



International Journal of Emerging Technologies in Computational and Applied Sciences (IJETCAS)

www.iasir.net

Ensemble of Soft Computing Techniques for Malware detection

Deepika Veerwal¹, Paranjay Menaria²

^{1,2} (Department of Computer Science & Engineering)

¹ (College of Technology and Engineering, MPUAT), ² (Modern Education Society's College of Engineering)

¹ (951, Gyan Nagar, Sector-4, Hiran magri, Udaipur, Rajasthan (313002))

India

Abstract: Execution of a malware on the system compromises its security. Analysis recommends that the consequences of malware are getting worse. Though we know that malware has had a exceptional significance on the world. The growing number of computer security incidents suggests that malware is contagious. Surfing the internet without antivirus installed and firewall protection enabled should induce any reader of the extensive and malicious nature of malware. For determining malwares, a system known as a malware detector is used which attempts to conclude whether a program has malicious objective. In order to evade uncovering, malware writers (hackers) habitually use obfuscation to morph malware.

To solve the problem of malware detection a number of pattern recognition and machine learning algorithms has been proposed. The paper states the problem of classifier fusion with soft labels for Malware Detection. Performance of Artificial Neural Networks (ANN) and Support Vector Machines (SVM) is presented here. The performance of fusing these classifiers using approaches based on Dempster-Shafer Theory, Average Bayes Combination and Neural Network is proposed. We also investigate extracting network level features for malwares and analyze the performance of classifiers when these features are included.

Keywords: Artificial neural network, Decision fusion, Malware analysis, Malware detection, Multiple classifier fusion.

I. Introduction

Malware is known by several names such as malicious code (MC), malcode and malicious software. Various definitions have being presented for describing malwares. [1] states that a malware instance is a program whose aim is malevolent. [2] describes malicious code as any code changed, added or deleted from the system in order to deliberately cause harm or subvert the projected function of the system. We adopt the explanation given by [3], which defines malware as a generic term which comprises Spywares, Viruses, Trojans, and other intrusive codes.

Malware (i.e. malicious software) sneak into user's computers, collect its private information, wreck havoc on the Internet and cause millions of rupees in harm. Astonishingly, even software provided by trustworthy vendors may include code that performs objectionable actions which may infringe user's privacy. For example, Google Desktop, an admired local file system search tool, in reality sends susceptible user information back to Google's servers in definitive configuration settings [4]. In another generally publicized example, Sony Media Player install a root kit without intimating the users in order to enforce copyright boundaries and sends back user's music listening habits. Malware analysis and detection is a difficult task, and present malware analysis and detection techniques habitually fall short and fail to sense many new, unknown malware samples.

A. Malware Detector

A malware detector is the execution of some malware detection techniques. Malware detector aims to protect the system by exploring malicious behavior. It is not necessary to reside malware detector on the same computer to which it is trying to protect.

Malware detection techniques can be classified generally into two categories: signature based detection and anomaly based detection. Signature based detection techniques works according to the description present in the database for detecting the maliciousness of a program under assessment. Though one may visualize the description and the pre defined signature of the malicious behavior, a key to the effectiveness of the signature based method. An anomaly based detection technique works on the knowledge for constituting normal activities in order to decide the maliciousness of a program under assessment. Specification based detection technique falls under the anomaly based detection technique and it is one of the special techniques. This technique leverages some rule set or specification for deciding the valid behavior in order to obtain the malicious program under assessment. Programs deviating from the specification are considered anomalous and generally malicious. All the three detection techniques can employ one of the three different approaches: static,

dynamic, or hybrid. The particular approach of signature based and anomaly based technique is determined by the way these techniques gather information for detecting malware. Static analysis uses structural properties or syntax of the program/process under assessment for determining its maliciousness. For example, signature based detection including static approach would only leverage structural information (such as sequence of bytes) for determining the maliciousness, whereas in dynamic approach it will leverage runtime information (such as systems seen on a runtime stack). In general, a static approach tries to identify malware before the program executes. On the contrary, a dynamic approach tries to detect malware or malicious behavior during or after the program execution. In hybrid techniques, static and dynamic approaches are combined [5]. Here, both static and dynamic information are used for detecting malware. Numerous issues make the signature based method less reliable as it cannot deal with the code obfuscations and unable to detect formerly unseen malware (novel attacks).

Malware writers (hackers) employ various methods for hiding the signature of their malicious creations such as code obfuscation techniques and software encryption tools. Garbage insertion is one of the example of these obfuscation algorithms which contains the addition of the sequences which do not modify the program's behavior (such as nop instructions); code reordering is another example which change the order of the program instructions and rename the variables; which in turn replaces a variable identifier with the other one [6].

