation

Operational Codes (Opcodes) are machine language instructions that perform CPU operations on operands such as arithmetic, memory/data manipulation, logical operations and program flow control. There are 15 opcodes directly referred to in this paper, which are grouped as follows:

1) Arithmetic operations—*add* , *adc* (add with carry flag), *inc*, *sub*;
2) Memory manipulation—*lea* (load effective address), *mov*, *pop*, *push* (retrieve and place data onto a stack);
3) Logical operations—*xor* (exclusive OR);
4) Program flow control—*call* (jump to a function), *ret* (return from function) *cmp* (compare data), *ja*, *je* (jump if a condition is met); *rep* (a prefix that repeats the specified operation).

The dataset is constructed by representing each executable file as a set of opcode density histograms obtained from runtime traces. Note that the operands associated with each opcode are omitted and that only the opcodes are recorded. Classification tasks involve separating data into training and test data. Each training-set instance is assigned a target value/label i.e., benign or malicious. The goal of the SVM is to construct a model that predicts the target values of the test data. There are 260 benign Windows XP files taken from the 'Program Files' directory (training files 230, validation files 30). There are 350 malware files (training files 310, validation files 40) which are malicious windows executable files downloaded from Vxheaven website (this website is no longer available) and consists of a range of malicious activities such as: back-door downloaders, system attack, fake alert/warnings, Ad-Aware, information stealer. To ensure that Ollydbg correctly unpacked and ran the malware, samples were restricted to programs that ollydbg correctly identified as packed or encrypted. The malware samples were run for 3 minutes ensuring that not only the loading and unpacking phases were recorded but

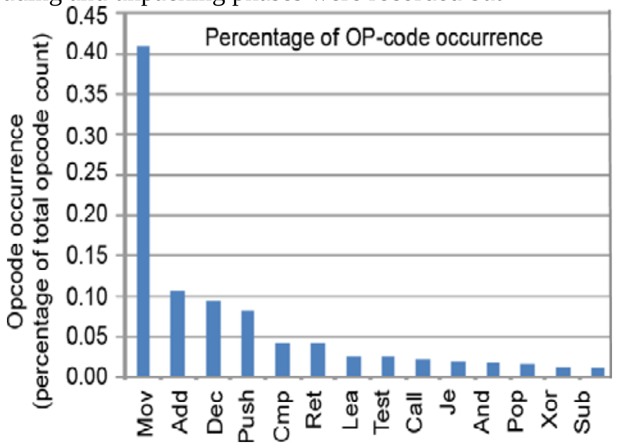


Figure 2: Histogram: Opcode percentage

also that malicious activity occurred, i.e., pop-up, writing to the disk or registry files. While there are 344 Intel opcodes, only 149 different opcodes are recorded during the captured datasets for all programs traced during this experiment. The dataset is normalized by calculating the percentage density of opcodes rather than the absolute opcode count to remove time variance introduced by different run lengths of the various programs. The dataset is sorted into most commonly occurring opcodes as illustrated in Fig. 2. An initial assessment of the data shows two key properties a) The distribution of the various opcodes does not conform to any consistent distribution shape; rather opcode distribution varies greatly as illustrated by the difference between the *mov* and *ret* opcodes: 'Area of Intersect'. Therefore, no one data shape could be assumed and hence a nonparametric method should be used. b) The data values are a percentage of the opcodes within a particular program. For example, 0 means that the opcode does not occur within that program trace or 0.25 means that 25% of the program trace comprises of that opcode. To improve the performance of the SVM the data is linearly scaled.

5. Support Vector Machine

Support Vector Machine (SVM) is a technique used for data classification and was introduced by Boser *et al.* in 1992 [16] and is categorized as a kernel method. The kernel method algorithm depends on dot-products function, which can be replaced by other kernel functions that map the data into a higher dimensional feature space. This has two advantages: Firstly, the ability to generate a nonlinear decision plane and secondly, allows the user to apply a classification to data that does not have an intuitive approach i.e., SVM training when the data has a non regular or unknown distribution [17]. The dataset consists of 149 different opcodes, each having their own unique distribution characteristics and therefore a SVM is an appropriate choice. As mentioned earlier, the data is linearly scaled to improve the performance of the SVM. The main advantages of scaling are a) it avoids attributes with greater numeric ranges dominating those with smaller numeric ranges and b) it avoids numerical difficulties during the calculation as kernel values usually depend on the inner products of feature vectors, e.g., in the case of the linear kernel and the polynomial kernel, very large attribute values might cause numerical problems [18].

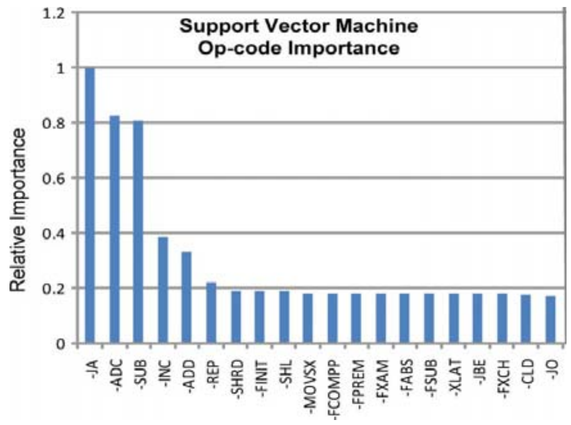


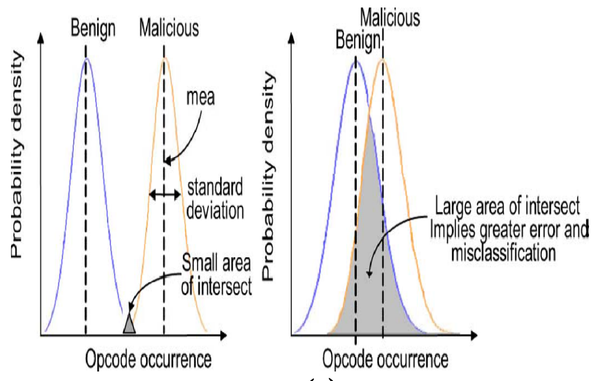
Figure 3: SVM opcode sensitivity

Table 1: Statistics for *ADD* AND *SUB*

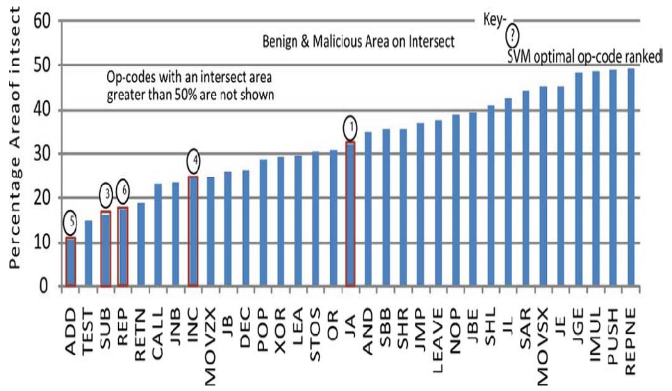
Opcode	Malware		Benign	
	Mean	Variance	Mean	Variance
add	0.00193	0.000285	0.0020	0.000251
sub	0.00783	7.19×10^{-6}	0.00917	15.84×10^{-6}

The RBF (Radial Basis Function) kernel is used as it is considered a reasonable first choice in that it provides a nonlinear mapping of samples into a higher dimensional space. This caters for instances where the relationship between the class label and attributes are nonlinear. SVM is used to create a reference model to validate the filter experiments that are presented in the subsequence sections. The SVM is configured to traverse through the dataset searching for opcodes that have a positive impact on the classification of benign and malicious software. The search starts with six opcodes scanning across the complete data sequence for all unique permutations for that number of opcodes. The search is repeated for five opcodes and then four opcodes. An average of these results is sorted by most occurrences as illustrated in Fig. 3, which show the most important opcodes as chosen by the SVM. Only unique opcodes are selected for eachSVM classification test and no duplicates of repeated opcode patterns are processed.