We propose a malware detection using n-gram based approach on API call sequences. Our work is based on the assumption that an original malware M contains a sequence of malicious API calls S . A variant of the original malware that is obtained by obfuscation retains the functionality of the original malware and contains a set of system calls S' . The problem is to locate the similarity measure between S and S' . Since the original functionality is preserved, we assume that the variation between S and S' would not be very large. A nonlinear classifier that has learnt to classify a program with system calls sequence S as malware will also classify S' as malware if the difference between S and S' is not much.

In rest of the paper, a concise introduction to the similar work in the area of malware detection is presented in section 2. A brief introduction to the proposed work is presented in section 3. In section 4 we present the experimental results of n-gram, ANNs, SVMs, and their ensemble. In section 5, we conclude our results.

II. Related Work

Numerous approaches have been presented by the research society to counter these obfuscation techniques. [7] presented a method for calculating the similarity by concentrating on the degree of similarity between two executables within syscall (API) sequences. Because of inability of this approach for detecting malwares it offered unsatisfactory performance though maintaining a low FPR (false positive ratio). Various other approaches have been rooted on Control Flow Graph (CFG) analysis. [8] introduced an example as a part of the MCF (Malicious Code Filter) project. In this method a program had been sliced into blocks whereas viewing for tell tale signs (such as operations that alter the state of a program for eg. network access events and file operations) for determining an executable to be malicious.

[9] proposed a number of methods to disassemble binary executables, in order to build a representation of binary executable's execution and enhanced a program's slices into sequences of instructions also known as idioms. Work of [10], who presented a method rooted on CFG analysis for handling obfuscations in malicious software. Later, [1] enhanced the work by including semantic templates of malicious specifications.

Other than the above described approaches, two other approaches have been introduced to deal with unknown malware in which classic signature method failed: anomaly and data mining based detectors. Anomaly detectors use information fetched from benign software to acquire a benign behavior profile. Then, every significant deviation from this profile is qualified as suspicious. [11] proposed the fileprint (or n-gram) analysis in which a model try to distinguish several file types on their structural (byte) composition. The main assumption behind this analysis is that benign files are composed of predictable regular byte structures for their respective types. Likewise, [12] used byte sequence frequencies to detect malware. Their goal was to use only information about benign software to measure the deviations from the benign behavior profile. Specifically, they applied a Gaussian Likelihood Model fitted with Principal Component Analysis (PCA). Unfortunately, these methods usually have a high false positive ratio that renders them difficult for commercial antivirus vendors to adopt.

Data mining based approaches rely on datasets that include several characteristic features of both malicious samples and benign software to build classification tools that detect malware in the wild (i.e., undocumented malware). To this end, authors in [13] were the first to implement data mining models for detecting different malicious programs rooted on their respective binary codes. Precisely, they applied a number of classifiers to three different feature extraction approaches: program headers, string features and byte sequence features. Later, [14] improved the results obtained by Schulz *et al.* by using n-grams (i.e., by overlapping byte sequences) in place of non-overlapping sequences. This method used several algorithms and the best results were achieved with a Boosted Decision tree. In a similar vein, substantial research has concentrated on data

mining and n-gram classification of byte sequences [15, 16, 17, 18]. Still, most of these methods are limited as they count certain bytes in the malware body; as the majority of the general transformations function at the source level and hence these detection methods were not satisfactory [19]. These approaches were also incorporated in [20] for detecting packed executables. Authors of [20] proposed their first approach based on (i) the extraction of some features from the Portable Executable (PE), e.g., the number of standard and non-standard sections, the number of executable sections, the entropy of the PE header; and (ii) the classification through machine-learning models, e.g., Naive Bayes, J48 and MLP. Later, [21] developed a rapid statistical malware detection tool by evolving above described methods. For combining results with the earlier MLP based classifier, they added innovative n-gram-based classifiers. Authors of [22] developed a model rooted on the frequency of the arrival of opcode sequences. In addition to this they depicted a technique for mining the significance of each opcode and assessing the frequency for each opcode sequence.

III. Proposed Work

Presently malware is known by rich and versatile behavior, even though large families of malware (variants of the Allaple worm) exhibit common behavioral patterns, such as acquiring and locking of distinctive mutexes on infected systems. These shared patterns can be exploited by using machine learning techniques and designing a method which is capable enough to classify malwares and benign programs automatically based on their behavior. Learning approach is outlined by the following fundamental steps: Data acquisition, Behavior Monitoring, Feature Extraction and Learning and Classification. In data acquisition corpus of malware binaries presently spreading in the wild is kept together through a variety of techniques, for e.g. honeypots and spam-traps. For identification of known malware instances and for enabling the learning and consequent classification of behavior, antivirus engine is applied. Behavior Monitoring executes the malware binaries and monitor them in a sandbox environment. The change of state in API function calls are the base for the generation of behavior based analysis report. Features reflecting behavioral patterns, for e.g. opening a file, locking and unlocking of a mutex and locating a registry key, are extracted with the help of analysis reports which are useful in embedding the malware behavior into a high dimensional vector space. Machine learning techniques are used for determining the behavior of malwares. Lastly, a classifier is constructed for detecting malware binaries and applied on different testing data.