Key points to note are:
1) The 6 opcodes *ja*, *adc*, *sub*, *inc*, *add* and *rep*, each having an importance rating of more than 20% of the peak detection rate, are selected as the most important indicators for classifying benign and malicious software.
2) *mov* has a negative impact on the classification and identification of software. i.e., when *mov* is part of the analysis data the output/classification is always incorrect. The *mov* has a high density (30% [2] and 40% in the presented dataset) in both benign and malicious software.
3) Packing and encrypting malware is fundamentally a loop that performs fetch, compute and store. The compute, in this context, is the act of encrypting or deciphering the code and is commonly performed by opcodes such as: *xor*, *add/sub* or *rol/ror*. Despite the fact that *xor* is found at the heart of many of these loops, it is not highlighted as an indicator of malware. The *xor* opcode is a versatile operation and we assume that either a) it is used in equal measures in benign programs or b) that insufficient malware programs use the *xor* opcode to produce a reliable feature. However, opcodes *add/sub*, which work in tandem are highlighted by the SVM. The low order statistics in Table I show little



(a)



(b)

Fig. 4. (a) Ideal characteristics. (b) Benign and malicious area of interest.

difference between the population density and variance of malicious and benign software and therefore it would be highly speculative to infer any meaning to *add* and *sub*.

6. Conclusion

This paper, proposes the use of SVM as a means of identifying malware. It shows that malware, that is packed/encrypted, can be detected using SVMs and by using the opcodes chosen by the SVM as a benchmark, determined a prefilter stage using eigenvectors that can reduce the feature set and therefore reduce the training effort. The results presented in this paper exposed three key points. Firstly, the identification of a high population opcode: *mov* that is not only is a poor indicator of benign/malicious software, but inhibits the ability to correctly classify software when used with other opcodes such as *ja*, *adc*, *sub*, *inc*, *add* and *rep*. Secondly, a subset of opcodes can be used to detect malware. However, the SVM analysis demonstrates that *ja*, *adc* and *sub* are strong indicators of malware as they are four times more likely to be used in the correct classification of malware than the next most significant opcodes *(inc)*. Several opcodes have been identified as potential indicators of malware, which provides the basis for an improvement in detection techniques beyond current state of the art [22]. Finally, using the 'eigenvector' prefilter, irrelevant features are safely removed by dataset.

References

[1] A. Lakhotia, E. U. Kumar, and M. Venable, "A method for detecting obfuscated calls in malicious binaries," *IEEE Trans. Software Eng.*, vol. 31, no. 11, pp. 955–968, Nov. 2005.

[2] D. Bilar, "Opcodes as predictor for malware," *Int. J. Electron. Security Digital Forensics*, vol. 1, no. 2, pp. 156–168, 2007.

[3] D. Bilar, "Callgraph properties of executables and generative mechanisms," *AI Commun., Special Issue on Network Anal. in Natural Sci. and Eng.*, vol. 20, no. 4, pp. 231–243, 2007. [4] I. Santos, Y. K. Penya, J. Devesa, and P. G. Garcia, "N-grams-based file signatures for malware detection," *S3Lab, Deusto Technological Found.*, 2009[Online].

[4] Available: pgbg@technologico.deusto.es

[5] R. Sekar, M. Bendre, D. Bollineni, and Bollineni, R. Needham and M. Abadi, Eds., "A fast automaton-based method for detecting anomalous program behaviors," in *Proc. 2001 IEEE Symp. Security and Privacy, IEEE Comput. Soc.*, Los Alamitos, CA, USA, 2001, pp. 144–155.

[6] W. L. K. Wang, S. Stolfo, and B. Herzog, "Fileprints: Identifying file types by n-gram analysis," in *Proc. 6th IEEE Inform. Assurance Workshop*, Jun. 2005, pp. : 64–71.