A. Methodology

The goal of this study was to assess the feasibility of detecting unknown malicious code based on the computers behavior (measurements), using machine learning techniques, and the potential accuracy of such methods. In order to create the datasets we use Sandboxing Technique. A sandbox is known as a mean of security mechanism used for separating running programs and executing untrusted programs or untested code, from unverified suppliers, third-parties, untrusted websites and untrusted users [23]. Sandboxes are a explicit example of virtualization. We have used Cuckoo Sandbox to analyze the programs. Cuckoo Sandbox is an open source malware analysis system. Many results are generated when a program is analyzed by Cuckoo Sandbox. Amongst them, the one that is useful to us is the WIN32 API calls and the captured network traffic trace [24]. Figure 3.1 shows the methodology we used in order to perform this study.

B. Feature Extraction and Selection: n-gram Analysis

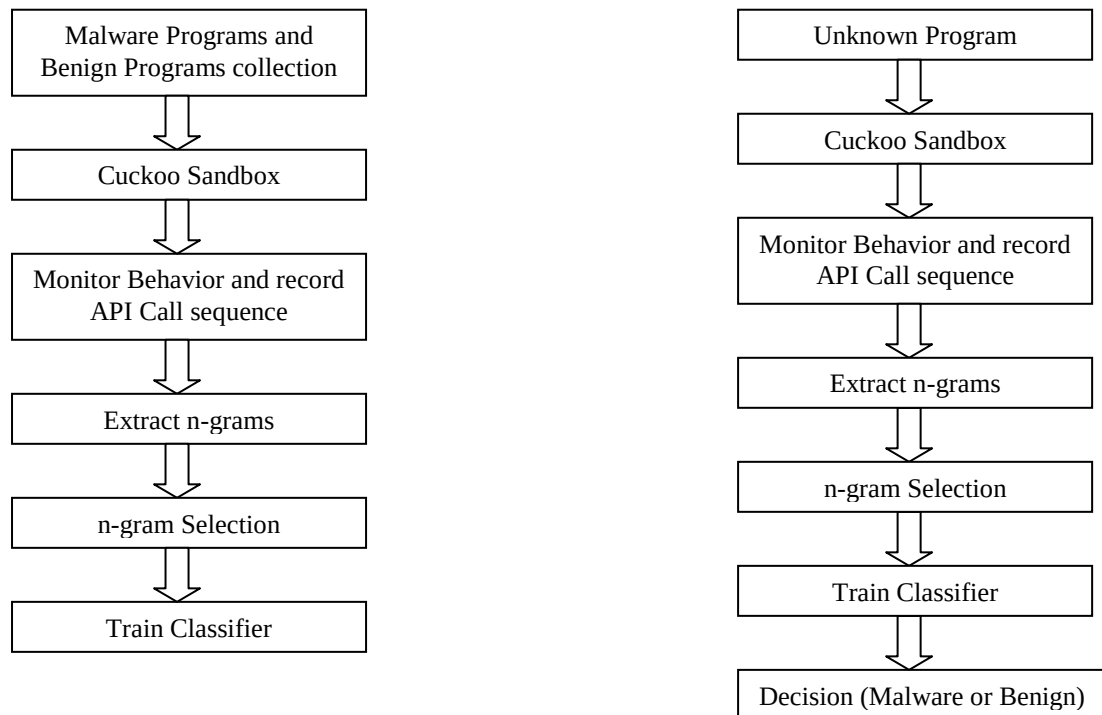
The sandbox discussed above outputs a log file with the API calls and the corresponding parameters for all the API calls ordered in the sequence they were called. This log file was reduced to contain only the API call sequence. An n-gram method is applied for extracting data features and used in document classification and natural language processing as it is good at acquiring the statistics of substrings of length n as well as implicit features of data. In our method, system call n-grams are overlapped substrings collected by a sliding window. Different lengths of grams can be set to obtain different lengths of features. One problem in the n-gram method is that the number of n-grams generated from system call streams is very large, which reduces the direct applicability of many classification techniques. Thus, selecting n-grams that are effective for malicious code detection is a necessity. This paper uses classwise document frequency, common ngram frequency and information gain techniques to select features.

Information gain is the reduction of entropy (or uncertainty) caused by partitioning a collection of examples according to a feature. It is a measure of the effectiveness of a feature in classifying examples.

One risk of the information gain approach is that most of the selected features may belong to only one class, which means the features of another classes are ignored. To overcome this problem, another feature selection approach called the document frequency approach is adopted; this approach separately selects features from benign streams and malicious streams.

Another feature selection method frequently encountered in ngram approach for document classification is classwise ngram frequency.

Figure 3.1: Methodology



C. Network Level features

API call sequence captures only the kernel level operating system features. Malwares like worms, adware and spyware actually depend extensively on network to spread or communicate stealthily. So it may be argued that including network level features for classification should improve the performance. This train of thought led us to monitor the program's network behavior along with the system level API monitoring. Cuckoo Sandbox outputs a traffic trace (pcap) file which contains packets sent by the virtual machine for the entire duration the program was being analyzed in the sandbox. All network based programs running in the operating system (like windows update, file sharing etc) were turned off. Using tshark and BRO-IDS we calculated the following features. BRO-IDS was used to aggregate packet statistics to flow level statistics for different protocols. tshark was used to calculate global features from the packet capture (pcap) file. The change in classification capabilities of the classifiers in presence and absence of these features are also studied.

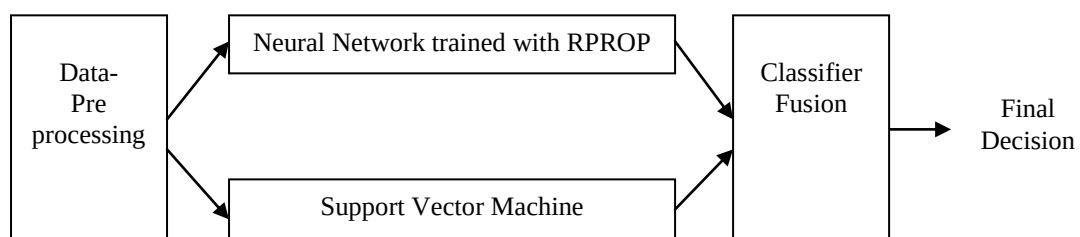
D. Classifier fusion

Using the features discussed we designed two classifiers. One using Neural Network and the other using Support Vector Machines. Then these Classifiers are fused using DST, Bayes method and Neural Network. Figure 3.2 shows how this has been achieved.

E. Dataset description & experimental setup

We amassed a set of 351 malwares from [25] which included spyware, virus, Trojan, adware and worms. We also collected 105 benign programs from [26] and windows directory. Out of these 75 benign programs and 246 malwares were used to train various classifiers. As proposed in our methodology we collected the API calls of all the programs and from these extracted n-grams for $n=2,3,4,5,6$. We had selected 200 n-grams by the three methods. Cuckoo Sandbox also gives a traffic capture file (pcap) which contains all the packets sent during the execution of the malware. Two different sets of experiments were carried out using this data. In one set of experiments we only used features comprising of the n-grams. The best combination of n-gram length and feature selection algorithm was obtained. We used this combination and appended the network level features and analyzed the performance of the resulting dataset.

Figure 3.2: Classifier Fusion



IV. Experimental Results

We make use of ROC (region of operating curve) to compare performances of classifiers and feature sets. We here show ROC of all possible combinations of feature reduction methods (classwise document frequency, common ngram frequency and information gain) of n-gram length (n=2,3,4,5,6). The classifier performance is quantified by the area under ROC. Larger areas mean better performances.

Figure 4.1: roc curve of classifiers with 2-grams and feature reduced using classwise document frequency

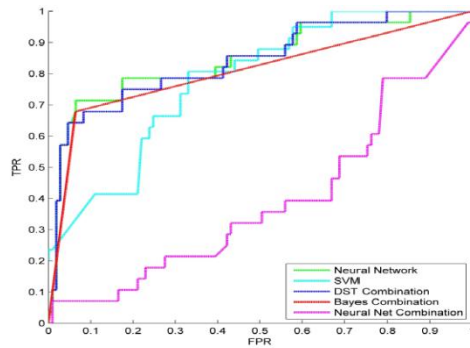


Figure 4.2: roc curve of classifiers with 2-grams and feature reduced using common ngram frequency

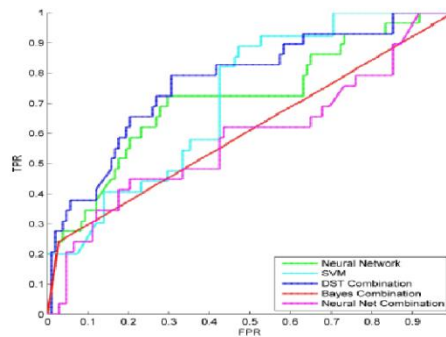


Figure 4.3: roc curve of classifiers with 2-grams and feature reduced using information gain