[7] I. Santos, F. Brezo, J. Nieves, Y. K. Penya, B. Sanz, C. Laorden, and P. G. Bringas, "Opcode-sequence-based malware detection," in *Proc. 2nd Int. Symp. Eng. Secure Software and Syst. (ESSoS)*, Pisa, Italy, Feb. 3–4, 2010, vol. LNCS 5965, pp. 35–43.

[8] I. Santos, F. Brezo, B. Sanz, C. Laorden, and Y. P. G. Bringas, "Using opcode sequences in single-class learning to detect unknown malware," *IET Inform. Security*, vol. 5, no. 4, pp. 220–227, 2011.

[9] I. Santos, F. Brezo, X. Ugarte-Pedrero, and Y. P. G. Bringas, "Opcode sequences as representation of executables for data-mining-based unknown malware detection," *Inform. Sci.*, 2011 [Online]. Available: http://dx.doi.org/10.1016/j.ins.2011.08.020

[10] A. Shabtai, R. Moskovitch, C. Feher, S. Dolev, and Y. Elovici, "Detecting unknown malicious code by applying classification techniques on opcode patterns," *Security Informatics*, vol. 1, pp. 1–22, 2012.

[11] R. Moskovitch, C. Feher, N. Tzachar, E. Berger,M.Gitelman, S.Dolev, and Y. Elovici, "Unknown malcode detection using opcode representation," in *Proc. 1st Eur Conf. Intell. and Security Informatics (EuroISI08)*, 2008, pp. 204–215.

[12] Y. Song, M. Locasto, and A. Stavro, "On the infeasibility of modeling polymorphic shellcode," in *Proc. ACM Conf. Computer and Commun. Security*, 2007, pp. 541–551.

[13] C. Kolbitsch, P. M. Comparetti, C. Kruegel, E. Kirda, X. Zhou, and X. Wang, "Effective and efficient malware detection at the end host," in *Proc. 18th Usenix Security Symp.*, 2009, pp. 351–366.

[14] P. Ferrie, The ultimate anti debugge reference May 2011 [Online]. Available: http://pferrie.host22.com/papers/antidebug.pdf

[15] X. Chen, "Towards an understanding of anti-virtualization and anti debugging behavior in modern malware," *ICDSN Proc.*, pp. 177–186, 2008.

[16] B. E. Bernhard, G. M. Isabelle, and V. N. Vladimir, H. Haussler, Ed., "A training algorithm for optimal margin classifiers," in *Proc. 5th Ann. ACM Workshop on COLT ACM Press*, Pittsburgh, PA, USA, 1992, pp. 144–152.

[17] C. Ko, M. Ruschitzka, and K. Levitt, "Execution monitoring of security- critical programs in distributed systems: A specification-based approach," in *Proc. 1997 IEEE Symp. Security and Privacy*, Oakland, CA, USA, May 1997, p. 175-1 87.

[18] C.-W. Hsu, C.-C. Chang, and C.-J. Lin, A ractical Guide to Support Vector Classification, Department of Computer Science National Taiwan University, Taipei, Taiwan, Apr. 15, 2010 [Online]. Available: http://www.csie.ntu.edu.tw/

[19] R. Vanderbei, *Linear Programming: Foundations and Extensions Pub.* New York, NY, USA: Springer, 2000,

ISBN: 0792373421.

[20] Matlab Statistics Toolbox Oct. 2011 [Online]. Available: http://www.mathworks.co.uk/help/toolbox/stats/

Author Profile

Mr. Dasu Vaman Ravi Prasad is working as associate professor in Computer Science Engineering from CVSR College of Engineering from Anurag Group of Institutions Venkatapur (V), Ghatkesar(M), Ranga Reddy District, Hyderabad-500088, Telangana State.

Mr. Pagidimarri Venu received the B.Tech degree in Computer Science of Engineering from JNTU Anantapur in 2011 and now pursuing M.Tech. degree in Computer science from CVSR College of Engineering in JNTU Hyderabad