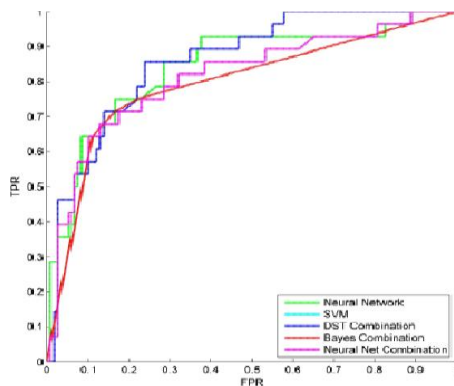


Figure 4.4: roc curve of classifiers with 3-grams and feature reduced using classwise document frequency

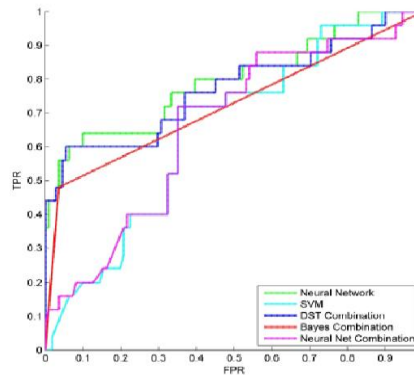


Figure 4.5: roc curve of classifiers with 3-grams and feature reduced using common ngram frequency

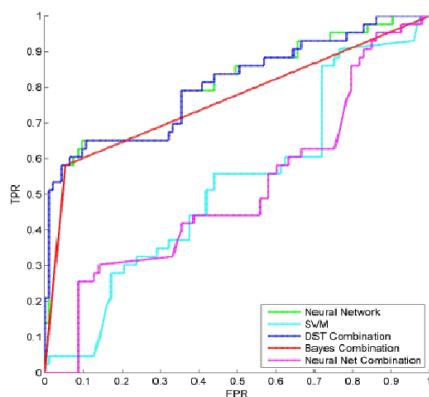


Figure 4.6: roc curve of classifiers with 3-grams and feature reduced using information gain

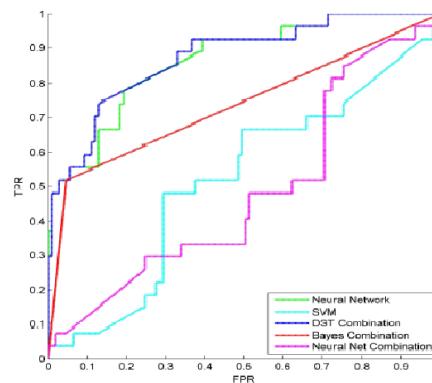


Figure 4.7: roc curve of classifiers with 4-grams and feature reduced using classwise document frequency

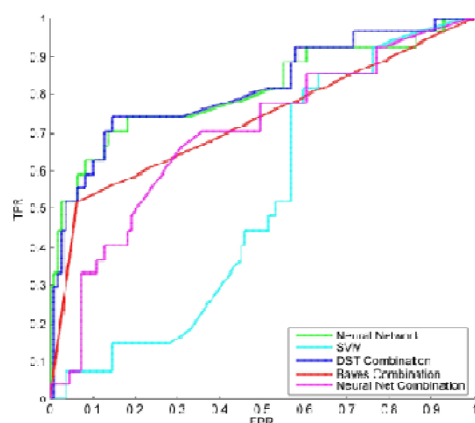


Figure 4.8: roc curve of classifiers with 4-grams and feature reduced using common ngram frequency

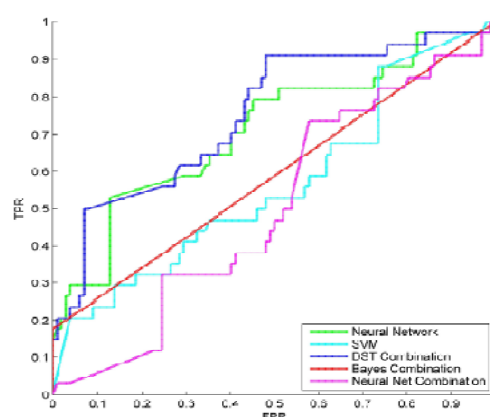


Figure 4.9: roc curve of classifiers with 4-grams and feature reduced using information gain

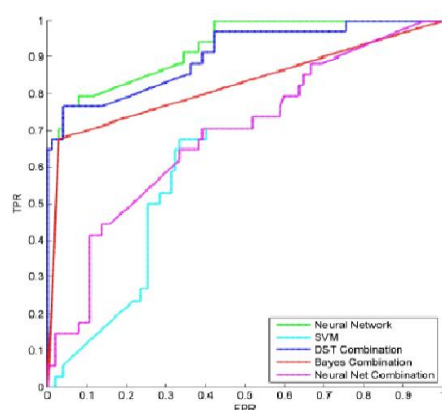


Figure 4.10: roc curve of classifiers with 5-grams and feature reduced using classwise document frequency

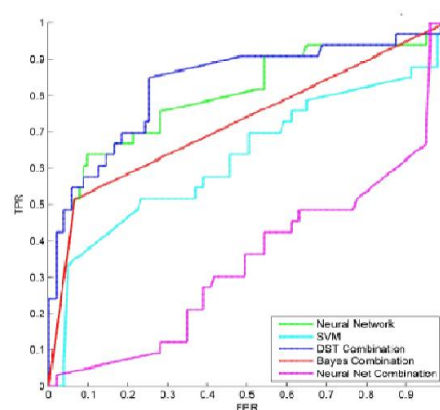


Figure 4.11: roc curve of classifiers with 5-grams and feature reduced using common ngram frequency

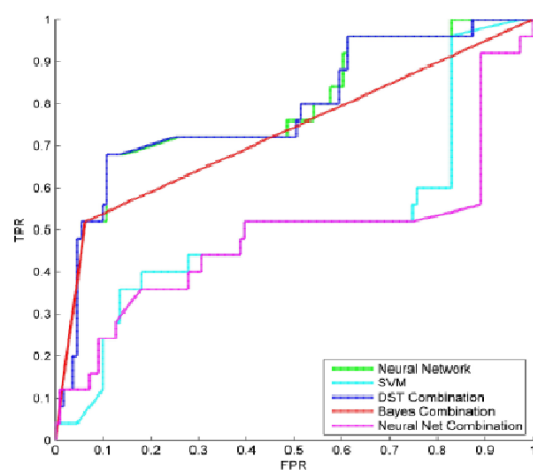


Figure 4.12: roc curve of classifiers with 5-grams and feature reduced using information gain

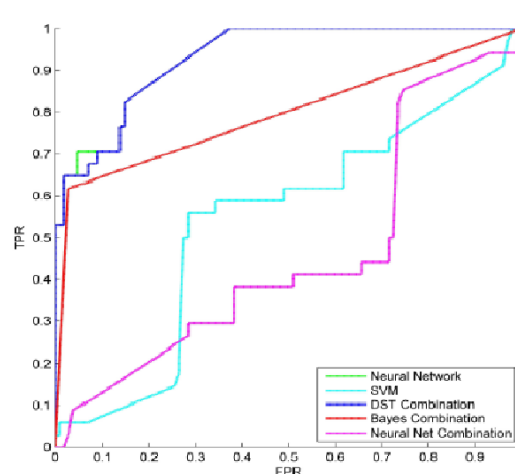


Figure 4.13: roc curve of classifiers with 6-grams and feature reduced using classwise document frequency

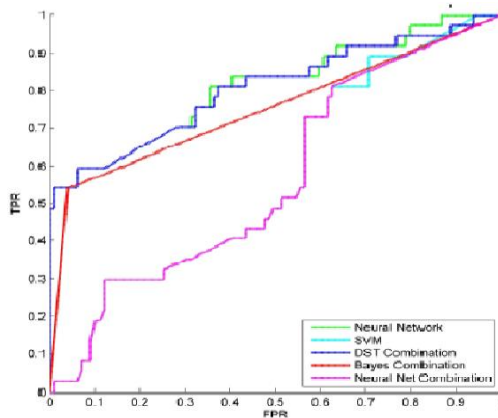


Figure 4.14: roc curve of classifiers with 6-grams and feature reduced using common ngram frequency

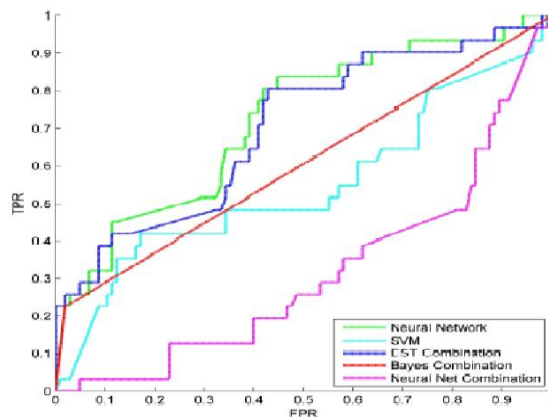
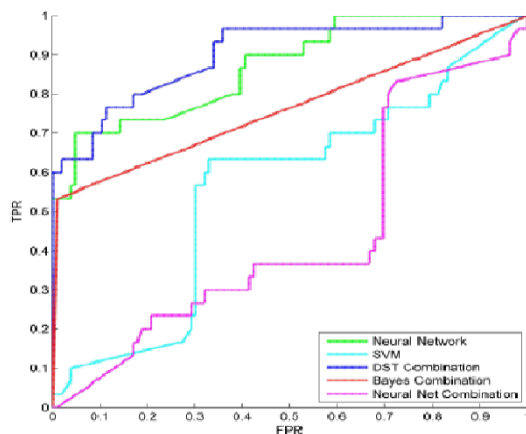


Figure 4.15: roc curve of classifiers with 6-grams and feature reduced using information gain



A. Effect of n -gram Length

We find the performance of classifiers trained using 5-grams to be the best for all three methods. 6-gram is close to second best in performance and hence it is acceptable while 2-grams is the worst performer. It can be argued that for n -grams with $n=2,3,4$ are not able to capture enough information required for the classifiers to make a demarcation between malware and benign files. While for $n=6$ the feature suffers the very famous curse of higher dimensionality.

To show the performance of Malware detection system, a confusion matrix is created. It consists of four outcomes namely TP, TN, FP, and FN. TP and TN are the correct prediction that the program is normal and malicious respectively. FP and FN are the incorrect prediction that the program is normal and malicious respectively. Based on this, we use the measures shown in Table 1 to quantify the performance of Malware detection system. A good Malware detector will have low FPR and high TPR.

Table 4.1: Confusion matrix

Predicted Positive	True Positive (TP)	False Positive (FP)
Predicted Negative	False Negative (FN)	True Negative (TN)
	Actually Positive	Actually Negative

Table 4.2: Performance measures

Measure	Formula
True Positive Rate (TPR)	$TP/(TP+FN)$
False Positive Rate (FPR)	$FP/(FP+TN)$
Detection Rate (DR)	$(TP+TN)/(TP+TN+FN+FP)$

- **TPR** is the ratio of positive cases correctly identified to the actually positive cases.

- **FPR** is the ratio of negatives cases incorrectly classified as positive to the actually negative cases.
- **DR** is the proportion of total number of correct predictions.

Table 4.3: Performance with network level features

	TPR	FPR	DR
NN	0.972804	0.425874	0.929152
SVM	0.921498	0.365806	0.932872
DST	0.982247	0.362352	0.99691
BAYES	0.969469	0.375917	0.950901
NN_Comb	0.915232	0.410973	0.89189

Table 4.4: Performance without network level features

	TPR	FPR	DR
NN	0.963543	0.429273	0.923242
SVM	0.8521	0.369837	0.87
DST	0.980392	0.370298	0.932897
BAYES	0.970588	0.382353	0.953472
NN_Comb	0.888235	0.417647	0.898126

Table 4.3 and 4.4 shows the performance with and without using the network level features. We used ngram with n=5 and selected top 200 ngrams using information gain. Then a set of 17 features were also appended to the selected 200 ngrams and each input instance had 217 features. After that the five classifier strategies studied were applied on them. From the table we can see there is no notable change in classification capabilities of Neural Networks, Bayes fusion and NN-COMB. But for svm we have a very stark difference. We can see an almost 5% improvement in all the three performance measures. In DST based fusion we see an increase in the DR. We can safely conclude that inclusion of network specific features is never detrimental. It may either show same performance as earlier or better performance than that using only ngrams.

V. Conclusion

The proposed methodologies evidently shown the significance of using ensemble approach based on distinct feature representation for modeling Malware detection system. Malware classification can be enhanced by including features other than API ngrams. Malware exhibiting network behavior gets detected easily when the network level features are also included. We find the performance of classifiers trained using 5-grams to be the best for all three methods. 6-gram is close to second best in performance and hence it is acceptable while 2-grams is the worst performer. Information gain method of selecting relevant ngrams seems to always perform better than the other two methods. Classwise document frequency always out performs common ngram frequency method. We have seen in almost all cases that the performance of Neural Network and SVM fused using DST is the best. Neural Network is close to second. Neural Network Combination and SVMs have poorest performances in all cases. The above discussion leads to the conclusion that to detect malwares in the best possible way from our limited dataset we must use ngram with n=5 and select the top 200 ngrams using information gain criteria. We also required to see the effect of inclusion of network features on classification results.

VI. References

- [1] M. Christodorescu, S. Jha, S.A. Seshia, D. Song, and R.E. Bryant, "Semantics-aware malware detection," *Proc. of the IEEE Symposium on Security and Privacy*, 2005, pp. 32-46.
- [2] G. McGraw and G. Morrisett, "Attacking malicious code: a report to the infosec research council," *Software IEEE*, vol. 17, 2000, pp. 33-41.
- [3] A. Vasudevan and R. Yerraballi, "Spike: Engineering malware analysis tools using unobtrusive," *Proc. of the 29th Australasian Computer Science Conference*, 2006.
- [4] H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda, "Panorama: Capturing System-wide Information Flow for Malware Detection and Analysis," *Proc. of the 14th ACM Conf. on Computer and Communication Security*, 2007, pp. 116-127.
- [5] J. C. Rabek, R. I. Khazan, S. M. Lewandowski, and R. K. Cunningham, "Detection of injected, dynamically generated, and obfuscated malicious code," *Proc. of the ACM workshop on Rapid Malcode*, 2003, pp. 76-82.
- [6] M. Christodorescu and S. Jha, "Testing malware detectors," *Proc. of the ACM SIGSOFT international symposium on Software testing and analysis*, New York, 2004, pp. 34-44.
- [7] J.Y. Xu, A.H. Sung, P. Chavez, and S. Mukkamala, "Polymorphic malicious executable scanner by api sequence analysis," *Proc. of the 4th International Conference on Hybrid Intelligent Systems*, 2004, pp. 378-383.
- [8] R. W. Lo, K. N. Levitt, and R. A. Olsson, "MCF: a malicious code filter," *Computers Security*, vol. 14, 1995, pp. 541-566.

- [9] J. Bergeron, M. Debbabi, M. M. Erhioui, and B. Ktari, "Static analysis of binary code to isolate malicious behaviors," *Proc. of the 8th IEEE Workshop on Enabling Technologies on Infrastructure for Collaborative Enterprises*, 1999, pp. 184-189.
- [10] M. Christodorescu and S. Jha, "Static Analysis of Executables to Detect Malicious Patterns," *USENIX Security '03*, vol. 8, 2003, pp. 169-186.
- [11] W.J. Li, K. Wang, S. Stolfo and B. Herzog, "Fileprints: identifying file types by n-gram analysis," *Proc. of the 6th Annual IEEE Systems Workshop on Man and Cybernetics (SMC) Information Assurance*, 2005, pp. 64-71.
- [12] A.D.M. Cai, J. Theiler, and M. Gokhale, "Detecting a malicious executable without prior knowledge of its patterns," *Proc. of the SPIE Conf. on Data Mining, Intrusion Detection, Information Assurance, and Data Networks Security*, 2005, pp. 1-12.
- [13] M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," *Proc. of the IEEE Symposium on Security and Privacy*, 2001, pp. 38-49.
- [14] J. Z. Kolter and M. A. Maloof, "Learning to detect malicious executables in the wild," *Proc. of the 10th ACM SIGKDD Conf. on Knowledge discovery and data mining*, New York, 2004, pp. 470-478.
- [15] R. Moskovitch, D. Stopel, C. Feher, N. Nissim, and Y. Elovici, "Unknown malware detection via text categorization and the imbalance problem," *Proc. of the IEEE Conf. on Intelligence and Security Informatics*, 2008, pp. 156-161.
- [16] M. Z. Shafiq, S. A. Khayam, and M. Farooq, "Embedded malware detection using markov n-grams," *Proc. of the 5th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Berlin, 2008, pp. 88-107.
- [17] Y. Zhou and W. M. Inge, "Malware detection using adaptive data compression," *Proc. of the 1st ACM Workshop on AISec '08*, New York, 2008, pp. 53-60.
- [18] I. Santos, Y. K. Peña, J. Devesa, and P. G. Bringas, "N-grams-based file signatures for malware detection," *Proc. of the 11th International Conference on Enterprise Information Systems (ICEIS)*, 2009, pp. 317-320.
- [19] M. Christodorescu, "Behavior-based malware detection," University of Wisconsin, Madison, USA, Phd, 2007.
- [20] R. Perdisci, A. Lanzi, and W. Lee, "Classification of packed executables for accurate computer virus detection," *Pattern Recognition Letters*, vol. 29, 2008, pp. 1941-1946.
- [21] R. Perdisci, A. Lanzi, W. Lee, "Mcboost: Boosting scalability in malware collection and analysis using statistical classification of executables," *Computer Security Applications Conference*, 2008, pp. 301-310.
- [22] I. Santos, F. Brezo, X.U. Pedrero, and P. G. Bringas, "Opcode sequences as representation of executables for data-mining-based unknown malware detection," *Information Sciences*, 2011.
- [23] I. Goldberg, D. Wagner, R. Thomas, and E. A. Brewer, "A secure environment for untrusted helper applications (confining the wily hacker)," *Proc. of the 6th USENIX UNIX Security Symposium*, California, 1996.
- [24] "www.cuckoo.org," 2013.
- [25] "www.safegroup.pl," 2013.
- [26] "www.sourceforge.net," 2013.